FMvet

Visualizing Multiple Layers of the Smart Diagnostics System at ASML

Verploegen, Cas

Award date:
2021

Department of Mathematics and Computer Science
Visualization Group

Failure Mode visual exploration tool

FMvet: Visualizing Multiple Layers of the Smart Diagnostics System at ASML

Master Thesis

Cas Verploegen

Supervisors
Prof. dr. ir. Jack van Wijk (TU/e)
Dr. ir. Peter van den Hamer (ASML)
Dr. ir. Wouter Meulemans (TU/e)

Eindhoven, August 9, 2021

Abstract

To keep ASML's chipmaking machines at the forefront of technology and to stay ahead of their competitors, they continuously look for ways to improve machine design and internal processes. One way to achieve that is to utilize and enhance generated machine data to its full extent, to diagnose and repair equipment failures in near real-time. The number of diagnoses, failures, and symptoms are high, making it difficult for experts to get an overview of how well the diagnostic system is performing. Ultimately there is one optimal diagnosis and repair procedure to follow when a failure occurs, but in the field this is not always the case. Improving the diagnostic quality (one fitting solution for one failure) and diagnostic hit rate (all failures should have a solution) is the goal. In this work, FMvet is introduced as a tool for visual data exploration. It shows the diagnostic gaps, visualizes overlap between diagnoses and displays the underlying symptoms. A prototype was built through an iterative design process, with a focus group of experts consulted at every step. The tool was put to the test by organizing a domain expert review, which resulted in positive feedback and pointers to future work. Overall, FMvet performs well for the initially supported activities, and visualization has potential to support more types of diagnostic work in the future.

Preface

I would like to thank the two main parties involved in this project, namely TU/e and ASML for their offered opportunities and great collaboration. Most notably I thank my direct supervisors for their input towards the project and feedback on all facets of the work, Jack van Wijk (TU/e) and Peter van den Hamer (ASML). Next to that many other people from both the Visualization Research Group (TU/e) and colleagues from various teams at ASML were of importance by providing their educated thoughts, ideas and bits of criticism. The FMkb team in particular helped me out by providing valuable insights and new API endpoints. The assistance and feedback was relevant and helpful for the development of the project.

Contents

Chapter 1

Introduction

ASML is one of the central players in a hub of innovation with its headquarters located in Veldhoven, namely the Brainport region in the Netherlands. At ASML, a legion of experts is working to develop the new generation of chipmaking machines. They are providing efficient production facilities and empower the high end field in the semiconductor industry, consisting of foundries and in-house chipmakers. At the customers' fabs hundreds of specialized and advanced machines are running around the clock to manufacture the computer chips of tomorrow. To stay ahead of the competition ASML is always looking to improve its machines and products, which continue to push modern day technology.

One of the ways to keep innovating and improving, is to put the huge amounts of data that can be mined to good use. The term Big Data is of relevance here, since machines all over the world are generating data by the second. To put this data to good use, there is a need to effectively process, filter, analyze and enhance it to gather insights that are valuable for engineers, researchers and developers. However, this is not an easy task and needs multiple iterations to get good results. The data is often fairly unstructured and the bulk of it has limited usefulness. Getting valuable insights can be compared to looking for needles in haystacks. Fortunately, lots of effort has already gone into optimizing the information gathering process and making insights actionable.

A particular use case of the data processing system is smart diagnostics. It is in the interest of ASML and all its customers to reach the maximum uptime possible for each machine. When machines are broken or underperforming, they need to be inspected and fixed as soon as possible. Data can be of benefit in two ways: by learning from previously made errors and by looking for ways to structurally improve the machines. With both comes a need for diagnostic investigation and interference where needed. Furthermore, the data can be used to predict issues in advance and take appropriate measures, known as predictive maintenance. It is one of the goals of ASML to make as many potential problems diagnosable and to have fitting solutions available.

The available diagnostic data is derived from the event logs each machine produces. From those event logs we can find the instances in which the machine was underperforming or not performing at all. The current setup with databases and a variety of tools offer the diagnostic experts ways to investigate present-day problems. These are compared with similar problems that previously occurred and the solutions that were applied and proven effective back then. When no solution exists yet, there is a need to get customer support engineers at work to generate solutions, in case the problem reoccurs in the future. Furthermore, development of new machines is influenced by the gathered insights. All knowledge from the data can be used to create even better machines by design with future generations. Looking for ways to structurally improve the processes and machines in the future is a core task to keep performing at the cutting edge of semiconductor technology.

A specific diagnostic problem that came to light recently is that experts and field engineers do not always have the best knowledge about structure, meaning and contents of the data. For instance, they do not

always have an overview of what faulty machines states often occur without a fitting solution. Next to that, maintaining high levels of data quality is hard with a big team of people working on different parts of the machines. It is often difficult to find out which solutions can be improved, or if similar solutions exist that were created before by another engineer. One idea from the diagnostic experts at ASML is that visualization can be of big help with data exploration and quality control tasks. As a result, this Master project was brought to life to create an application, which became known as the Failure Mode visual exploration tool (FMvet). The aim is to visualize multiple layers of the smart diagnostics system at ASML. In the following problem definition section we discuss what it entails in more detail.

1.1 Problem definition

The ultimate goal of the project is to develop a graphical and interactive visualization tool to show the similarities, differences and overlap between Failure Modes. The tool will be a visual extension to the readily available Failure Mode knowledge base (FMkb), which represents diagnostic data through advanced filtering, data tables and basic visual properties. Failure Modes (FM1s) are used to classify and act on incidents that happen in fabs all over the world. In essence, there is a mapping between incidents of a faulty machine state (Faults) and the higher level diagnosis of the problem (FM1). The aim is to get a better overview of what this mapping looks like and how to improve it. To summarize:

The problem is to visualize diagnostic data, primarily overlap between FM1s (and their Faults), to support users to investigate, maintain and improve the diagnostic quality and hit rate.

Three main use cases were identified that can benefit from visualizations of diagnostic data. First, we want to better inform users on the underlying structure in the contents of the data through visual exploration. When working with large datasets, graphical representations make it easier for the user to dive into interesting cases and is often more powerful than numerical representations. Second, the investigation of FM1 quality and automatic classification methods can be sped up and brought to light. Often data quality issues fly under the radar and cannot be easily detected. Examples of data quality issues that might occur are overlap between FM1s, noise in underlying Symptom data, or unforeseen relations to other Functional Clusters and Subsystems. The third use case is to find gaps in the diagnostic landscape, to find relevant instances where an FM1 has no solution yet (type Other) or only an old solution available (type PCS). A thorough explanation of these concepts is given in Chapter 2 (Background).

We elaborate on this and discuss optimal diagnoses using Figure 1. In the top part the mapping between FM1s and Faults is visualized. When there is a line between the two, it means the Fault is classified as an instance of the FM1. Ideally, one Fault is only associated with one FM1 that provides the fitting solution. However, in the field we often encounter two other situations. In the first one there is no FM1 available yet, which means the issue has not been detected before or has no solution yet. In the second one, there are multiple FM1s related to the Fault. This is also not desired, as now the engineer needs to guess and try which solution is going to work. Both situations cause unnecessary machine downtime and are not desired. In reality, the problem is even worse as the mapping between FM1s and Faults is very chaotic and overlaps in most cases. Next, we consider the bottom visual, which introduces the FM1 types used in the diagnostics system.

The focus of this project is on visualizing the mapping between FM1s and Faults, which is outlined with a red border. The task for the domain experts is to put this visual information to use and get the data to

look like depicted on the left (goal), such that we have a relatively clear and uncluttered mapping. The ultimate goal is to have exactly one fitting FM1 (of type DDF) available for every Fault that occurs in the field. This means domain experts need to transform FM1s of type PCS and Other into DDFs. That might sound like a relatively easy task to complete, but has proven to be a difficult problem. The price of creating one DDF is relatively high as many work-hours have to be spend to guarantee high quality. Furthermore, it is often not simple to investigate whether similar FM1s already exist and whether they also apply to the new problem at hand.

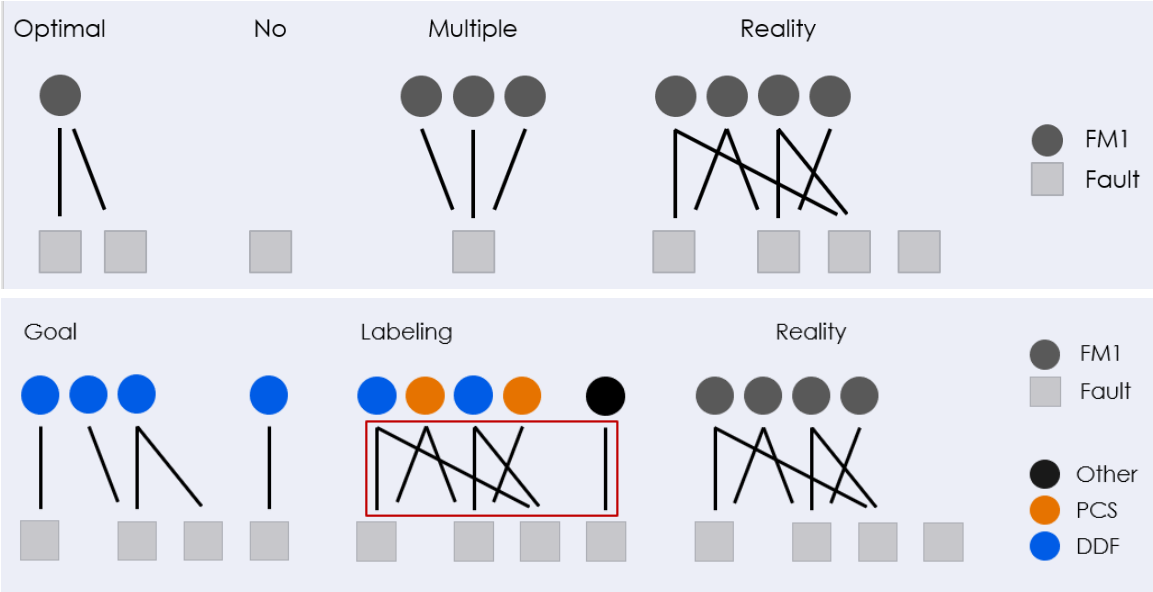

Figure 1: optimal diagnosis (top: mapping between FM1 and Fault; bottom: FM1 types and focus)

One example of field data where the mapping is not perfect is shown in Figure 2. As can be seen, for the selected Fault, two FM1s are available and therefore they are overlapping. The engineer working on this Fault has to make an educated guess on which FM1 is most fitting for this instance, which can result in unnecessary downtime. FMkb has a way to indicate the problem, but only with a manual and time-intensive process. The overlap cannot be retrieved automatically and is investigated for one Fault at a time. When thousands of Faults occur daily, a more sustainable approach to resolve data quality issues should be constructed.

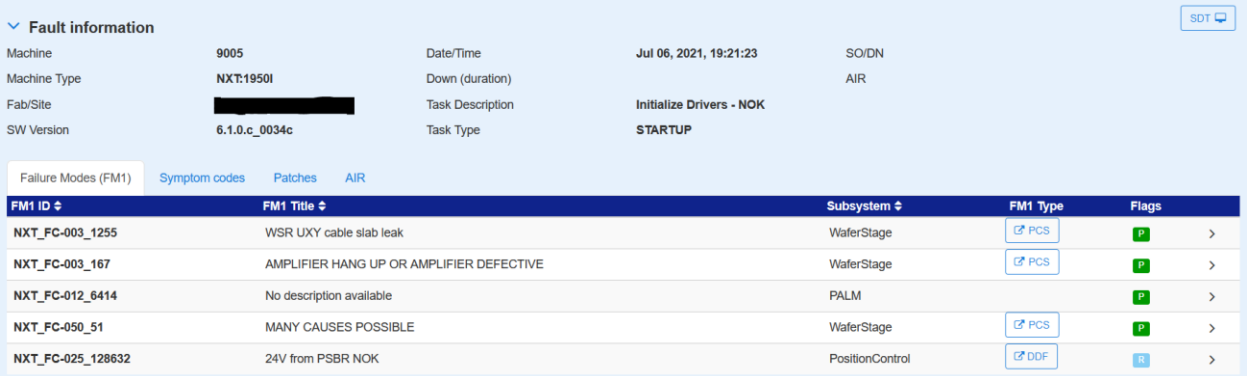

Figure 2: FMkb interface showing non-perfect diagnoses for one particular Fault

1.2 Thesis structure

In the remainder of this thesis, we describe how the problem is attacked. In Chapter 2 (Background), an overview of the current tools, available data and the concepts used at ASML is given. Several new concepts were invented throughout this project, they are defined for later reference. Next, a thorough analysis of the workflows, target groups and use cases was performed, which is described in Chapter 3. The analysis leads to a list of tasks and requirements to be taken into account while designing solutions. In Chapter 4 we consider the current state of the art in related work, such as visualization literature and comparable real-world applications. This provides a foundation we can build upon to design a solution. Chapter 5 on visualization design explains the process and choices that were made along the way to come to a visualization tool.

In Chapter 6 (Final Visualization) we go deeper into the end-result of the design process and document the created tool in its final form for this project. The technicalities of the prototype setup are further described in Chapter 7 on implementation, which also touches upon the used software and integration with live production data. To evaluate the tool, we invited participants for a critical domain experts review (Chapter 8), to verify whether the tool improves the established way of work and has potential to become a frequently exploited resource. The results of the user evaluation are presented and interpreted as the gathered qualitative and quantitative insights. In Chapter 9 a discussion of the advantages and limitations of the current visualization tool is presented. Suggestions are made for future work on the subject and improvements to be made in terms of getting more value out of the data. The report is wrapped up with a short summary in Chapter 10 (Conclusion), followed by a list of References and Abbreviations used in this document. The Appendices present screenshots and documents that were relevant throughout the project, but which had less relevance for the bigger picture.

Chapter 2

Background

After the description of the problem in the previous chapter, we consider what tools, concepts and data are already introduced by ASML to work towards a solution. First, we discuss what tools are currently available, what purpose they fulfill and what people and tasks they support. Next, an overview of the in-use general machine concepts and specific diagnostic concepts is given. Several new definitions are introduced during this project as well. Afterwards a simplified sketch of the database is provided with data relations and cardinalities attached.

2.1 Established tools

Two main tools that support diagnostic tasks are currently in use, namely the Smart Diagnostics Tool (SDT) and the Failure Mode knowledge base (FMkb). SDT can be seen as the facilitator of incoming machine event data and provider of direct views on relevant data points. The tool provides valuable insights at the moment a machine failure occurs, by providing potential solutions, underlying symptoms and related machine failures. Therefore, the tool is mostly used by Customer Support (CS) engineers who are situated in the field or by central engineers who come up with solutions to often recurring problems. The main focus of SDT is on diagnosing individual cases, to quickly get fitting solutions to the right engineers. Next to that, FMkb is more focused on the analytical colleagues who try to improve data quality and find diagnostic gaps. The main use case of FMkb is to investigate, tune and optimize the diagnostic process. The tool is usually part of the toolset of diagnostic Development & Engineering (D&E) engineers, who work to continually improve machine design and the processes around it. Both SDT and FMkb operate on the same dataset, where SDT is mostly responsible for data entry and FMkb for data analysis and process enhancement.

2.2 Concepts

Many concepts have been established by the diagnostic teams to help with communication and to get a good grasp of the data system. The most important data concepts have a hierarchical structure (indented from left to right), as is depicted in Figure 3. In the Definitions section they are further elaborated.

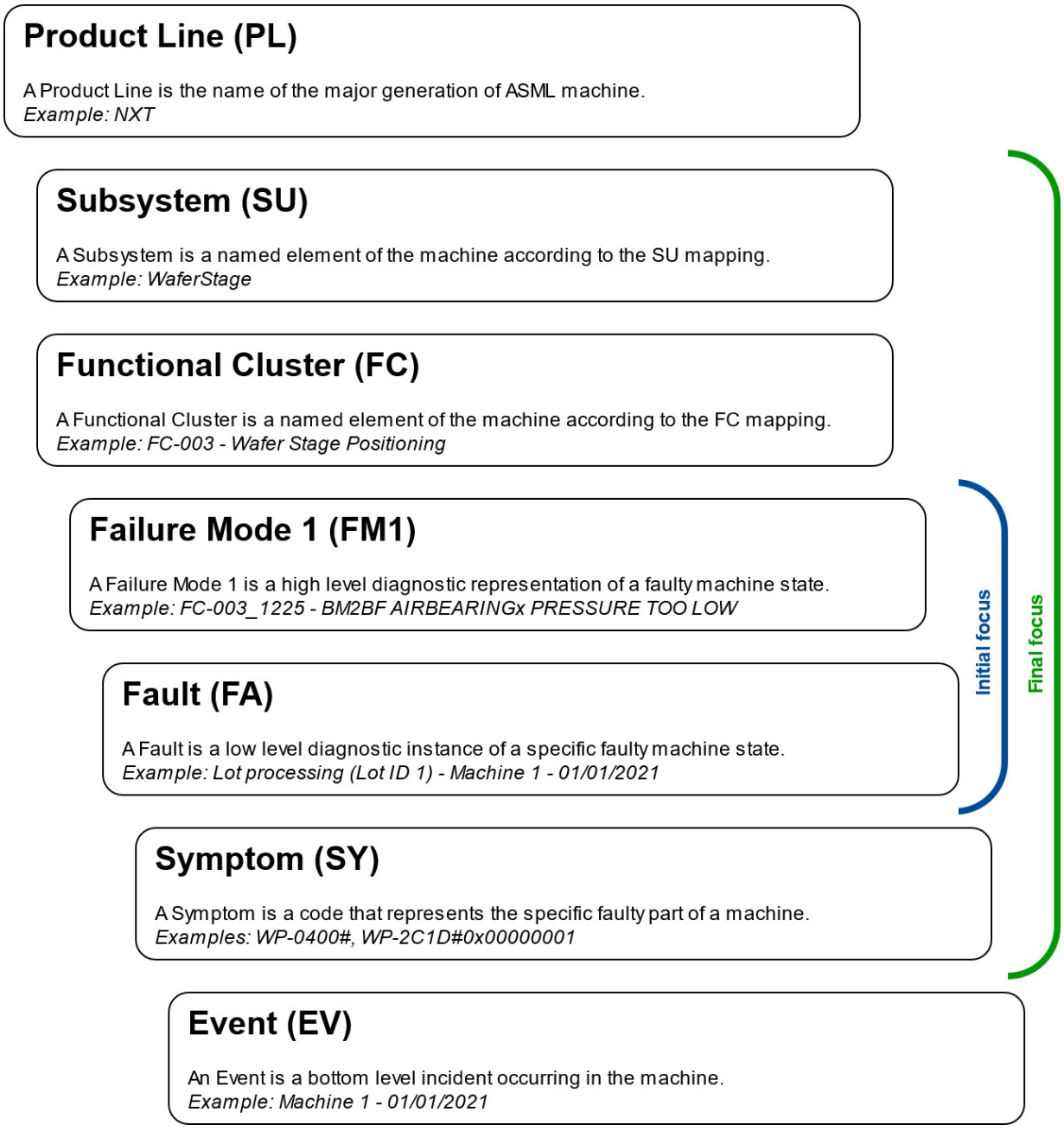

Figure 3: data concepts and definitions

2.2.1 Definitions

In this section we introduce the concepts that are currently used by the engineers at ASML and the new concepts generated by this work. Formalizing these gets us a good understanding of the established diagnostic terminology and supports the description of choices and novel solutions presented later on. For each definition a description is given, and where needed abbreviations, properties and examples are displayed also provided.

Data definitions

Starting off we have the data definitions, which are usually originating from data tables and describe the main diagnostic systems. The definitions are currently in use by SDT and FMkb, and are well-known to most other disciplines at ASML.

Definition 1 Product Line (PL)

A Product Line is the name of the major generation of ASML machine. The progress went from XT (older) to NXT (mainstream) to NXE (newest). Later on SRC (light source for NXE) was added as an individual PL, as it is almost a standalone element that has minimal connection to other parts of a machine.

Properties

Name	Name of the PL.
Set of SUs	All SUs that are part of this PL.

Example

NXT

Definition 2 Subsystem (SU)

A Subsystem is a named element of the machine according to the SU mapping. The mappings of FC and SU are different. The Subsystem usually consist of multiple machine elements and can span over several FCs.

Properties

Name	Name and identifier of the SU.
Set of FCs	All FCs that are part of this SU.
Set of FM1s	All FM1s that are categorized for this SU.

Example

WaferStage

Definition 3 Functional Cluster (FC)

A Functional Cluster is an organizational design discipline responsible for part of the machine. The mappings of FC and SU are different. The FC usually consists of specific machine elements, that can belong to multiple SUs. Each FC usually has a team of engineers and analysts working on the continuous improvement of the design (performance, reliability, cost) of a specific machine part. More and more, this team is becoming responsible for the ability to rapidly diagnose issues related to their machine part.

Properties

Name	Name and identifier of the FC.
Description	Short description of the FC.
Set of FM1s	All FM1s that are categorized for this FC.

Example

FC-003 - Wafer Stage Positioning

Definition 4 Failure Mode 1 (FM1)

A Failure Mode 1 is a diagnostic representation of a faulty machine state. FM1s provide a classification of Faults based on their cause and potentially provides a solution to its associated Faults. The FM1 can be of type DDF, PCS or Other, were DDF is most desirable. We need a 1 in the name here as FM2s are currently in development, they are essentially higher level abstractions of FM1s.

Properties

ID	Identifier of the FM1.
Name	Name of the FM1.
Type	Type of the FM1, can either be DDF, PCS or Other.
DDF	Deterministic Diagnostic Flow; this is an advanced diagnostic solution of the newest type. A DDF makes use of expert rules to classify Faults and has a high quality due to extensive setup and release procedures.
PCS	Problem Cause Solution; it means an old diagnostic solution is available. It is suboptimal since data quality can be low, but is still better than having no solution. The goal is to replace all PCSs by DDFs in the future.
Other	Also referred to as Unknown; it means there is no description or solution available and results in a gap in the diagnostic landscape. An unknown FM1 is created for a new Fault when no sufficient pattern match with the established FM1s is present.
Set of FAs	All faults that are matched with this FM1, via either Pattern matching (P flag), Rule matching (R flag) or Diagnosed by user (D flag).

Example

FC-003_1225 - BM2BF AIRBEARINGx PRESSURE TOO LOW

Definition 5 Fault (FA)

A Fault is a diagnostic instance of a specific faulty machine state. This is a particular incident that happened anywhere in the field and should be resolved as quickly as possible. Faults are pulled from all ASML machines daily and combined into one central dataset. Related to a Fault is a Down, which is a high impact diagnostic instance of a faulty machine, usually consisting of multiple Faults. Downs are not further considered for now, as they are more complex in relation to FM1 overlap. In the end, the invented methods for Faults can be used for Downs as well.

Properties

ID	Identifier of the FA.
Description	Short description of the FA.
Timestamp	Date and time at which the FA occurred.
Duration	Duration of the Fault, amount of time the machine was out of order.
Others	There are many other interesting properties associated with a Fault. Machine, Machine type, Fab/Site, Software version, Task.
Set of FM1s	All FM1s that are matched with this FA, via either pattern matching (P flag), rule matching (R flag) or Diagnosed by user (D flag).

Set of SYs — All SYs that are associated with this FA, taken from the machine state and event log. A boolean value is associated with each SY, indicating whether this Symptom has to be filtered out. In other words, whether it is noise for this particular Fault or useful data.

Example
Lot Processing (Lot ID 1) - Machine 1 - 01/01/2021

Definition 6 Symptom (SY)

A Symptom is a code that represents a specific issue with a specific part of a machine. The Symptoms are taken from the machine state and event logs. They give an indication on which parts of the machine were affected by the Fault and the underlying issues. A Fault typically has a set of 3-20 Symptoms associated with it.

Properties

Code — Identifying code of the SY. The code consist of a two letter machine part indication and then a four character identifier. When multiple SYs have the same two and four character part, they are very similar to each other. Therefore, the code is optionally followed by a byte representing the reasoning behind the SY.

Examples
WP-0400#, WP-2C1D#0x00000001

Definition 7 Event (EV)

An Event is a bottom level incident occurring in the machine. An Event can either be normal behavior or abnormal behavior. Events are taken from the extensive machine event log and then filtered by complex algorithms to obtain the interesting bits of information. A Fault always consists of at least of one starting and one ending Event.

Properties

ID — Identifier of the EV.

Timestamp — Date and time at which the EV occurred.

Others — There are many other interesting properties associated with an event, similar to those of Faults. As this data is usually aggregated to relevant insights, we will not look into the exact data properties that are available at this lowest event layer.

Example
Machine 1 - 01/01/2021

Performance metrics definitions

Furthermore, there is an established set of metrics to measure the performance of the diagnostics system. Most of these were introduced by diagnostics teams and management to get an overview of the current state of the data and processes in use.

Definition 8 Diagnostic hit rate

The diagnostic hit rate metric represents the chance of the diagnostic software returning a valid FM1 diagnosis for a given Fault. It is expressed as a percentage of FM1 coverage (DDF or PCS) for all Faults, since the outcome depends on the relation between the amount of covered Faults

and the total amount of Faults. I.e., hit rate = covered Faults / total Faults. The hit rate is estimated by running the diagnostic algorithms on a sample of recent Faults from the field. Using a random sample of Faults ensures that the hit rate is based on a realistic frequency distribution of FM1 occurrences. In practice, the sample of Faults is often filtered to include Faults with a significant impact (hours of downtime) and of the type DDF.

Definition 9 Diagnostic quality

The diagnostic quality metric represents the chance that a diagnostic recommendation is considered acceptable by the end-user. In other words, it is the overall correctness and usefulness of the available FM1s. This metric is not straightforward to measure, as the 100% truth is impossible to know at any point in time. Therefore, the quality of diagnostics is estimated by asking feedback from actual users of the SDT diagnostic recommender. It is measured as the percentage of cases where field engineers agree with the diagnosis provided by SDT. The data is collected by a mini-questionnaire within the diagnostic software where engineers can essentially Like or Dislike the provided diagnosis. The metric can be investigated per FM1, thus allowing faulty diagnoses to be identified, analyzed and improved.

ASML strives to get a fair balance between the diagnostic hit rate and the diagnostic quality. Having high coverage does not mean much when the quality of FM1s is low. The other way around, high quality FM1s are of not useful when the majority of the Faults cannot be diagnosed to begin with.

Definition 10 Mean Time To Repair and Mean Time To Diagnose

Mean Time To Repair (MTTR) is a metric used to investigate the speed at which detected Faults and Downs can be resolved. The Mean Time To Diagnose (MTTD) is closely related to the MTTR as it is part of its time calculation. With the MTTD we are able to look more closely at the time it takes to diagnose a Fault. The two metrics are essentially indicating the speed of which an FM1 can be found that correctly solves the Fault at hand. A rough estimate is that the MTTD is usually about 1/3 of the MTTR.

New conceptual definitions

Finally, during the project we have introduced new definitions. The FM1 overlap, FM1 cluster, FM1 neighborhood and FM1 symptom fingerprint notions are defined to describe observed problems in the field. Defining these concepts is thought to be relevant for further development of and improvements to the diagnostics system.

Definition 11 FM1 overlap and FM1 connection

Two FM1s have overlap when there are Faults categorized for both FM1s. In other words, the intersection between the sets of Faults from both FM1s is non-empty. The FM1s are said to have a connection when the amount of overlap is higher than the FM1 overlap threshold. This threshold varies for each use case and should be adjustable by the experts. Overlap is defined as the intersection between A and B, where A and B are sets of Faults belonging to two FM1s.

Absolute FM1 overlap $AFO = A \cap B$

An important part of the FM1 overlap situation is that it is not symmetric, which results in the need for the definition of relative FM1 overlap. Between the relative overlap from A to B or from B to A, there can be different outcomes. For example, A can be a subset of B ($A \cap B = A$) with a relative overlap of 100%, whereas B can only have a 30% relative overlap with A in the same example. Both cases are valuable information for diagnostic experts.

Relative FM1 overlap $RFO = |A \cap B| / |A|$ or $RFO = |A \cap B| / |B|$

To overcome the difference between relative overlap depending on the viewpoint, symmetric overlap is introduced. The symmetric FM1 overlap is calculated by taking the amount of overlap between A and B and divided it by the average size of A and B.

Symmetric FM1 overlap $SFO = |A \cap B| / ((|A| + |B|) / 2)$

The FM1 connection is defined as a Boolean value that indicates whether the symmetric FM1 overlap is higher than the set overlap threshold. Here, α denotes the overlap threshold as a fraction, thus a value between 0 and 1.

FM1 connection $CON = |A \cap B| / ((|A| + |B|) / 2) \geq \alpha$

Definition 12 FM1 cluster

An FM1 cluster is a set of FM1s that are related to each other, either directly or indirectly via FM1 connections. It usually results into a dense graph with overlap between most of the FM1s in the cluster. The FM1 cluster is bound to the applied filters, such as the selected FCs or SUs. Interesting properties from the individual FM1s can be combined and aggregated to show the importance and structure of a cluster.

Properties

ID	Identifier of the FM1 cluster. The identifier is set to be the FM1 id of the biggest FM1 in the cluster.
FM1 ids	The list of FM1 ids that are part of the cluster.
No. of FM1s	The number of FM1s that are part of the cluster.
No. of FAs	The number of distinct Faults that are associated with the FM1s in this cluster. Note that we cannot simply sum the set of Faults from each FM1, as there is overlap between the Faults that needs to be taken into consideration.

Definition 13 FM1 neighborhood

The neighborhood of a set of FM1s are all Failure Modes that have a direct FM1 connection to it. The neighborhood is determined prior to any filtering, thus represents the underlying data directly. This makes the FM1 neighborhood is a good indicator of FM1 quality, since it clearly shows which FM1s are overlapping without imposed bias through selection criteria.

To better explain this concept, an analogy with airplane flights can be made. Domestic flights between cities stand for the FM1s in the selected set, where connections only occur between elements of the initial set. Foreign flights can be seen as flights with a single hop from cities in the selected set. In total, we now

get the selected set plus all its single hop neighbors (domestic flights and foreign flights together). This results in a total set of FM1s, the FM1 neighborhood.

Definition 14 FM1 symptom fingerprint

The FM1 symptom fingerprint is a computed representation of the Symptoms that are frequently occurring for an FM1. The number of Symptoms are aggregated, which gives the user a frequency based indication of how often the Symptom was encountered in its associated Faults. Such a symptom fingerprint gives the domain expert an indication of what the underlying broken parts of the machine could be and what the root cause of a particular Fault is.

2.2.2 Analogy

To be able to follow along with the presented designs and argumentations in future sections, it is important to have a good understanding of the above concepts. Serving as an example, an everyday analogy is made between ASML machines and bicycles.

The city bicycle can be the Product Line, whereas a mountain bike is another PL. An example of a Functional Cluster is the wheel. An often occurring FM1 that can be set up is the broken tire, which is of type DDF and has an associated solution to replace the outer tire. When an incident (Fault) of the broken tire occurs on a certain day, the Symptoms are important to look into. The Symptoms could be 'hole in tire' or 'deflated tire', they indicate that this is indeed a problem classified as this FM1.

2.3 Data

Another part of the background material is the available data and the way it is structured. Event logs from all fabs around the world are combined into one central database, where mining techniques are applied for data enhancement. The Customer Support and analytical diagnostics teams are working hard to add FM1s (most DDFs) to the database as well. This results in an ever-growing set of data that should be stored correctly and analyzed effectively. The abundance of data becomes more clear when we look at the actual numbers for each data concept; 4 PLs, 40 SUs, 90 FCs, 70.000 Symptoms, 1.200.000 FM1s, 5.800.000 Faults and 450.000 Downs. Roughly speaking, for each data concept one data table exists in the database. Furthermore, bits of information are captured in tables that represent relationships between the concepts and provide additional data properties.

An overly simplified version of the database we are working with is shown in Figure 4. The actual database is rather complex and consists of roughly 50 tables spread over five different layers. Here we only show the six most interesting tables for this project, to keep the focus clear and make it easier to understand. From the image a couple of interesting data relations can be derived, such as the one between FCs and FM1s and between FM1s and Faults. As mentioned previously, FM1s can have overlap with each other based on their corresponding Faults. When we mine the overlap data, a graph structure becomes evident. In this graph, the FM1s are the nodes and the FM1 connections are the links. Another interesting property to be mined is the Symptom fingerprint of an FM1. If we aggregate the Faults that belong to an FM1 and the Symptoms belonging to said Faults, a histogram showing the distribution of Symptoms over an FM1 can be generated. This FM1 symptom fingerprint can be used to investigate overlap between FM1s and to define expert rules for new DDFs.

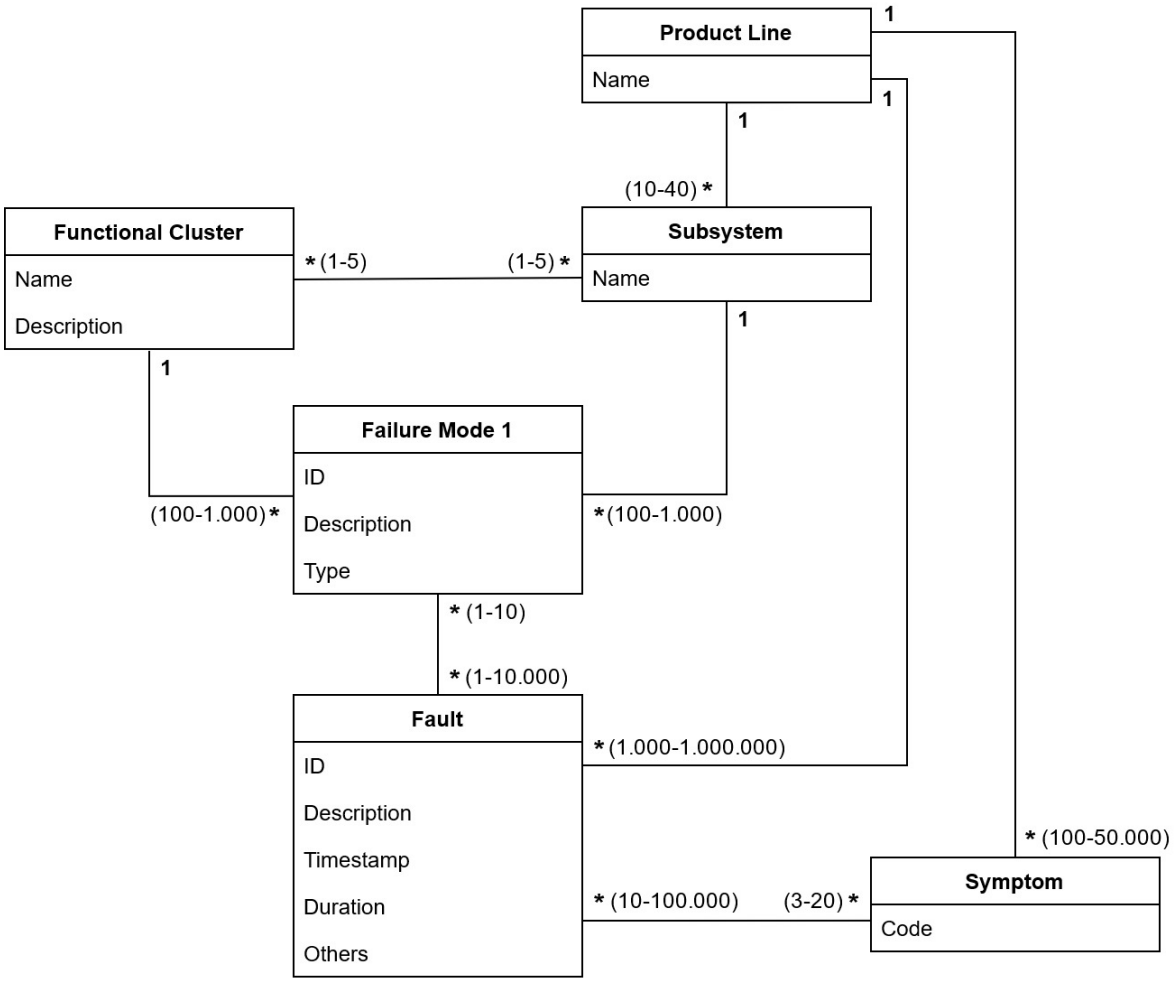

Figure 4: simplified database diagram

Chapter 3

Analysis

In this chapter, we further shape the solution landscape by looking at the diagnostic process at ASML. Analyzing the process provides a better focus, scope and goals, and also the target group and their activities become evident. Based on the activities, tasks can be defined that users perform on a regular basis. These tasks should be taken into account when designing the solution, following the requirements set up around the full analysis.

3.1 Diagnostic process

In total, four experts were interviewed to get a good overview of what the entire diagnostic process looks like. Multiple teams from different departments are working together to keep the machine uptime to a maximum. Next to that, a variety of tools is used to support the diagnostic work. The main tools are SDT and FMkb, which were already discussed in the previous section. There is a list of other tools that can be used for more specific use cases, those are not further looked into here. An overview on how the machines, teams and tools come together in the diagnostic process is depicted in Figure 5.

The fabs are the starting point of this diagram, as they provide data in the form of event logs to the database. SDT is used as a practical tool that does issue tracking and provides direct solutions to machine problems. The Field engineers utilize this tool to quickly look for more information on the failure and its potential solution. Field engineers are supported by a team of Central engineers, who have more specialized knowledge about certain parts of the machine and can come up with new solutions as well. Furthermore, FMkb was built alongside SDT. They have synchronized databases, as FMkb loads and synchronizes its FM1s, Faults, Downs and Symptoms. FMkb was developed for analytical purposes and supports a wider variety of teams, namely the Central engineers, R&D engineers, Factory engineers and Analysts. FMkb's main focus is controlling data quality and finding diagnostic gaps. All this information is used by the teams in different ways, mostly to guarantee good uptime, structurally prevent issues in the future and steer engineers in the right direction for new diagnoses and solutions. In this work, FMvet is introduced as a tool that utilizes the FMkb database and aims to support the Analysts, Central engineers and R&D engineers. The visual focus provides a new view on the data, that can generate valuable insights related to data quality (FM1 overlap and FM1 neighborhood) and could speed up the data analysis process.

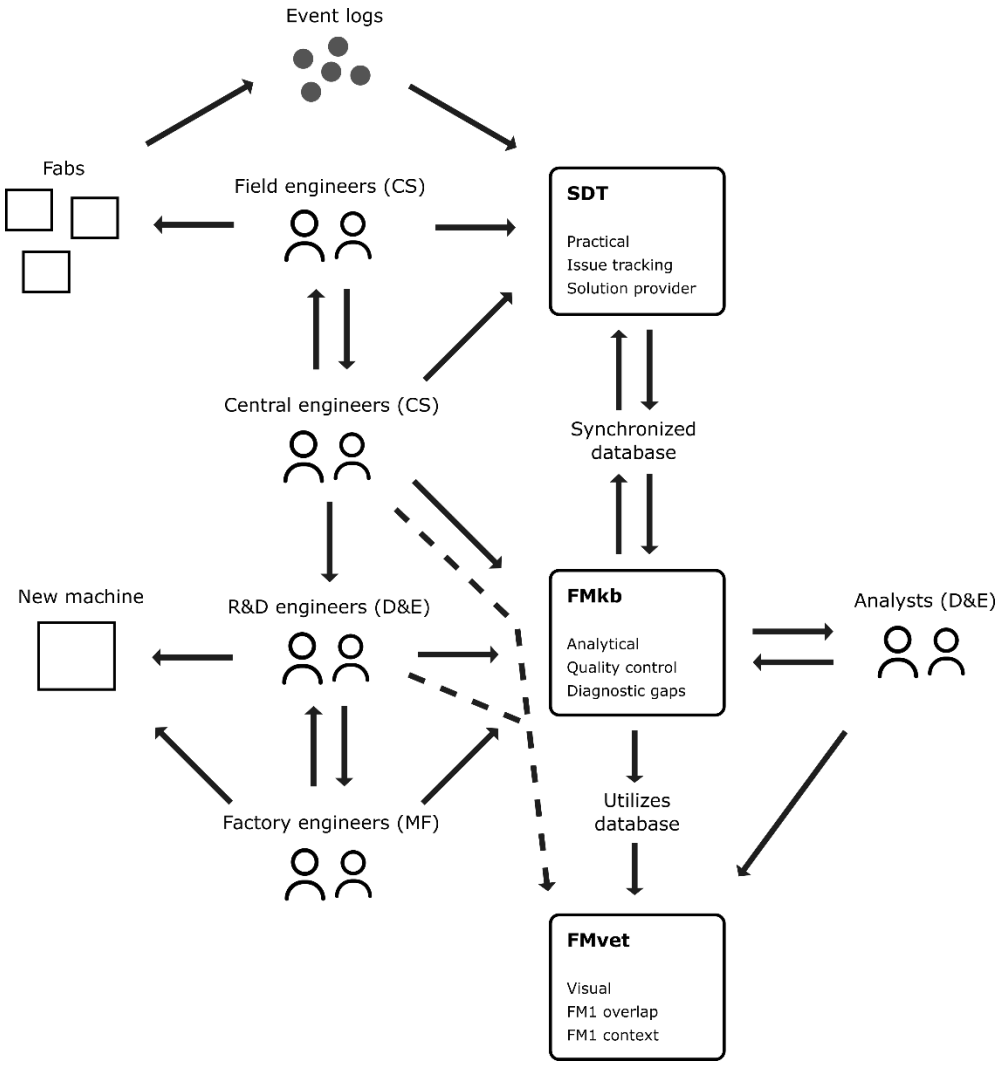

Figure 5: schematic overview of the diagnostic process at ASML

3.1.1 Goals and focus

For the correct positioning of FMvet in the diagnostic process, clear goals and focus are important. As shown in Figure 5, the tool is mostly used by Analysts. By adding the visual tool to their toolset we hope to support them to reach three main goals, which were the purpose of designing it to begin with: (1) the Mean Time To Diagnose should be reduced; (2) the diagnostic quality of FM1s should be increased; and (3) the amount of correctly diagnosable FM1s (hit rate) should be increased as well. The influence FMvet has on these metrics is hard to measure in the short term and in an isolated manner. Therefore, initially a Domain Expert Review is used, followed by analyzing the tool's usage data over prolonged periods of time (outside of scope). Interesting usage data is whether the amount of users is growing over time and whether the tool is consulted over long periods by the same user.

To be able to reach these goals in a structured process, a set of focus points was defined. Initially the focus was kept minimal by only investigating the relations between FM1s and Faults (see Figure 3 in the Background chapter). Later on the focus was broadened to include the higher level FC and SU mappings, and the lower level Symptom mappings. These concepts are all used extensively on a daily basis and could therefore benefit from visual representation. Other concepts such as PLs are set in stone, and concepts such as Events are too unstructured and detailed, hence they are not included. Furthermore, it was decided to keep the functional implementation within FMkb outside the scope of this project. This gives freedom to develop the tool in a closed environment without having to worry about established methods, technology or breaking production tools.

3.2 Target groups

The three main departments involved in the diagnostic process are Customer Support (CS), Development & Engineering (D&E), and Machine Factory (MF). CS mostly consists of Field engineers and Central engineers that work directly with machines, whereas D&E is more concerned with the continuous improvement of deployed machines and the development of new generations. MF is a hybrid department focusing on assembling the machines. The three departments are integral to having an efficient diagnostic system and are therefore potential target groups of FMvet. The general roles of each target group are abstracted here, but in reality there can be overlap and edge cases. One Persona (P1 to P5) is defined for each type of team within a target group, painting a picture of what their daily work characteristics are like. By no means an exhaustive overview of the teams is given here, as our interest only lies in the ones most related to the diagnostic field.

Customer Support (CS)

As the name suggest, Customer Support people are usually working directly with or at the customer fabs. They have a mindset and way of working that follows the bottom-up approach. Whenever a Fault occurs in the field, it is looked at with urgency to fix the problem. From this specific data point, a look into similar Faults and corresponding FM1s is performed. The main helpful metric to improve for this target group is to decrease the MTTR and MTTD, which means problems can be identified, diagnosed and resolved in a timely manner.

P1 Field engineer

- Working on the machine and fab location to quickly resolve Faults and Downs by following the prescribed FM1 diagnosis. Usually part of a team of specifically educated engineers about a specific type of machine.

P2 Central engineer

- Generating new solutions for Faults in the form of FM1s. Decrease complexity and cardinality of Faults and FM1s. Generally have a good understanding of all parts and types of machines.
- Finding gaps in the diagnostic landscape; investigate the effectiveness of diagnostics and how many times certain FM1s occur.

Manufacturing (MF)

The Manufacturing department performs the assembly of new machines and is located in the factories in Veldhoven. Factory engineers focus on testing machines and quick machine fixes, as well as conceptualizing design improvements. Optimizing cycle time, the time it takes to a build machine, is their

main concern. Separate subdepartments for XT/NXT machines and for NXE/SRC machines exist, both employing specially trained engineers for the specific technology.

P3 Factory engineer

- Assembling new machines and troubleshooting where needed.
- Design structural machine advancements or improve assembling process efficiency.

Development & Engineering (D&E)

The Development & Engineering group is usually based at ASML premises and does not directly work with active machines in the field. Their focus is to have a more analytical, preventative and developmental role, typically using a top-down approach. The main metrics here are to increase the diagnostic hit rate and quality (Analysts), as well as to work on structural reliability improvements (R&D engineers).

P4 R&D quality/reliability/diagnosability/availability engineers (referred to as analysts from here on)

- Improve Faults and FM1 detection methods, pattern matching algorithms and noise filtering algorithms.
- Support CS by investigating diagnostic gaps, setting priorities on which unknown FM1s should be handled first and whether systematic process improvements can be made.

P5 R&D design engineers

- Prevention of inefficiencies by utilizing diagnostics data; the system design can be altered based on findings related to what machine parts often result in errors and how they can be prevented in the future.

3.3 Activities, tasks and functionalities

The aforementioned target groups have main activities that can be seen as high level tasks that need to be completed. The aim of FMvet is to support users in activities and tasks by presenting the data in useful and appealing ways. Related to the activities, there is a list of low level tasks. These tasks focus on certain aspects of the daily work that are performed on a frequent basis. A translation is made from the tasks to functionalities. The functionalities function as abstractions of the data concepts and blueprint for the visualizations. Formalizing the activities, tasks and functionalities is used as a tool to gain better insight into what aspects of the work can be improved with the visual exploration tool.

3.3.1 Activities

The three most prevalent user activities supported by the tool are listed below. The concrete tasks associated with each activity are displayed under the tasks subsection.

A1 Assistance with diagnostic data creation (DDF, expert rules).

A2 Quality control during data release (DDF, expert rules).

A3 Quality control through data cleanup (DDF, PCS, Other).

3.3.2 Tasks

The most relevant diagnostic tasks are listed below. The tasks are corresponding to certain personas and activities, the relations between them are displayed in Figure 6.

T1 Find and set priorities on diagnostic gaps.

T2 Inspect undiagnosable Faults and get hints on what FM1s to create.

T3 Determine which new expert rules can be created to correctly identify Faults belonging to the investigated FM1.

T4 Investigate similarities between a newly created FM1 and established FM1s.

T5 Determine whether all intended Faults are covered by the FM1 of type DDF that was introduced as a replacement of an FM1 of type PCS or Other.

T6 Investigate whether the quality of an FM1 of type DDF can be improved by altering its expert rules.

T7 Investigate which FM1s are overlapping with the selected FM1.

T8 Investigate whether FM1s can be merged, based on FM1 similarity and clustering.

T9 Investigate underlying data similarities in relation to what factors cause the generation of FM1 clusters.

T10 Find FM1s to deprecate if they are not used or not correct anymore.

T11 Identify opportunities for design reliability improvements.

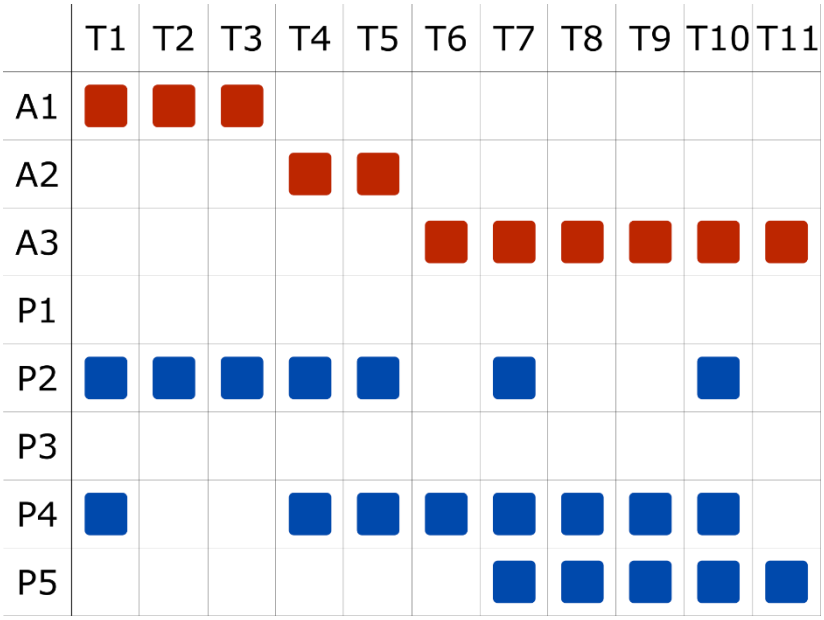

Figure 6: traceability matrix between the tasks, activities and personas

3.3.3 Functionalities

A translation step is needed from the introduced activities and tasks, to functionalities that should be included in the visualization tool. An abstraction is made from tasks that the user would like to perform, to the data concepts and properties that the tool should visualize. Underlying data characteristics such as overlap between FM1s and Symptom fingerprints are starting points for the potential features as well. The list of functionalities is displayed below, and from which tasks they were derived is shown in Figure 7.

F1 Visualize diagnostic gaps (FM1s of type Other or, to lesser extent, type PCS).

F2 Visualize diagnostic coverage (FM1s of type DDF).

F3 Show the importance and relevance of FM1s in a particular set.

F4 Show the Symptoms, and their frequencies, belonging to FM1s and FM1 clusters.

F5 Visualize overlap between FM1s and overlap within FM1 clusters.

F6 Show the neighborhood of an FM1.

F7 Compute and visualize the FM1 clusters.

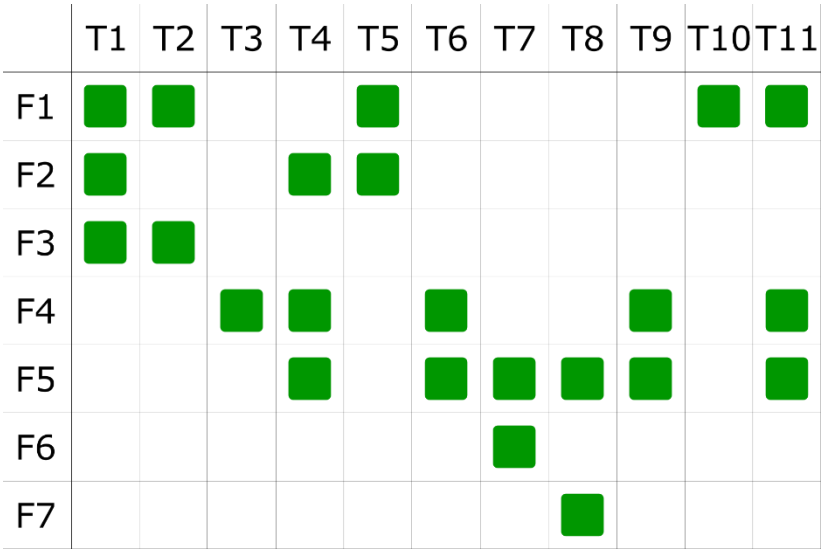

Figure 7: traceability matrix between the tasks and functionalities

3.4 Requirements

Following the conducted analysis, a set of requirements was defined for FMvet to live up to. These help to keep a sharp focus in the design process and to evaluate the conceptualized solution against.

R1 The tool should facilitate real and up-to-date data (FMkb production database).

R2 The tool should be intuitive; easy to learn and easy to use. The basic features should be operable by all users after two hours of usage.

R3 The tool should be fast; data loading can take five seconds at most and from there on the visual representations and animations should be fluent.

R4 The tool should allow the user to explore data and find underlying explanations, it should present multiple views on the data.

R5 The tool should be in line with the structure (data, API) and design (interface, UI) of FMkb. Filtering should work in a similar way as FMkb handles it.

R6 The tool should be deterministic and have a savable state.

R7 The tool should have up-to-date documentation.

R8 The tool should be interactive.

R9 The tool should enable the user to focus on specific data.

R10 The tool should provide correct results, especially in the visual representation.

R11 The tool should facilitate variability of the results. It shows the relevant low level data and links to more details in FMkb.

Chapter 4

Related Work

In this chapter, we look into related academic work. Familiarizing with the previously committed efforts in the related academic fields helps pave the way to designing a valuable application. The two major topics that are relevant to this project are Set visualization and Graph visualization. The FM1s and their connections have a graph structure, whereas the visualization of sets of Faults can come in handy as well.

4.1 Set visualization

The set is a mathematical definition of a collection of distinct elements. Sets are usually denoted with capital letters, whereas its elements are depicted with the roster notation (inside curly braces). For the academic work on Set visualization two overview papers from Alsallakh et al. (2014 and 2016) were consulted. Three examples of sets and their set relations being visualized are shown in Figure 8.

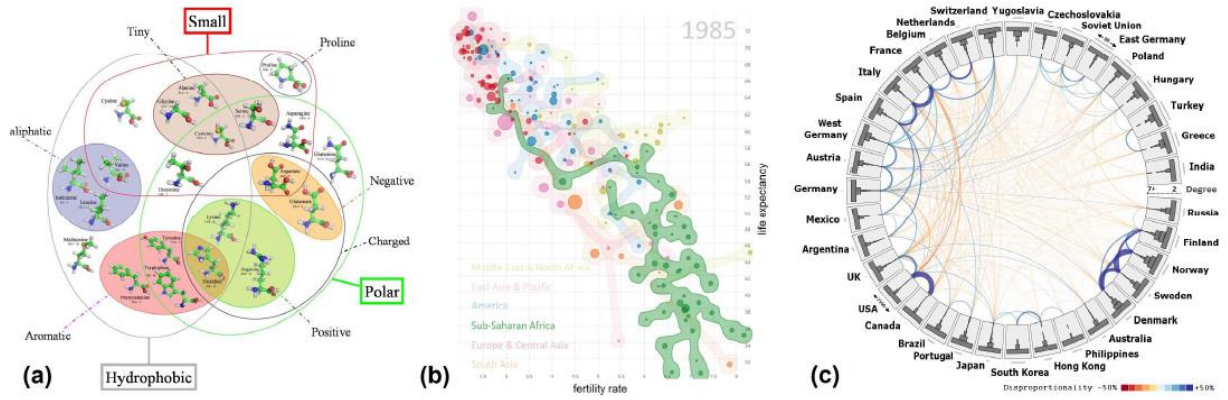

Figure 8: three visualizations of different sets (Alsallakh et al., 2014)
a) Euler diagram (Podtelezhnikov, 2008), b) Bubble sets (Collins et al., 2009),
c) Radial sets (Alsallakh et al., 2013)

Visualizing sets is a non-trivial problem due to the large number of potential relations between them. There are six main categories of Set Vis, these are: Euler and Venn diagrams, overlays, node-link diagrams, matrix-based techniques, aggregation-based techniques, and scatter plots. Many types of set visualizations are easy to understand and can be turned to fit the domain specific problem. The visualization techniques are especially powerful when used with sets of small size and with few relations between them. Therefore, limitations include the scalability, the limited possibilities of showing related data properties of the sets, and the unambiguous role of element ordering in visual set representations due to set elements inherently not having a particular order. For example, Euler and Venn diagrams only work well up to about ten sets, and bipartite graphs only concern two sets.

4.2 Graph visualization

The graph is a mathematical definition of structured data, consisting of entities with connections. A graph is a pair G = (V, E). V is a set of elements called vertices, which are the entities of the graph. E is a set of elements called edges, which are the connections or relations between the vertices. Graphs come in many different types, shapes, and have other varying properties. For example, they can be (un)directed, (un)weighted, and dense or sparse. Graphs can be stored as an adjacency list or as an adjacency matrix in computer systems, both having their own time complexity or space efficiency benefits. For more information on graphs, see Bollobás' book on Modern Graph Theory (2013).

The overview paper from Herman et al. (2000) was used as a starting point on the different approaches to Graph visualization. Graph drawing and layout algorithms have been optimized over time and come to a wide variety of potential visualization methods. The main recurring issue in Graph Vis is the large size of graphs, other concerns mostly depend on the type of graph data at hand. Furthermore, a debate has been going on about 2D versus 3D layouts. 3D usually allows for more space when needed, but makes it hard to find the right view as edges overlap and it hinders data exploration. Therefore, 2D representations are preferred for most use cases.

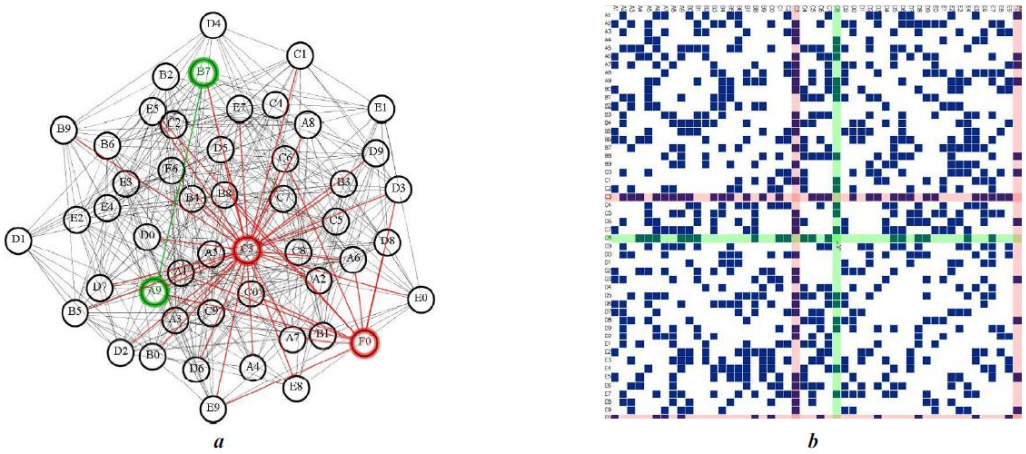

Figure 9: two visualizations of the same undirected graph (Ghoniem et al., 2004)
a) node-link diagram, b) matrix representation

The two most popular methods in Graph Vis are the node-link diagram and the matrix representation. New algorithms for both were constructed, respectively for better node positioning and for reordering the matrix. The two types of visualization are shown alongside each other in Figure 9. A comparison of the readability between graphs using node-link and matrix-based representations was performed (Ghoniem et al., 2004). In a task-based evaluation it was found that readability of node-link diagrams deteriorates when the graph size and link density are large, but they are superior for small and medium graphs. Users usually have experience with the node-link diagram, and not with the matrix representation. Next to that, node-link diagram are more intuitive to use and tasks like path tracing are much easier to perform. Matrix representation was found to be helpful in specific types of tasks such as 'find' queries or dealing with very dense graphs, but is lacking in other regards.

Node-link diagram

Node link diagrams are used to represent graphs in a visual manner. The vertices of a graph are represented as nodes or circles, whereas the edges are represented as links or lines between the nodes. Visual properties that may vary are the properties of a node (size, location, color, opacity, shape) and the properties of a link (width, length, color, opacity, style). This visualization type therefore allows for high adaptability and can be tailored towards the specific use case. The benefits of a node link diagram are easy data exploration, visual clustering, and its intuitiveness.

The elements in node-link diagrams are usually positioned by computing their optimal location according to a force-directed layout algorithm. There are specific best practices involved with these algorithms. In terms of improving readability, reduction of the number of crossing between links is most effective. Next to that, predictability should be high; two different runs of the algorithm, involving the same or similar graphs should not lead to radically different visual representations.

One application of node-link diagrams is the visualization of social networks, as is done in the work by Heer & Boyd (2005). They visualize online social networks, with the aim of exploring connectivity in large graph structures, supporting visual search and analysis, and automatically identifying and visualizing community structures. The developed techniques for easy data exploration and analyzing clusters also worked well for dense graphs. Navigation was done through panning and zooming, whilst highlighting specific parts could benefit from the ‘x-ray’ mode.

Matrix representation

The other type of graph representation is a 2D matrix. The header of a row or column is corresponding to a vertex, whereas a cell of the matrix indicates whether an edge exists between the two vertices. The row and column headers are usually identical and follow the same order, thus the diagonal represents the self-link. Visual properties that can be changed are the order of the vertices and the properties of cells such as background color, opacity, and content (numeric values, bar chart, etc.). The benefits of a matrix representation are the fast vertex and edge lookup, good indication of sparseness or density, and the grouping of vertices which have similar properties.

The order of a matrix is arbitrary, since the vertices do not inherently have an associated importance. Reordering the matrix can be used to identify clusters, highly-connected vertices, and other data properties of graphs. Developments in this field have been focused on automatic reordering and improvements to the efficiency of algorithms. Algorithms are ranked both on space and time complexity, as well as the quality of their results. Different graph patterns can be highlighted with different approaches, such as the block patterns, line or star pattern, bands pattern and noise anti-patterns. Each pattern has its own relevance and can show certain structure in the data, resulting in various insights to be gathered. Matrix representation works well for relatively small graphs and can show structure which is hard to grasp with other visualization methods, especially in dense graphs (Behrisch et al., 2016).

Chapter 5

Visualization Design

Chapter 5 describes and presents all of the visualization designs that were applied to the domain-specific problem. An extensive design process with an iterative approach was followed. In each iteration, the problem and solution at that point in time were reevaluated and improved upon were possible. The choices made in each iteration are documented and argued upon. Throughout the design process the final solution was formed by input from data experts, architects, visualization experts, and users. A diagram of the iterations and their main focus points is given in Figure 10 (two pages). Afterwards we discuss each iteration in its own subsection.

The benefits of having a small group of domain experts on short notice became apparent in the early phase of the project. It helped to get a clear focus and idea of what kind of tool was needed, to get a quick feedback loop on potential solutions, and to get interest from other ASML colleagues. This steamrolled into a focus group that could be contacted on a regular basis, providing feedback on major functional concerns and visual representation aspects.

Next to having external input, we applied a set of general design principles in the work. These principles are either personal views on visualization design or considered best practice by the visualization community. The three most prevalent design principles in this work are; consistency between views, Shneidermann's mantra, and high usability and learnability. Consistency between views means that all parts of the tool that represent the same data should be fairly identical to each other. It helps with a faster and more easy understanding of the visual language and prevents confusion. Shneidermann's mantra, also known as the Visual Information-Seeking mantra, says "overview first, zoom and filter, then details on demand" (Shneidermann, 2003; Craft & Cairns, 2005). This means first an overview should be presented to the user, that they can apply zooming and filters on. Whenever interesting cases are found, details can be requested if needed. Last, a general principal is that the tool should be ease to use and easy to learn. It is important the tool has a low entry barrier and experts should quickly get up and running. The tool will be used by a select group of users who are working with it for prolonged periods of time. Therefore, mastering the tool might take more time, and plenty of opportunity should be given to explore advanced features.

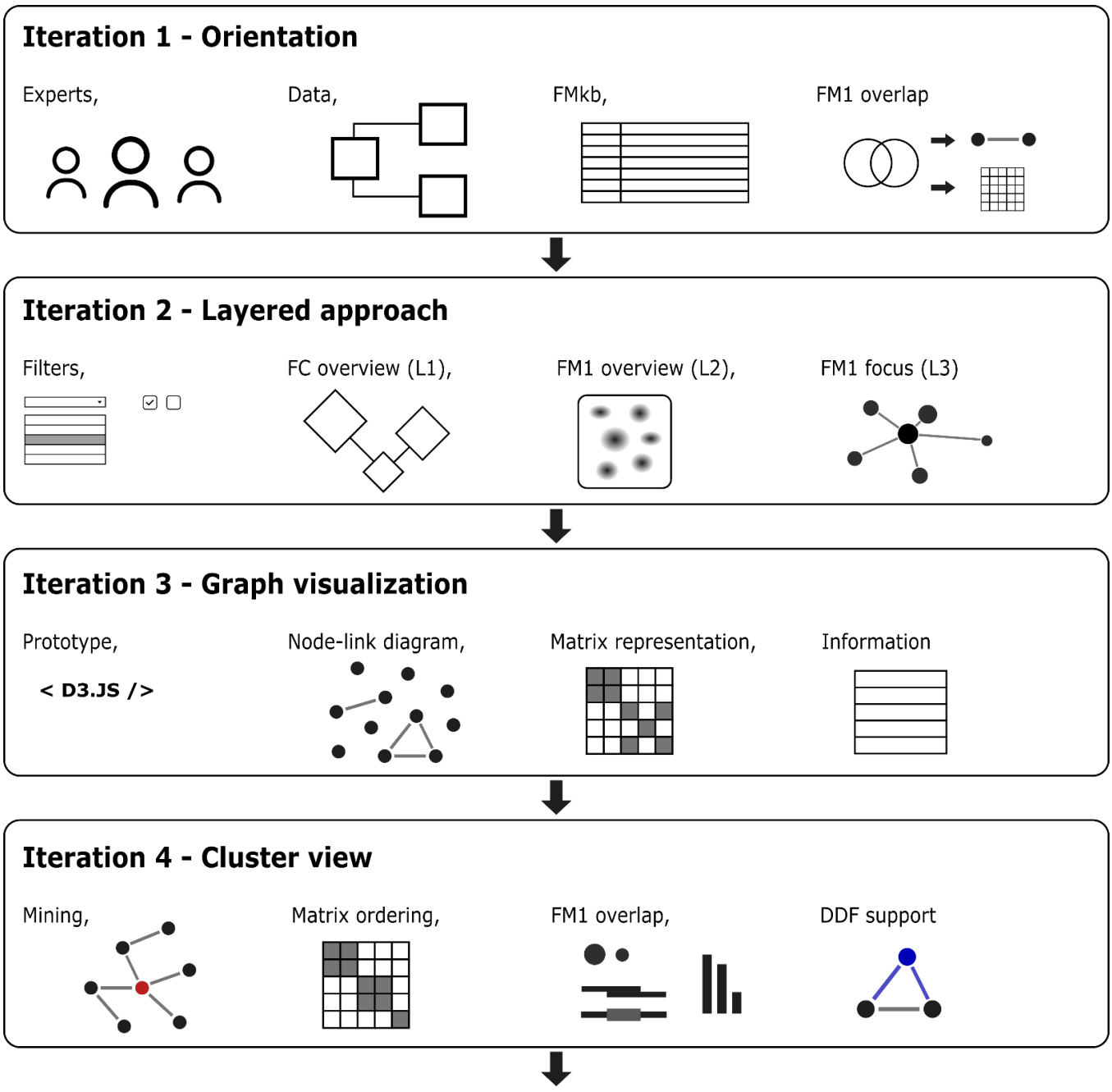

Figure continues on the next page

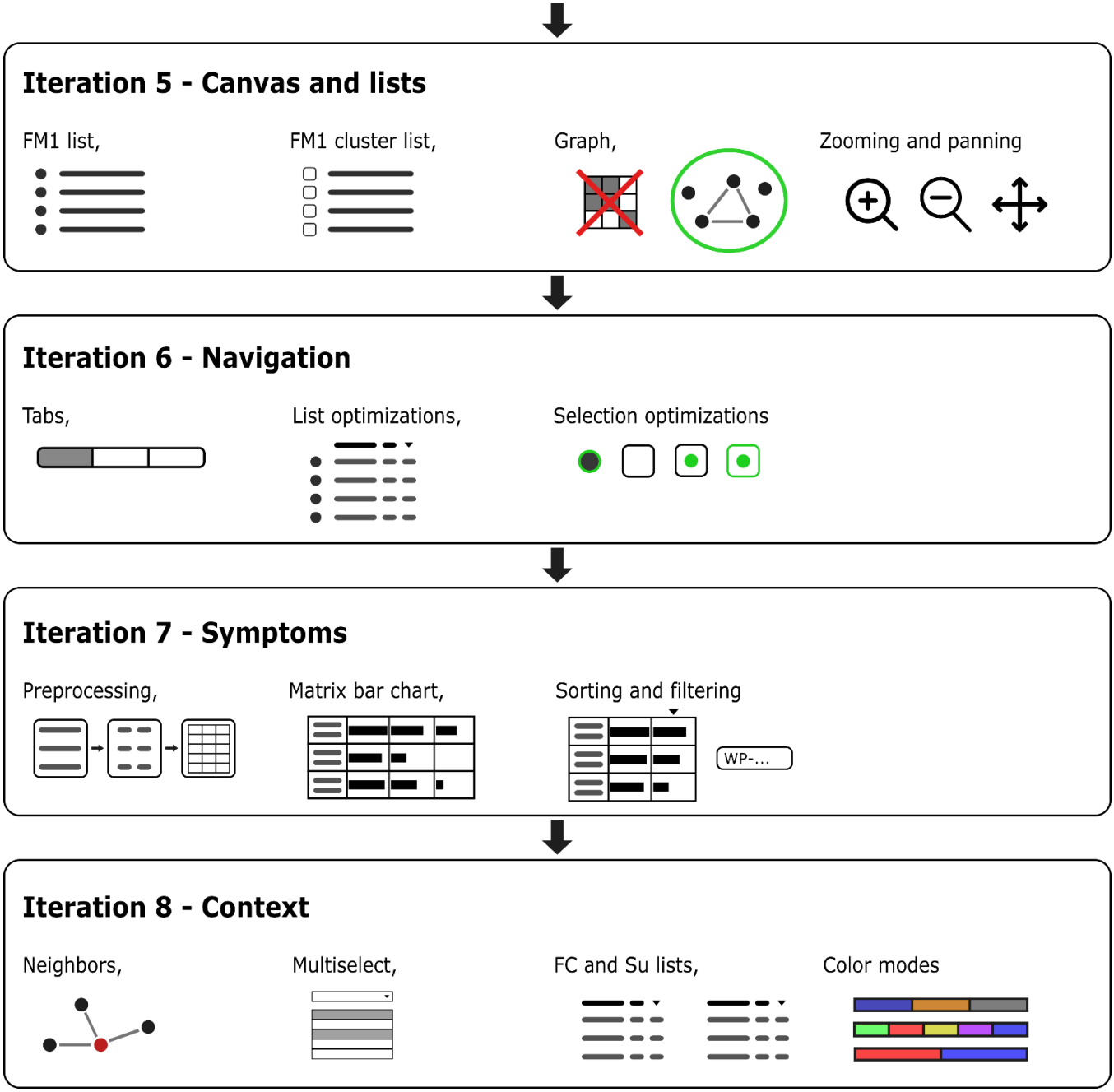

Figure 10: overview of the main iterations in the visualization design process
Some icons were made by Freepik from www.flaticon.com

5.1 Iteration 1 - Orientation

The first iteration was used as the orientation phase, in which the problem was defined and mockups were created to explore what data could benefit from visualization. Reading materials were collected and discussions with experts were organized to get an initial idea of where room for improvement lies. From the gathered information, it became evident an interactive visualization tool was needed, which served as an extension to the readily available FMkb interface. An important step was to get an indication of the target user's familiarity with FMkb data and with visualization in general.

Next, a deeper dive into the data was performed. Playing around with the FMkb interface gave a good overview of the underlying data structure and available data properties. Data access was handled via direct database queries or via the FMkb API, both have their own advantages and were used for data exploration. SQL queries were constructed to look at specific data points and to request relations that were not available via the UI. Furthermore, data was loaded into MS Excel sheets allowing us to interact with it and create mockups of potentially interesting visualization problems.

In this stage, two main data characteristics surfaced that would be interesting to visualize. The first is the existence of diagnostic gaps, such as FM1s of type PCS or Other. Often multiple gaps were found in the top 100 FM1s for an FC, resulting in undiagnosable Faults and unnecessary machine downtime. The second is the existence of FM1 similarity in the data, where a group of FM1s has overlapping sets of Faults. A group of related FM1s based on overlap can be seen as an FM1 cluster. On a small scale, overlap between the FM1s can be visualized with a Venn diagram, as is shown in Figure 11. However, the amount of interesting FM1s in an FC was in the hundreds and thus another solution would be more viable.

The FM1 connections were further analyzed for FC003, where the data looked like a weighted undirected graph. On closer inspection, the graph is a network of relatively small (3-10 FM1s) dense disconnected graphs (FM1 clusters). A way to abstract away from individual Faults was constructed, it describes the data solely as connections between FM1s. This translation step reduces the amount of elements that need to be visualized, and was performed as displayed in Figure 11. The overlapping Faults for each FM1 are divided by the total number of associated Faults, resulting in a relative overlap percentage as calculated following Definition 11. Symmetric overlap leads to a single undirected edge between two FM1s and allows us to further consider this data as a large connected graph.

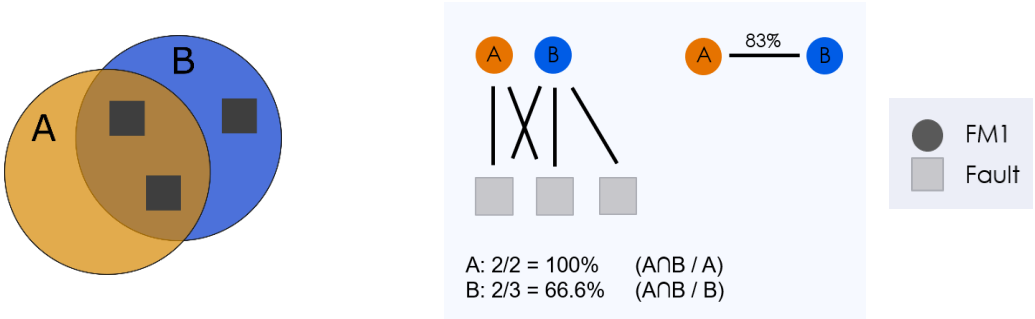

Figure 11: Venn diagram (left) and node-link representation (right) of a small set of FM1s and Faults

5.2 Iteration 2 - Layered approach

The orientation phase resulted in a conceptual design sketch. Several layers of the Smart Diagnostics System were to be visualized, each taking their own part in the tool. A top-down approach was considered to be most effective, following Shneidermann's mantra. Alongside the three different views, information was displayed on the side about the user's selected items. Small multiples or multiple views were considered as an option as well (Roberts, 2007). It was not a viable approach, since the number of required small multiples was too large and multiple properties were to be changed between views. Therefore, it was better to keep everything together in one big overview with data lists on the side, containing additional details. The notion of consistency between multiple coordinated views was kept and integrated in the layered approach.

First, the user needs to set several filters that narrow down the amount of data and make it manageable for visualization purposes. Next an overview of the FCs is given, from which the user can explore relations between FCs and select the FC they are interested in at the moment. Once an FC is selected the selection of interesting cases takes place in the overview of FM1s. Here, the FM1s and their underlying relations, through overlap of Faults, are shown. Diagnostic gaps and FM1 clusters can be identified and selected for further investigation. When an interesting case is found, the user can look for explanations by focusing on the particular FM1. All connections the FM1 has are shown, next to a popup that contains more information on the FM1. Each layer will be discussed in more detail below and a sketch is provided for the initial visualization design ideas.

Filters

A subset of filters has to be selected, as not all FMkb filters can be included in the visual overview due to API limitations and time constraints. Keeping the set of filters simple helps to retain the user's focus on the most important elements and allows for easy data exploration. The initially selected filters are Product Line, Functional Cluster, Date Range, FM1 Type, and Overlap Threshold. A mockup of the filters is displayed in Figure 12.

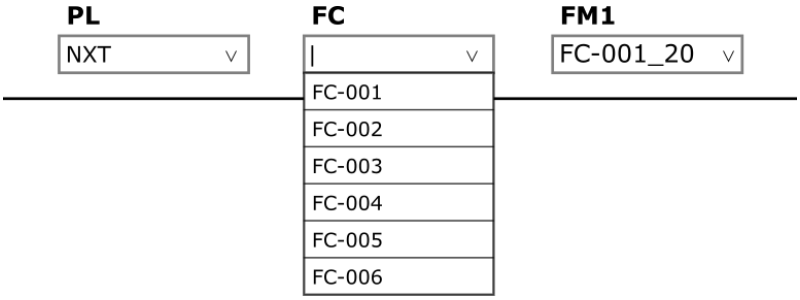

Figure 12: mockup of the filtering setup

Overview of FCs

An overview of the FCs was conceptualized as the entry point of the tool. It provides an easy way to select an FC the user is interested in and visualizes the relations between FCs. The relation between two FCs is to be computed from the relations of FM1s in an FC with FM1s from the other FC. The amount of FM1s belonging to the FC is corresponding to its area size. A list of FCs is shown on the right, presenting basic

information about the FCs and SUs, and aggregated statistics of their corresponding FM1s. A mockup of the overview of FCs is displayed in Figure 13.

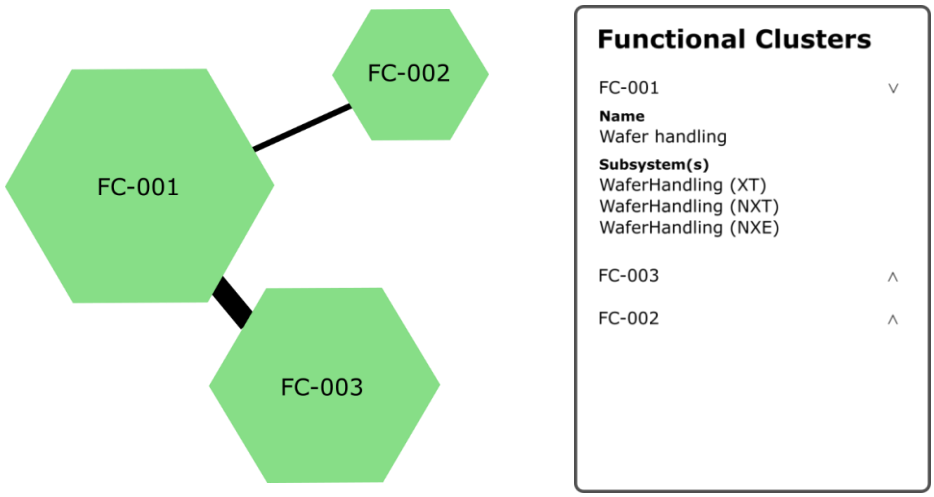

Figure 13: mockup of the overview of FCs

Overview of FM1s

Next, the overview of FM1s is introduced. It is used to further explore the diagnostic landscape within an FC, allowing the user to find and select interesting groups or individual FM1s. The FM1 data and the relations between them form a graph data structure. Graph visualization has two main visualization methods as we saw in Chapter 4, namely the node-link diagram and the matrix representation. Both methods were explored and considered as potential solutions for this layer.

A zoomed out node-link diagram looks a bit like a heatmap of FM1s, where a dense region indicates lots of overlap and thus an FM1 cluster. The position and size of FM1s could indicate their importance in the set, and the color can communicate the FM1 type. In Figure 14, we show the progress made from abstract idea (left) to concrete graph layout (right).

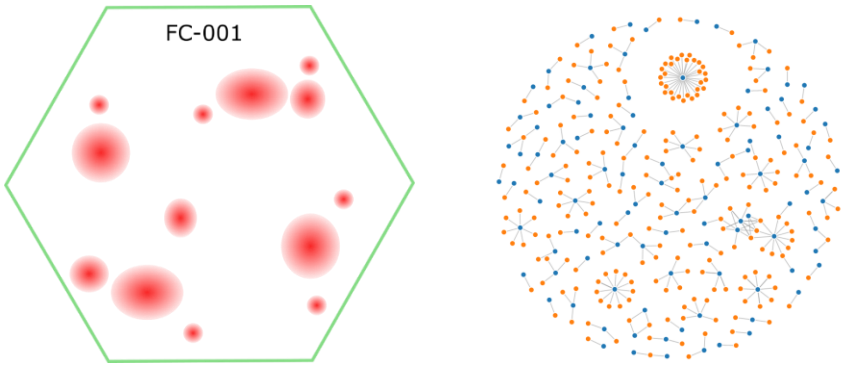

Figure 14: the progress on the node-link diagram representation
Left: abstract idea, right: concrete graph layout with D3JS fore-directed layout algorithm

Furthermore, a matrix representation was considered as overview of the FM1s. The rows and columns correspond to the set of FM1s. Each cell contains information about whether there is a connections between the corresponding FM1s and how strong the connection is. Applying color scaling gives a quick overview of the density of the graph and a rough idea of the FM1 clusters. Reordering the rows and columns can be a helpful tool to show the underlying structure in the data. An impression of what a matrix representation can look like is given in Figure 15.

	A	B	C	D	E
A	100%	85%	<80%	<80%	<80%
B	85%	100%	84%	<80%	<80%
C	<80%	84%	100%	90%	90%
D	<80%	<80%	90%	100%	92%
E	<80%	<80%	90%	92%	100%

Figure 15: mockup of the matrix representation of FM1s
Image courtesy of the FMkb team

Focus on FM1

When an FM1 is selected in the overview of FM1s, more information needs to be shown. This is done in two ways, namely through a visual representation and an information popup belonging to the FM1. See Figure 16 for an initial mockup of this view. In the visual graph, relations are displayed from the selected FM1 to the FM1s it has overlap with. Properties like the node size, an colored area around the FM1 and positioning were explored as means to convey additional information. In the list, we display relevant information like the description, status and type of the FM1. Next, a list of the similar FM1s is given, allowing the user to go into detail about amounts of overlap and investigate what FM1s have a higher overlap percentage than the set overlap threshold (blue horizontal line).

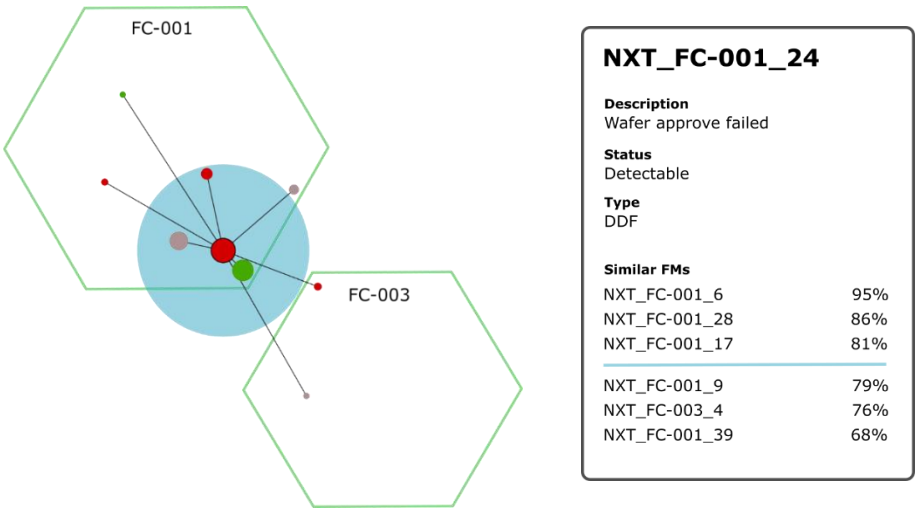

Figure 16: mockup of the 'focus on FM1' view, with a visual (left) and data list (right)

5.3 Iteration 3 - Graph visualization

The focus group was consulted on the initial conceptual design, the collected feedback resulted in a shift of focus to the overview of FM1s as a starting point. The focus on FM1 view was also important for investigating details and specific cases, since individual FM1s could be selected and investigated in the information modal. Consistency between visual language of the two views should be high, preferably without changing many visual properties and showing animations for the properties that do change. The overview of FCs is an interested use case, but was discarded for now. Most target users work within an FC and do not look over the border to other FCs. Next to that, visualizing relations between FCs results into a mapping of the machine, which is already known to most users.

A first prototype was constructed as an experimental tool in which designed features could be tested with production data and end users. The prototype is implemented as a web application featuring the D3JS library, making it easy to work with visual elements inside a canvas. See Figure 17 for a glance of the first functional prototype. In this phase, data for a selection of FCs was preloaded, eliminating the need for API calls and having quick loading times.

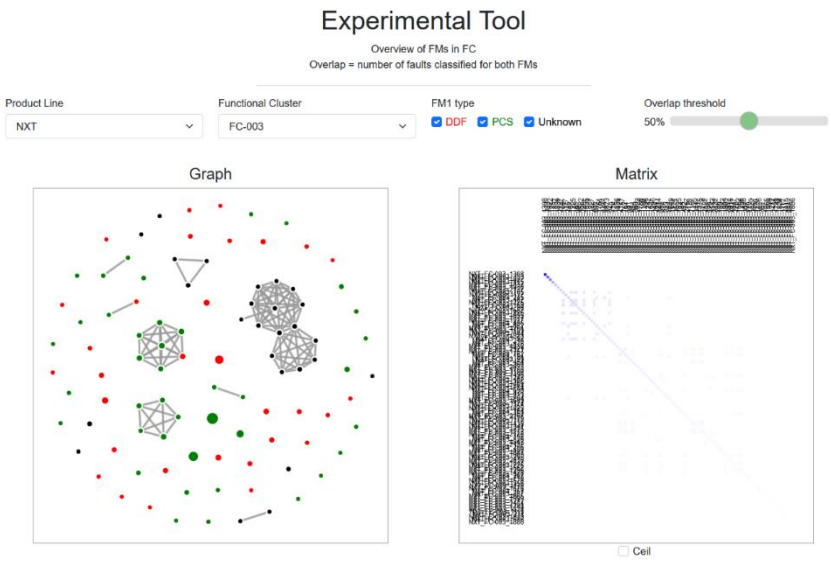

Figure 17: first functional prototype

Graph visualization stood at the center of this first prototype, with a node-link diagram implemented on the left and a matrix representation on the right. The advantages and limitations of these two visualizations were tested with real data and experts, but the decision for one over the other was postponed to a future iteration. After a demonstration and review session with experts, several improvements were incorporated. The major one being the addition of FM1 selection, highlighting the FM1 in the graph and matrix. It also included the information popup, showing basic data properties and similar FM1s. See Figure 18 for the updated prototype.

The tool was named FMvet, an acronym for Failure Mode visual exploration tool. The name is inspired by the Failure Mode knowledge base (FMkb), which it serves as a visual extension.

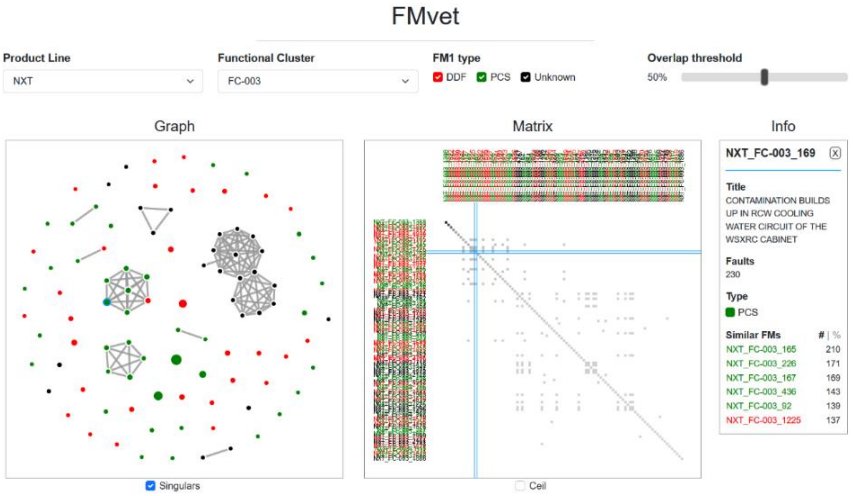

Figure 18: prototype with several conceptual improvements, most notably FM1 selection

In this iteration, the different properties for visualizing FM1 importance and FM1 overlap in the graph were explored. Examples of properties that may convey information are the node size, node location, node color/opacity, link width, link distance, and link color/opacity. Furthermore, the color scheme for the FM1 types was updated. The colors green and red should be avoided as they contain associations with good and bad. Therefore, the blue/orange/grey scheme is more neutral and easier on the eyes.

A look into symbols, glyphs and icons (Li et al., 2010; Borgo, 2013; Fuchs et al., 2016; Maguire et al., 2012) was conducted as well. A mood board with potential visual designs was constructed, but it had little results. Conveying information with varying symbols has limitations like the perception of size, limited number of options, and the availability of properties that can already be adjusted. An example being that the comparison between the square, triangle and circle area is hard to accurately estimate with the human eye. Furthermore, a tooltip was added to the graph that displays more information about an FM1 (the ID and description) when the mouse hovers over it. See Figure 19 more details.

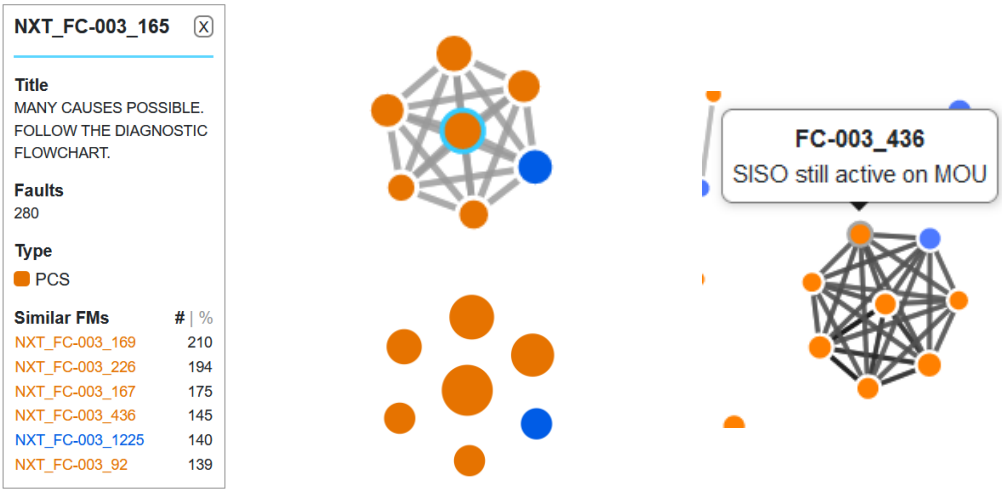

Figure 19: the selected FM1's info (left), initial cluster visualization (middle) and FM1 tooltip (right)

5.4 Iteration 4 - Cluster view

The Cluster view was added as an intermediate step when going from the overview of FM1s to a singular FM1. The description of an FM1 cluster is given in Definition 12. This change is in line with Shneidermann's mantra of "Overview first, zoom and filter, then details on demand". The Overview view provides an overview of the FM1s in the total set, the Cluster view provides particular zooming and filtering on a smaller set of FM1s. Last, the selection of one FM1 visually shows details in the graph and a list of more data properties.

Often an FM1 cluster conveys information as a group and the cluster could be also investigated on this level. Interesting aggregation properties are the amount of FM1s the cluster consists of, the amount of distinct Faults that are associated with it, the amount of affected machines, and the total downtime duration. The FM1 clusters were computed from the graph data with a bread-first search algorithm. Figure 20 shows what the Cluster view looks like for a particular FM1 cluster.

The limited amount of FM1s in a cluster is better representable as a matrix, thus now we can see the matrix' strengths as well. The matrix cell gradients give a good indication of the relations between FM1s at a glance. Sorting the matrix provides valuable insights into which FM1s are strongly related. From this view, the absolute overlap between FM1s could easily be investigated. However, relative overlap is often more telling, therefore the overlap in relation to the total amount of Faults is displayed in the selection column as well. For more information on the different types of overlap, consult Definition 11.

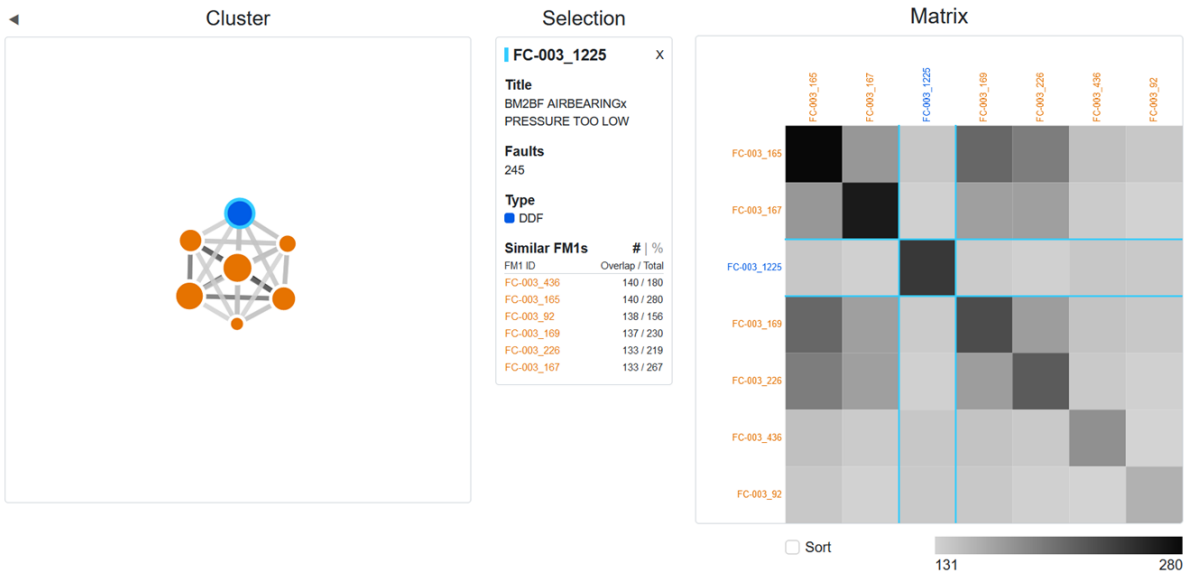

Figure 20: one selected FM1 cluster in the Cluster view of the prototype

FM1 overlap

Many different approaches to visualizing overlap were designed, since this was the most important and challenging bit of information to convey visually. The options are depicted in Figure 21. Properties that vary between the visualizations are node size, inner node size, node position, link length, link color/opacity, link overlap (overlay or side by side). Furthermore, ideas for bar charts were generated. In the end, it was decided to visualize the overlap between FM1s with the inner dark nodes and the link

opacity. These methods preserve the consistency between the Overview and Cluster views whilst showing symmetric and relative overlap.

As the number of links between FM1s in a cluster quickly increases due to the density of the small graphs, solutions to limit the amount of edge crossings were investigated. Among which a force-directed layout algorithm, partially drawn links for directed graph edges, and highlighting the edges belonging to an FM1 when it is selected. Having less cluttering and link crossings makes dense graphs easier to understand and explore (Burch et al., 2011). In the end, it was chosen to rely on the force-directed algorithm to create a sensible layout for the nodes and edges in an FM1 cluster. Furthermore, the nodes could be dragged to reposition them and the edges could be highlighted by selecting an FM1.

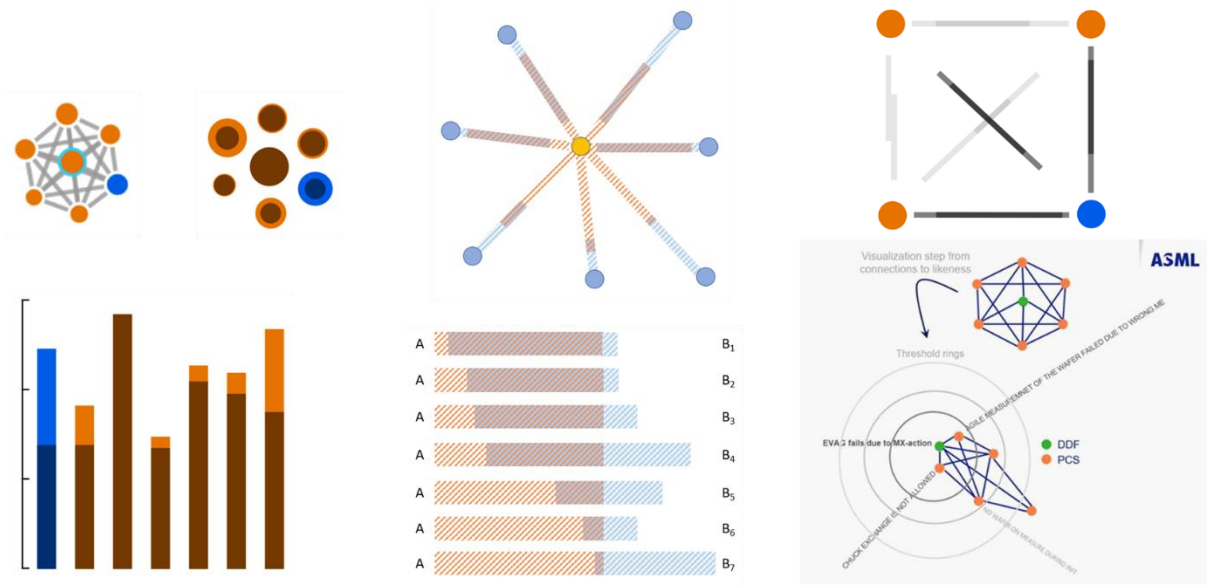

Figure 21: brainstorm of the various approaches to visualize overlap between FM1s

DDF creation support

The support of DDF creation was briefly analyzed as a side-project with promising ideas. It would let the user play with the creation of DDFs and see what the impact on an FM1 cluster is. This part was put on hold as only a limited portion of users would find this useful. Next to that, the tool was meant to be a viewer and DDF editing or addition is performed with the help of other tools and by other users. The idea was to have a visual indication (light blue color) of the DDFs impact in the graph, as shown in Figure 22.

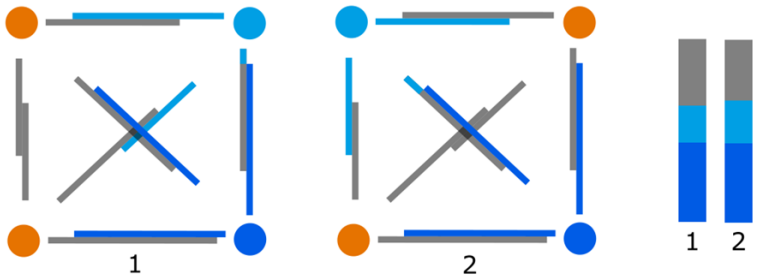

Figure 22: brainstorm of the DDF creation support ideas

5.5 Iteration 5 - Canvas and lists

In this iteration focus was brought to the graph visualization by making it central and by having sortable data lists on the side, namely the FM1s list and the FM1 clusters list. FM1s list presents the most relevant FM1s on top and provides valuable information like the description, amount of Faults and amount of overlap. The cluster list presents a priority queue for what groups of FM1s to look at first, indicated by the amount of distinct Faults associated with it. Four filters were added in this stage, namely Subsystem, FM1 IDs, Fabs and Singulars. These filters were requested by experts and are used to tweak the set of filtered FM1s to their liking. Singulars could now be removed from the dataset to provide a clear focus on FM1 clusters only. See Figure 23 for what the tool looked like after iteration 5.

The decision was made to only have a node-link diagram as graph visualization. The matrix representation was removed, since it only works well for small data sets and consistency between views is hard to maintain with a matrix only used for FM1 clusters. Next to that, all information conveyed with the matrix can also be read from the node-link diagram and FM1s list. The relations between FM1s are shown in the graph, while the overlap column in the FM1s list shows a column of the matrix.

Selection mechanics were updated for clarity and to communicate additional information. An FM1 is selected by clicking on a node in the graph, and deselected by clicking on the empty space. Next to that, an FM1 can be selected in the FM1s list. The same holds for selecting an FM1 cluster in the FM1 clusters list. Once selected, the corresponding FM1 in the graph gets a circular border. Cyan was chosen as the selection color, since it draws attention and is differentiable from the other colors in the tool. Later on, the decision was made to switch to neon green, as it would interfere even less with the blue DDF color.

The work by Van Wijk & Nuij (2003) was consulted for information on the best approaches to canvas zooming and panning. However, the build-in zooming and panning functionality from D3JS was used with minor adaptations. It is largely in line with the presented related work and simplified implementation.

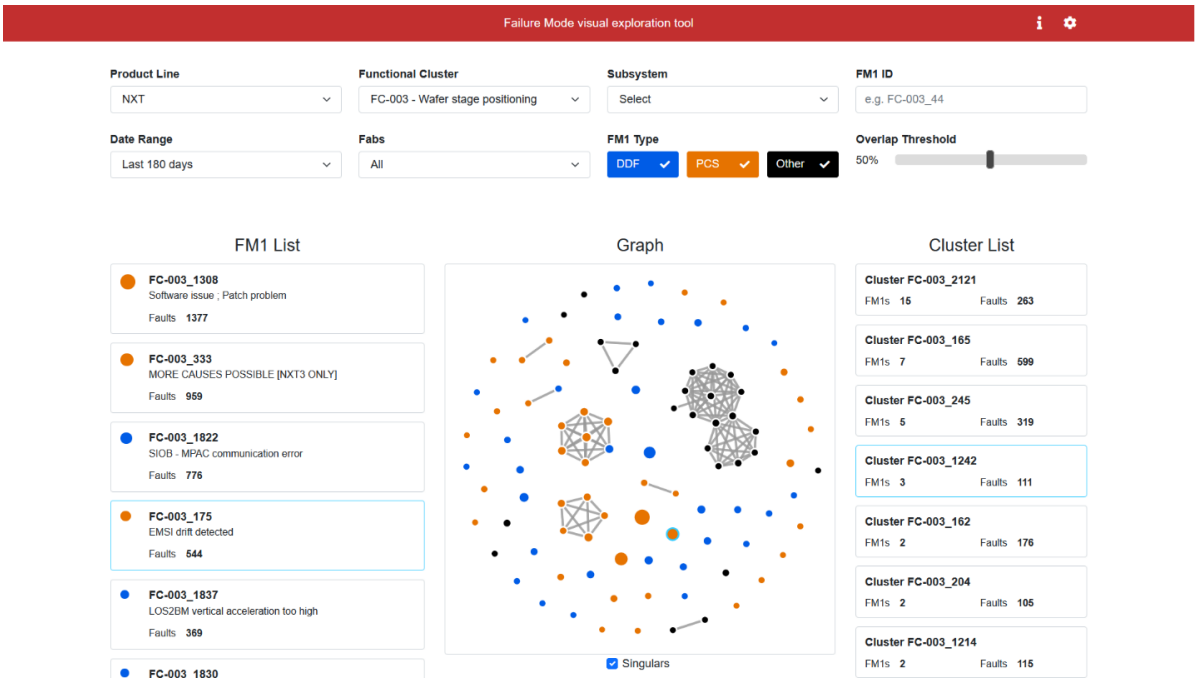

Figure 23: state of the visualization after iteration 5

5.6 Iteration 6 - Navigation

Iteration 6 mainly concerned quality of life improvements to the prototype. The three layered views come together nicely, but an indication of what view you are currently in and navigation between the views should be added. The new navigation tabs are shown in Figure 24, and include the Overview, Cluster, and Symptoms views. The currently selected view is indicated with a red background. A spot for the Symptoms view was reserved, it will be discussed in more detail in the next iteration.

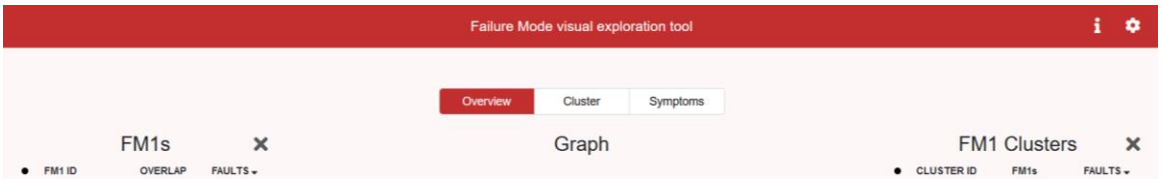

Figure 24: navigation pane having tabs for the Overview, Cluster and Symptoms views

First, the data lists were improved by making them more space-efficient, sortable on all columns and contain an icon for each item. An overlap column was incorporated in the FM1s list as well. Second, the selection methods and its visual representation were improved. The selected item is now indicated by a vertical bar in the list, a color change to its icon, and a visual representation in the graph. The selection indicator in the graph matches with the indicator in the list. For example, the selected FM1 cluster is shown with a large green square around it, thus the same green square is used in the FM1 clusters list. Furthermore, cross-referencing between FM1s and FM1 clusters is added. This means the green square is added to the FM1s which belong to the selected FM1 cluster. The other way around, a green circle is added to the FM1 cluster in which the selected FM1 is contained. Third, standard graph controls were added to show feedforward of the possibility of zooming and panning in the graph. The icons are displayed on the bottom right of the graph and function as follows: left icon resets the zoom to regain an overview of all FM1s, the middle icon zooms out, and the right icon zooms in. The additions introduced in this iteration are shown in action in Figure 25.

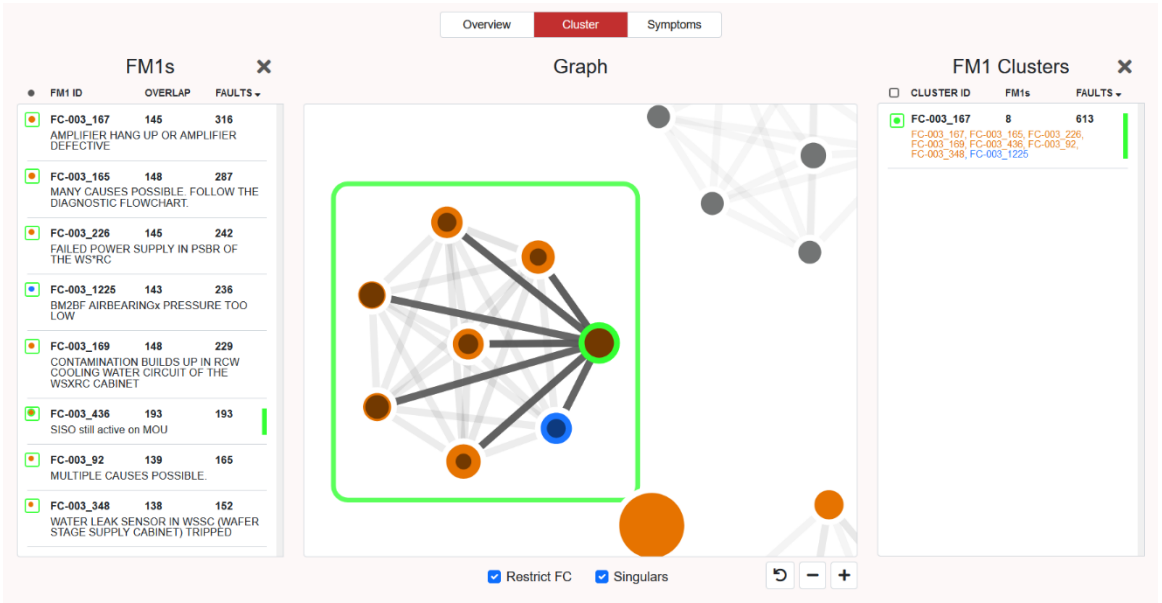

Figure 25: FMvet in the Cluster view with multiple selections applied

5.7 Iteration 7 - Symptoms

In a demonstration session with mostly Customer Support colleagues, a desire for the visualization of the underlying Symptom fingerprint (Definition 14) for FM1s was voiced. Several SQL queries were performed on the database to investigate what this data looked like and how it could be best visualized. Preprocessing steps were applied to transform the raw data into interesting visualization properties. The Symptom data was retrieved per FM1 and normalized in relation to its Faults. Next, all FM1s' symptom codes were aggregated and averaged to get an overview of the Symptoms belonging to the selected set of FM1s. The Symptoms could be sorted for all FM1s, the FM1s belonging to the FM1 cluster, and for an individual FM1. Once presentable data was obtained and computed, it could be visualized as a large matrix containing small bar charts for the frequency of a Symptom in relation to the FM1. The initial design is displayed in Figure 26.

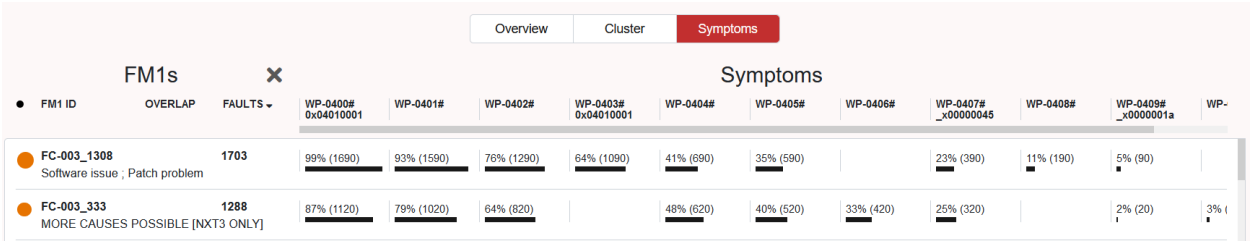

Figure 26: initial prototype of the Symptoms view

Another feedback session with users from CS, resulted in several suggestions for improvements to the Symptoms view as shown in Figure 27. First, an option to filter symptoms by a text phrase was added at the top left. Next, the individual Symptom columns were made sortable, allowing the user to investigate whether the Symptom is significant for the selected set of FM1s and could be a potential candidate for an expert rule. Last, the bar chart colors were matched to the FM1 type color, for easier comparison between the different types of FM1s and to make horizontal rows easier to identify. Future work on the Symptoms view should include a way to display or highlight active expert rules for the FM1s.

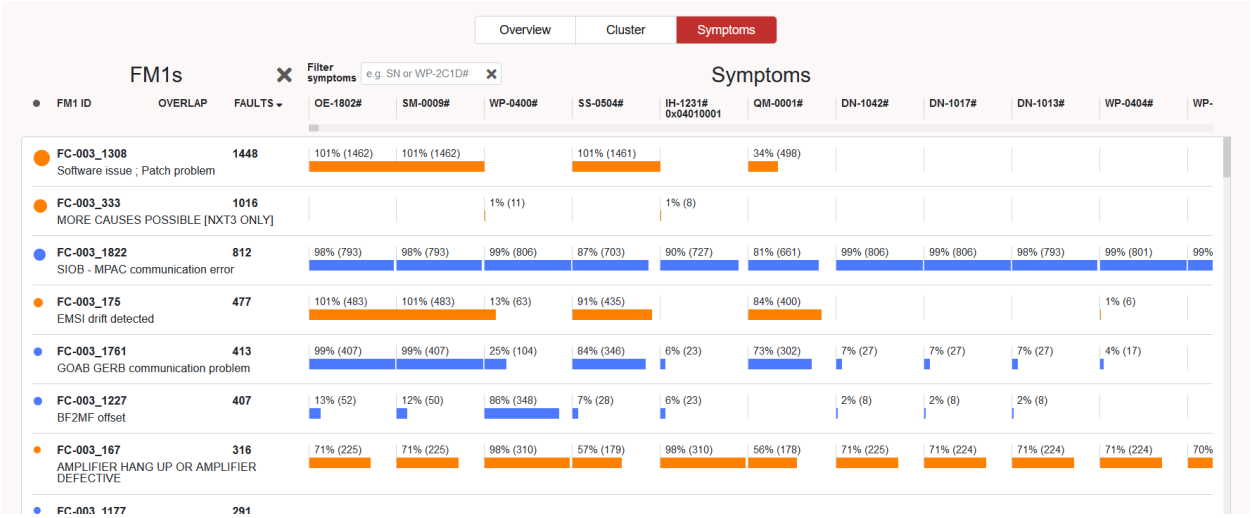

Figure 27: final design of the Symptoms view in FMvet

5.8 Iteration 8 - Neighborhood

The last iteration contains a major feature update (neighborhoods) and several polishing improvements, as can be seen in Figure 28. Relations between FCs and SUs can be investigated with the multiselect option in the filters section. However, it is a manual process and still hard to see cross-FC and cross-SU FM1 connections. Thus, an option to show the neighborhood of the selected FM1 set was introduced. The neighborhood includes the single hop neighbors of the selected set, without taking imposed filtering criteria into account. For more information on the notion of the FM1 neighborhood, see Definition 13. Ideally, the primary (filtered) and secondary (neighborhood) FM1s will be shown in two separate lists. Due to time constraints and complexity of adding this feature now, it was considered to add the secondary FM1s to the main list with a differentiating style. Other ideas that were considered surrounding the neighborhood options were to add smart grouping to allow focus on some FCs (discussed under FCs and SUs), and to add smart suggestions. Smart suggestions display hints to the user on which filters to adjust (e.g. add FC001 to the FC multiselect), and could be incorporated in a future version.

FCs and SUs

The corresponding FC and SU of FM1s in the graph are not easy to analyze with the current prototype. A need for the visualization of FCs and SUs was voiced by the focus group of users as well. As a solution, the FCs and SUs lists and color modes were introduced. They feature computed aggregations of FCs and SUs for all FM1s in the graph, displayed as the number of FM1s and number of Faults belonging to it. The lists have similar sortable columns as the FM1s and FM1 clusters lists. One or more FCs can be selected from the list, resulting in the corresponding FM1s being highlighted in the graph, whilst the other FM1s are greyed out.

This idea is taken a step further by visualizing the FCs and SUs with colors in the graph. When the color mode is switched from FM1 Type (default) to FC, several things change. Each FC gets a color assigned to it from a predefined color array. A predefined array was used to make the color modes deterministic and to avoid interference with the selection color. Next, all FM1s in the graph belonging to an FC are colored in the same style. This way, a visual overview of the FCs and relations between them is displayed. The same can be done for SUs.

Polishing

The integration with FMkb was implemented in this iteration. For FMkb to FMvet integration, filters are made readable from the FMvet URL. In FMkb, a button with the link to FMvet can be included, for which the filters are automatically transferred. For FMvet to FMkb integration, two buttons are added to the FMvet interface: 1) open one FM1s stats and 2) open one FM1 cluster's stats. The buttons are located in the FM1s and FM1 clusters lists, visually represented by an external link icon. Further details on the integration are discussed in the Final Visualization chapter.

Furthermore, modals with information about FMvet and global settings were added. They are accessed via the corresponding icons in the top bar of the tool. The information modal contains several notes and instructions on the usage of FMvet, as well as contact information in case of issues or feature requests. The global settings modal contains a list of configurations that are used sparsely, and allow the user to tweak their experience with FMvet. The global settings are persistent through multiple sessions.

Last, there is an addition of the FM1 type color legend to the bottom left of the graph, as the colors were sometimes hard to grasp by first-time users.

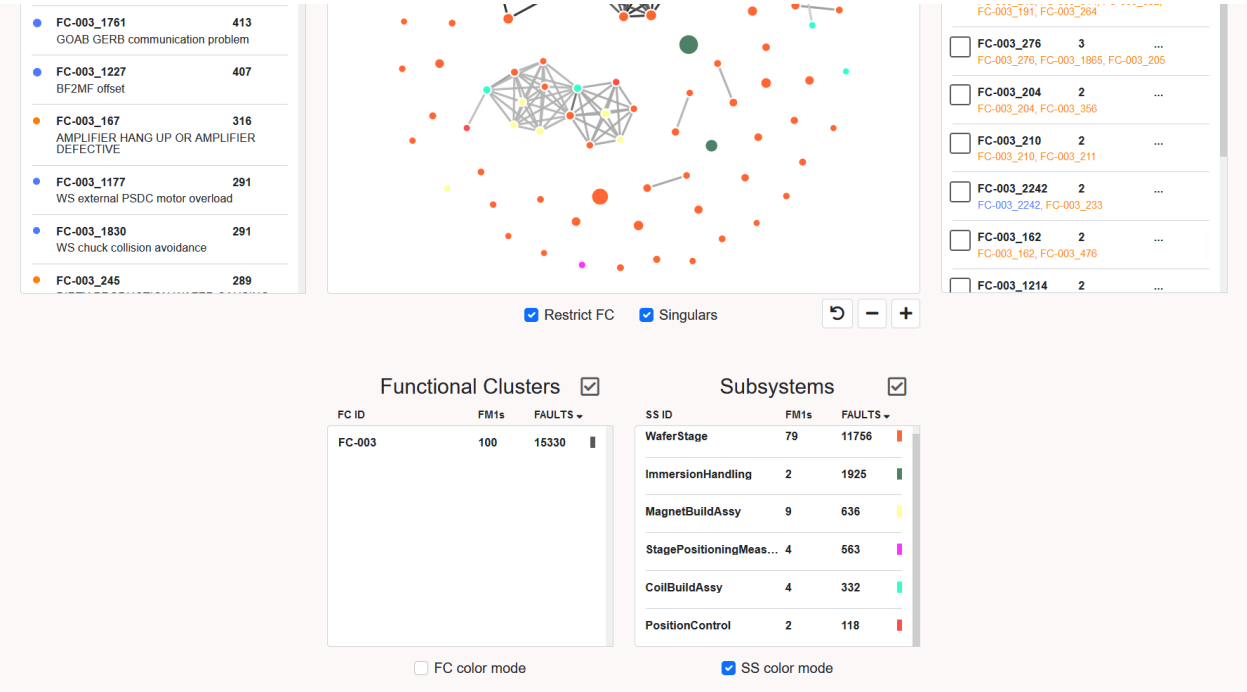

Figure 28: the Functional Clusters, Subsystems, and color modes in the prototype

Chapter 6

Final Visualization

The final visualization is the result of the extensive design process and is discussed in detail in this chapter. First, we address the conceptual design including the filters and the three main views on the data, namely the Overview, Cluster and Symptoms views. Next, we reflect on the design with regards to the performed analysis, and to evaluate whether the user's activities and tasks can be completed with it. Last, a video demonstration of the prototype is linked and the recommended integration into FMkb is presented. Some figures are included in the text, but the majority is moved to Appendix C.

6.1 Conceptual design

Figure 30 presents an overview of what parts the tool consists of. On top, the floating navigation bar displays the title and contains buttons for more information and settings. Clicking on the information button displays a popup with general information about FMvet, as shown in Figure 53. Below that, we have the filters section for selecting data and for changing visual settings. Next are the navigation tabs, allowing the user to quickly switch between the three different views (Overview, Cluster, Symptoms). The three views are shown side by side in Figure 30, 31 and 32. The focus of each user may vary, dependent on his interests and stage of the project. Hence the main part of the interface has different levels of detail, specific features and other visual aspects dependent on the selected view. The views are further explained below, where we show that consistency in visual language between them was important. The main part of the Overview and Cluster view consist of the Graph in the center, the FM1s and the FM1 clusters on the sides, and the FCs and the SUs on the bottom. For a schematic overview of the potential flows through the tool, see Figure 29. The filters have a special status, as they can be adjusted independent of the chosen views.

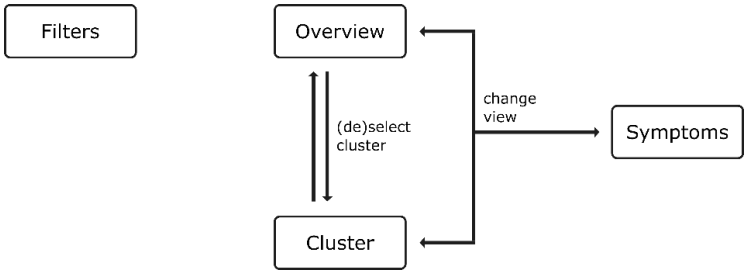

Figure 29: schematic overview of the flow through the views of FMvet

Video demonstration

A video was created to demonstrate the prototype. It better captures the functionality, interaction and flow through the tool than text and images can. The access information is listed below.

URL https://vimeo.com/584645349

Password Q?DaD2XR

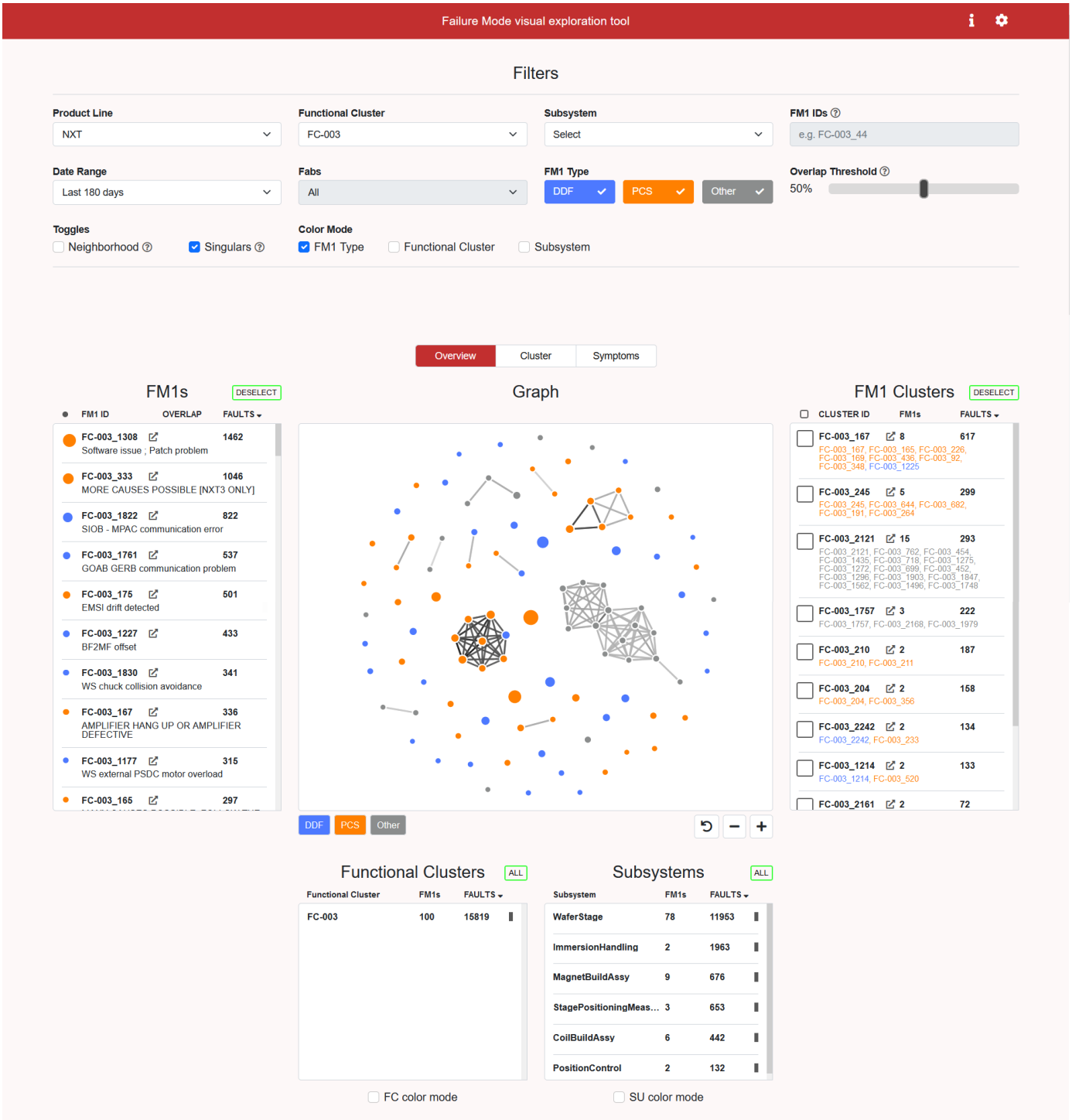

Figure 30: FMvet prototype in the Overview view

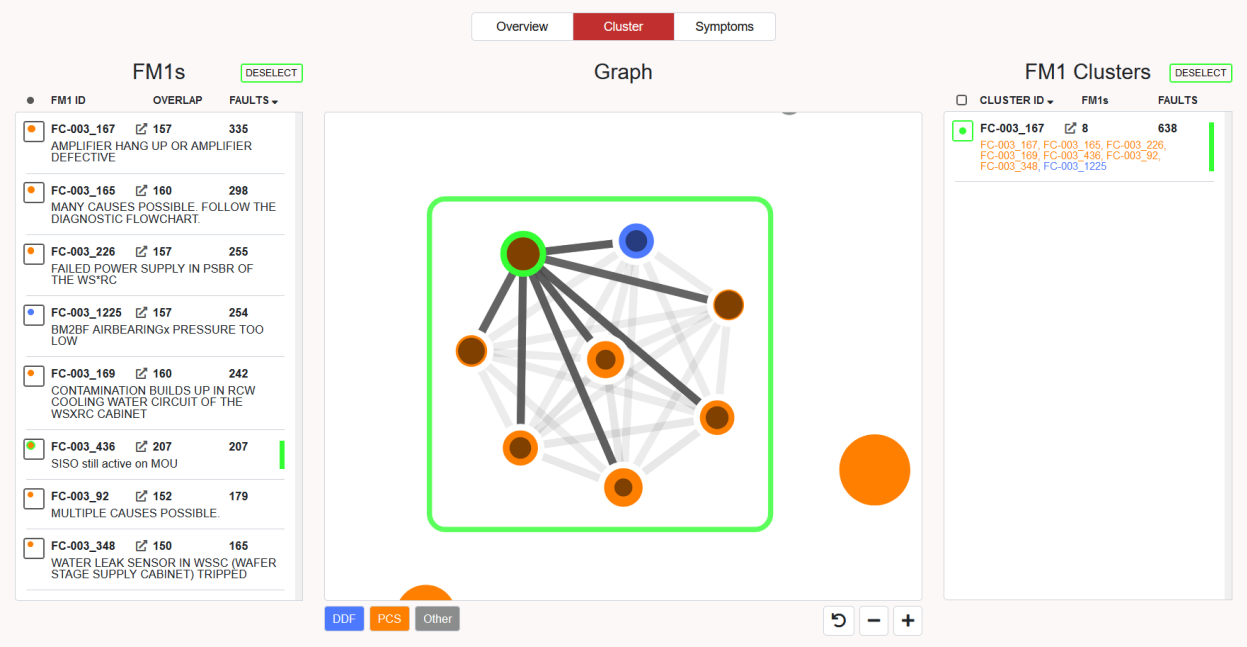

Figure 31: FMvet prototype in the Cluster view

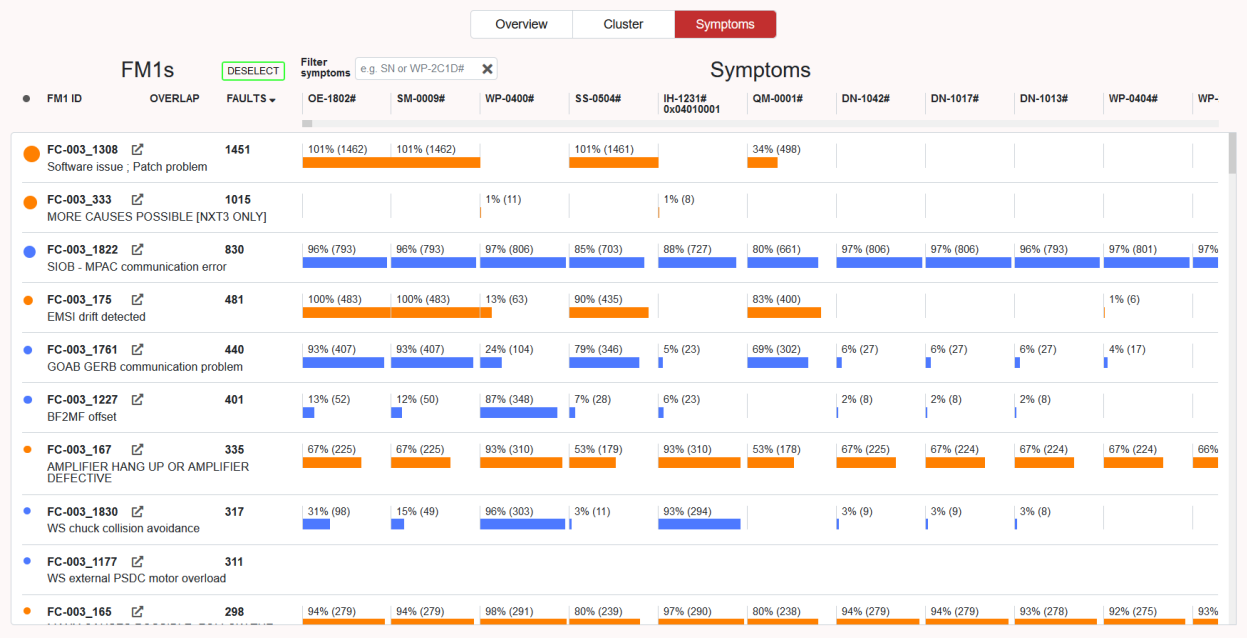

Figure 32: FMvet prototype in the Symptoms view

6.1.1 Filters

Filtering is an integral part of any tool that works with data, it allows users to select specific parts of the dataset and to highlight cases they are interested in. The main type of data we select in this tool are which FM1s to load, and the relations and statistics associated with them. The filters are divided into four categories, namely row filters, statistic filters, visualization settings, and global settings. Two different states of the filters section are shown in Figure 42 and 43.

FM1 filters

As the name suggests, FM1 filters limit the number of FM1s to be returned. Setting these filters results into a selection of FM1s being made from the total set of FM1s, following the faceted search design pattern. The potential filters are:

- Product Line (select: XT, NXT, NXE, SRC);
- Functional Cluster (multiselect: 90 options);
- Subsystem (multiselect: 40 options);
- FM1 IDs (text input);
  Multiple FM1 IDs can be entered, separated by a comma.
- Neighborhood (checkbox).
  Toggle indicating whether the neighborhood of the selected set of FM1s should be included or not. See Definition 13 for more information.

A special group of FM1 filters is incorporated in the tool. These filters do not affect the data loaded from the API, but are applied in the preprocessing step just before visualizing the data. A differentiation is made to allow the user to quickly change the most used data filters and visual settings without prompting a data reload. Therefore, these filters can also be referred to as visualization settings, since they allow users to further change what is shown and how it is shown. The visualization settings are:

- FM1 Type (checkboxes: DDF, PCS, Other);
- Overlap Threshold (slider: 0% to 100%);
- Singulars (checkbox);
  Enabling this toggle allows the singular FM1s to be part of the dataset. It means that the FM1 has no connection to other FM1s in the selection and is therefore a singleton cluster on its own.
- Color Mode (switch: FM1 Type, Functional Cluster, Subsystem).
  It lets the user switch between three color modes, each highlighting different data properties.

Fault filters

The Fault filters limit the aggregation of Fault properties belonging to the FM1s. For example, they only select Faults belonging to an FM1 which occurred within the specified Date Range or happened at a particular customer Fab. Therefore, these filters change the associated statistics of each FM1, but do not affect which FM1s are included in the output. The filters are:

- Date Range (select: last 30/90/120/180/360 days, all time, custom range);
- Fabs (select: all, customer fabs only, ASML factories only, individual fab).
  This filter was not feasible to implement yet, as it required updating API calls.

Global settings

A more general group of settings is available as well, allowing to tweak specific settings of the tool to their own preference. The settings are hidden behind the settings icon in the floating top bar, as they are only used sparsely. The settings popup is displayed in Figure 54. The global settings are:

- Data: max number of FM1s to load;
- Data: min number of overlap between FM1s;
- Text: FC prefix visibility;
- Text: FM1 clusters include a list of its FM1s;
- Graph: Cluster link color intensity is variable;
- Symptoms: Filtered pattern;
- Symptoms: Normalized sorting.

6.1.2 Overview view

The Overview view is the starting point for most use cases. In this overview the exploratory phase of a task is performed. For example, the user might search for interesting data points in the graph or get an overview of which FCs are related to each other. The view concerns all the main parts of the tool, with extra focus on the graph and FM1 clusters. See Figure 30 for what FMvet looks like in the overview state. In Figure 44, 45 and 46 other states of the Overview view are displayed with various selections applied.

Graph

The graph is a central element of the tool, bringing all the surrounding lists together in one graphical overview. In this graph, the FM1s are represented as nodes and the FM1 connections between them as edges. All specific properties and features of the graph are listed below. See Figure 49 for a graph close-up.

- The FM1s are shown as nodes using circles. The area of an FM1 is related to the number of associated Faults, whereas the color indicates what type is has.
- FM1 connections are shown as links (straight lines), forming visual clusters of related FM1s. The links have a gradient applied to them, indicating the strength of the connection.
- Node and link positioning is based on a force-directed layout algorithm implemented and adjusted from the D3JS library. It optimizes the positions according to the following heuristics: limit the amount of edge crossings, equalize edge length, equalize space between nodes, and bigger nodes are central whilst smaller nodes are positioned on the outside.
- A tooltip with FM1 info is shown when the mouse hovers over an FM1.
- A color legend of FM1 types is displayed on the bottom left (DDF blue, PCS orange, Other grey).
- Zoom adjustments controls are displayed on the bottom right. The buttons allow to reset the zoom, to zoom out, and to zoom in.
- Zooming and panning is supported in the canvas, controlled by the mouse.
- Clicking an FM1 in the canvas selects it. Clicking the empty space in the canvas deselects the currently selected FM1.

FM1s

In the Overview view, all FM1s are displayed in the FM1 list. Each FM1 consists of a circle, ID, description, amount of overlap, and amount of Faults. The list can be sorted on the properties: ID, overlap and Faults

(default). The circle is corresponding to this FM1's circle in the graph, thus shows the FM1 type through its color and amount of Faults through its area. The external link button can be clicked to open this FM1 in the FMkb interface, allowing for more details to be looked up. When clicking on an FM1 in the list, it gets selected. The selection is show in the list through green indicators and in the graph with a green border around the node. An example of one selected FM1 in the FM1s list is shown in Figure 33.

Figure 33: the FM1s list with one selected FM1

FM1 clusters

The FM1 clusters list is similar to the FM1 list. Instead of the overlap we show the number of FM1s associated with the cluster and instead of the description we show a list of all FM1 IDs in this cluster. The list can be sorted on the properties: ID, FM1s and Faults (default). When a cluster is selected, the square turns green and the cluster is highlighted and zoomed to in the graph.

FCs and SUs

The FC and SU lists function in a similar way, and always concern the entire dataset (without taking the selected FM1 and FM1 cluster into account). The sortable properties are: ID, amount of FM1s (default sort), and amount of Faults. These lists give a good understanding of which FCs and SUs play a big role in the selected FM1 set. An FC can be selected to highlight the FM1s associated with it. To investigate relations between the FCs, the color mode can be switched. A color is then assigned to each FC, in the graph the FC's corresponding FM1s are painted in this color as well. The default state of the FC and SU lists is displayed in Figure 50. Different color modes and selections are shown in Figures 51 and 52.

6.1.3 Cluster view

For further investigation, the user can switch from the Overview to the Cluster view. Navigating to this view is done by selecting an FM1 cluster in the corresponding list or by clicking the Cluster navigation tab. Focus is brought to the selected cluster in various ways. Several visual properties and features are updated when switching to the Cluster view, in the following texts the exact changes are explained per element. In this view, the main elements are the graph and the FM1s, as can be seen in Figure 31.

Graph

First, the selected FM1 cluster is highlighted in the graph with a green border. The zooming and panning settings are adjusted to show the cluster enlarged and centered in the canvas. When a particular FM1 in the cluster is selected, the corresponding node is given a green circular border and a darker inner circle appears inside the node. The inner circle visualizes the overlap between the selected FM1s and its related FM1s, as the area of the inner circle corresponds to the percentage of overlap. Next to that, the edges that are connected to the selected FM1 are highlighted, whereas the other edges fade out.

FM1s

In the Cluster view, only the FM1s belonging to the selected FM1 cluster are shown in the FM1s list. The overlap column will be filled when a particular FM1 is selected, it shows the number of overlapping Faults with the selected FM1.

FM1 clusters

The FM1 clusters list is reduced to only the selected FM1 cluster, with a green square on the left indicating it is the selected one.

6.1.4 Symptoms view

In this view, the focus is solely on the symptoms matrix, which is an expansion of the FM1s to the left. Other parts of the tool are still visible when scrolling down, if there is a need to simultaneously consult the graph or other parts. The Symptoms view helps to better understand the underlying data similarities between FM1s. Questions like: why do these FM1s look alike and does that make sense? and are the current expert rules of high quality and what potential new rules can be constructed?, are answered here. See Figure 32 for the initial state of the Symptoms view. Other states with selections and filters enables are displayed in Figure 47 and 48.

FM1s and Symptoms

The FM1s list is expanded to the right with a Symptoms section. A list of all Symptoms that have occurred for the selected set of FM1s is shown with their frequencies. Entries for Symptoms that did not occur for the specific FM1 are left blank. This results in a big matrix being shown, which can be scrolled horizontally or vertically. A small bar chart is displayed for the frequency of a Symptom in relation to the FM1. It gives a quick visual indication of which Symptoms are of significant importance to the FM1 and how they related to other FM1s. The list of Symptoms can be filtered with the text input field on the top left called ‘filter symptoms’. Furthermore, various ways of sorting can be configured like by Symptom, by individual FM1, by FM1 cluster, or by all FM1s.

6.2 FMkb integration

The initial goal was to have a visual extension to the FMkb software. For now, a standalone prototype was constructed that has two integration steps with FMkb completed. We first discuss the current steps of integration, next recommendations for a seamless and optimal integration are provided.

Current integration

One integration step is the addition of external links to FMkb for all FM1s and FM1 clusters, highlighted with a red border in Figure 34. The icon button next to an FM1 ID in FMvet takes the user to that FM1 in the FMkb FM1 view, where additional statistics can be viewed. On the right side, the external link of an FM1 cluster takes the user to the FMkb FM1 selection view, where all FM1s are selected. Unfortunately, there is no support for FM1 clusters in FMkb yet, thus the user needs to select particular FM1s to investigate further. The applied filters in FMvet such as the Date Range are automatically configured for FMkb as well. The other integration step is meant as a bridge from FMkb to FMvet. Filters can be retrieved from the URL parameters by the FMvet backend and will be adjusted accordingly in the interface. This means a button to FMvet can be added in the FMkb UI, which passes the set filters along with the URL.

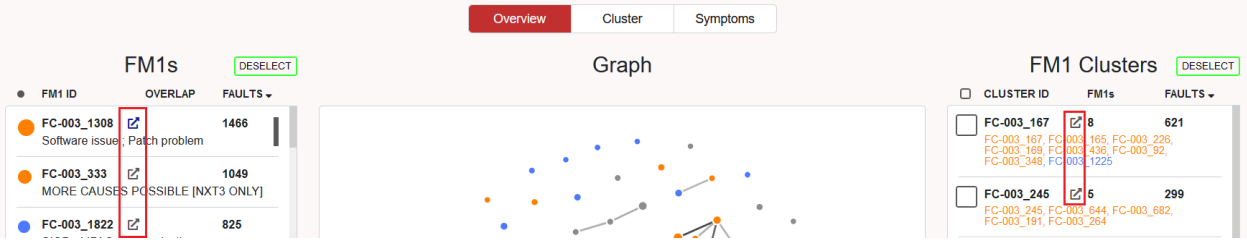

Figure 34: external links from FMvet to FMkb

Recommended integration

The optimal integration of FMvet with FMkb requires several extra steps to be completed. These could not be finished yet due to technical complexities and time constraints, but are recommended before the visual extension goes live. First, all FMkb filters should be integrated with FMvet. This entails that the FM1 overlap and FM1 symptoms API calls should be adjusted to support those filters. Once this step is completed, the FMkb filtering UI can be used and coupled. The FMvet interface will then be displayed in the FMkb tool itself, instead of as a standalone app. The visual extension is activated by clicking the 'Visualize FM1s' button in the FMkb interface, as can be seen in Figure 35. As a result, the view on the data can switch from Visualize FM1s, to Fault stats, and to Down stats.

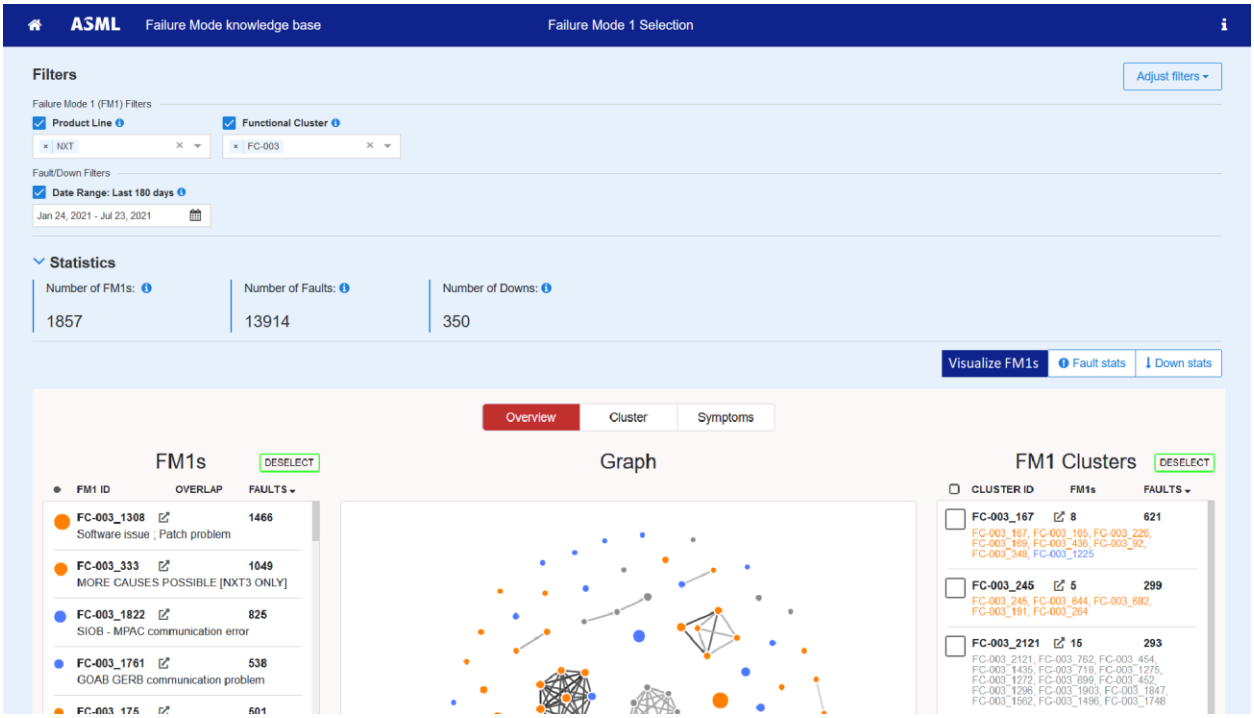

Figure 35: mockup of full FMvet integration within the FMkb interface

6.3 Reflection on the analysis

A review of the Analysis chapter was performed in relation to the final designed visualization. In this step, we evaluated whether the intended tasks can be completed with the current tool. Next to that, we check whether all envisioned functionalities and requirements are accounted for.

Tasks

Most user tasks can be completed in full. However, two are only partially supported by the tool. For T2 (inspect undiagnosable Faults and get hints on what FM1s to create) the main reason being that the focus slightly shifted away from showing individual Faults and towards the bigger picture of FM1s. For T10 (find FM1s to deprecate if they are not used or not correct anymore) the cause was of conceptual nature, since the FM1s that are not used anymore will not be part of the top 100 FM1s loaded in the tool.

Functionalities

All intended functionalities are supported by FMvet. The list of functionalities was followed as a blueprint, therefore they are all covered in the constructed tool by design.

Requirements

The requirements were incorporated by visualization design as well. However, technical limitations related to the large size of the FMkb dataset resulted in a undesired loading time for one data query. The neighborhood loading requires three sequential API calls due to its complex nature. Therefore, loading this data is performed in around 15 seconds, thus R3 (the tool should be fast; data loading can take five seconds at most and from there on the visual representations and animations should be fluent) could not be fully satisfied. In practice, the long loading time occurs infrequently and is therefore no real problem for the user. It only happens when working with advanced features such as the neighborhood filter, all other data loading calls are performed in five to eight seconds.

Chapter 7

Implementation

The aforementioned final visualization was realized as a prototype. The prototype is implemented as a webapp, making use of several web technologies. The FMvet prototype pulled data from the readily available FMkb API, which fulfills that same function for the FMkb tool. Adaptation of the FMkb API was needed to support requests for new types of data from the underlying database. Once the data was returned from the API, several preprocessing steps were applied to make the data ready for visualization purposes.

7.1 Software

The software stack consists of a variety of web technologies, each featuring their own strengths. The main programming language used for the functionality in FMvet is JavaScript. It provides interactivity with the UI, data requests, data preprocessing and the visualization dynamics. The user interface (UI) builds upon Bootstrap, which makes the layout follow a grid system and has many ready-to-use UI components. Custom HTML and CSS was used to glue the other parts of the UI together. Several custom made objects were introduced, like the multiselect dropdown, all data lists (FM1s, FM1 clusters, etc.) and the Symptoms matrix. The more advanced SCSS helped to structure the code for styling and provided advanced features like rule nesting, variables, improved selection, and browser prefixing. The SCSS code was compiled to standard CSS for it to be usable by an internet browser. Furthermore, icons were fetched from the Font Awesome library.

PHP is a server-side language which serves HTML pages to the user and makes API calls. It enabled to run the prototype on local machines and facilitate http requests to external sources like the FMkb API. Support for local machines was needed to access the tool from the participants' laptops in the domain expert review. For the final integration with FMkb, the PHP code will be transformed into Java code that runs on the established FMkb server.

The main part of the tool, visualization of the graph, is supported by the D3.JS framework. D3 is a well-known framework for building various types of visualization in the browser. The graph algorithms, selection predicates, data coupling with the UI and automatic scales were particularly useful. The force-directed graph algorithm was tweaked to become the powerhouse of the tool, as it provides dynamic positioning of nodes and links within the canvas.

7.1.1 Infrastructure

An infrastructure setup was designed with the support of developers from FMkb. The preferred method of requested data from the FMkb was to request the required data from the FMkb API, instead of directly querying the database. The final data exchange setup is depicted in Figure 36, where we see that data is requested from the API, which in turn queries the database. The database responds with the data, which is turned into JSON format by the API to be easier to handle by FMvet and other frontend applications.

Two new types of API calls needed to be added to the REST API, since support for some types of data was not integrated yet. First, a specification document was made for the call that returns FM1 overlap data. Second, the specification for symptom data belonging to a singular FM1 was constructed. Both specifications consist of the API endpoint, input parameters, expected output and an example SQL query. The documents can be found in Appendix D (Specification FM1 overlap query) and Appendix E (Specification symptoms query).

Due to in-place security measurements, authentication was handled via a proxy with the FMkb tool. The user was requested to login to the FMkb interface and to reuse the access token generated by that tool.

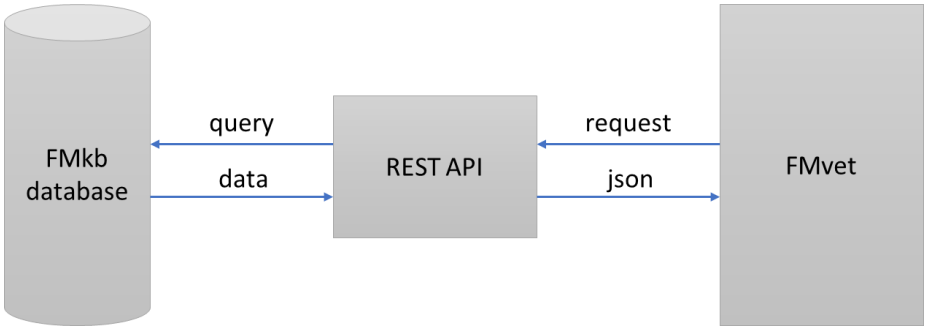

Figure 36: Data exchange setup via the REST API

7.1.2 Performance

Once the basic functionality was set up, there was room for performance improvements. For example, parts of data requests were cached and UI elements were only generated whenever they were requested by the user. Due to caching of the data, several calls to the API could be eliminated. This impacts the filters FM1 type, Overlap threshold, Singulars, and the Color modes, which now do not require data loading to display the adapted content. The biggest time saving comes from not loading the Symptoms data at first glance, but only when the Symptoms view is actually opened by the user. This prevents the document from creating the immense Symptom matrix when it is not needed, and therefore does not slow down the page. Furthermore, the extra information associated with FM1 clusters is loaded in the background. It results in a speed improvement for the list's time to display. Whenever the call retrieving the distinct faults belonging to an FM1 cluster is ready, the list gets updated with the new information and sorted when needed.

7.2 Data

The data is requested from the FMkb API, depending on the filters that are set in the FMvet UI. A translation of the UI filters to API parameters is performed in the background, such that the desired data can be retrieved. There are three main datasets used in the tool, namely the FM1 array (nodes in the graph), the FM1 overlap array (links in the graph) and the FM1 symptoms matrix. First, the FM1s are loaded. Next, the ids of these FM1s serve as input to the FM1 overlap query, as the overlap between those FM1s is what we are interested in. Later on, the FM1 symptoms can be requested on demand for each individual FM1. The three API calls to access the database are:

FM1s (/api/v1/fm1s) | output: array of FM1s

Input parameters: PL, FC(s), SU(s), Date range, Maximum number of FM1s to return

Output properties per FM1: ID, Description, Type, Number of Faults, FC, SU

FM1 overlap (/api/v1/fm1s/overlap) | output: array of FM1 overlap

Input parameters: PL, Set of FM1 IDs, Date range, Minimum required overlap, Include neighborhood

Output properties per FM1 overlap: Source FM1, Target FM1, Number of overlapping Faults

Symptoms (/api/v1/fm1s/symptom_faults) | output: array of Symptoms

Input parameters: PL, FM1 ID, Date range, Minimum required Faults, Filtered pattern

Output properties per Symptom: ID, Number of Faults

Furthermore, extra properties are requested for the FM1 clusters, such as the number of distinct Faults that belong to it. Other properties could be added to this list in the future, like the number of affected machines, the average rule coverage and the total duration. The FMkb team could consider adding support for an FM1 cluster API call, where interesting properties are efficiently retrieved from the database and returned in one response.

The neighborhood filter requires extra API calls to function properly. Under the hood, one API call is made to retrieve the FM1s belonging to the filtered set; next one API call is made to retrieve the inner overlap between this set and the single hop neighbors of the filtered set. Finally, one more request to the FM1s endpoint is made to retrieve the FM1s that were not in the initial filtered set. This path is more complex than normal queries, and therefore results in a total response time of around 10 seconds for standard use cases.

7.2.1 Preprocessing

After the data is loaded from the API, several step of preprocessing are applied. These steps further filter the data and remove noise; enhance the data by adding links and aggregations; or compute additional information like the FM1 clusters, FC aggregations, and SU aggregations. A list of the chronologic steps is displayed below.

1. Remove links with non-existent source or target and links with too little overlap (according to the overlap threshold value)
2. Remove nodes with excluded FM1 types
3. Remove links with non-existent sources or targets
4. Remove nodes without outgoing links (only when singulars are filtered out)
5. Add bi-directional links where there is only a single connection (A->B OR B->A becomes A->B AND B->A)
6. Compute FM1 clusters (by breadth-first search)
7. Compute FC aggregations (by looping over FM1s)
8. Compute SU aggregations (by looping over FM1s)

After the preprocessing and aggregation steps are completed, the data is cached and linked directly to the UI. The data properties are inserted into the data lists in the UI, which can later be retrieved via the cache or via the HTML DOM element. When all steps are completed, the user can access the functionality and data that the tool provides.

Chapter 8

Domain Expert Review

To evaluate the designed solution, the implemented prototype was put to the test with a domain expert review. Several users from different disciplines and with various background knowledge were given early access to the tool with production data available. In total seven experts were selected to participate in the trial period, four from Development & Engineering (D&E) and three from Customer Support (CS). The participants were asked to give informed consent to handling and analyzing their data anonymously in this study.

The goal of the user evaluation is to get both qualitative and quantitative insights on how well the tool performs and whether it brings value to the users' day-to-day work. Other investigative aims are to find out what the advantages and limitations of the prototype are. The results are divided into functionality, usability and usefulness aspects. We look into what other features were expected, whether the users are able to gain better views of the underlying data and whether the visualizations are understandable after a certain period of usage.

8.1 Experiment setup

The setup of the user evaluation was strictly defined in advance, to obtain as much objective insights as possible from the domain experts. The participants were to be guided through a trajectory of three steps, namely the setup session, a week of usage and the review session.

The first step was to get the users up and running with FMvet, both functionally and conceptually. Three setup sessions were planned from which the user could select the slot that best suited their calendar. The setup session consisted of an explanation of the domain expert review setup, a demonstration of the prototype and instructions on what access roles are needed and how to set up the prototype locally. The choice was made to have a demo and thorough explanation of the functionality of the tool before the users started their week of usage. The reason here is that the real-world users are going to get a brief training and use the tool over extended periods of time as well. The demonstration was identical to the video which is mentioned in Section 6.1, describing the final visualization. As the local setup was proven to be rather challenging, the instructions were send to the user as a PDF document for later reference.

The second step was for the user to explore FMvet on their own. Instructions were given to take sufficient time for the first time setup (around 30 minutes), both to get the technical setup done and to familiarize with the functionality and usability of the prototype. One to one-and-a-half week was scheduled for this part, to give the user ample opportunities to further explore the tool and encounter work tasks that might benefit from using the tool. During the week of usage the users were not contacted, but advised to open the tool whenever they deemed it useful for their day-to-day work. When technical or conceptual issues were stumbled upon, the user could ask for help and guidance on how to continue.

The third step is the individual or pair-wise review session with each participant. The review sessions were scheduled to take 45 minutes, in which the researcher guided the user through an open interview, predefined tasks and a questionnaire. Each part is explained in more detail below.

8.1.1 Interview

The open interview was mainly intended to gather qualitative information from the experts, such as feedback in the form of criticism, positives and new ideas. The intention here is to get the overall gist of and feelings towards the new tool. The main point of interest is whether the tool fits in the user's current workflow and brings new and beneficial insights that were not available before. The guiding questions displayed below are used to start the discussion with the expert. From this starting point, the researcher will guide the interview by asking follow-up questions and investigate what opinions the user has about the prototype and its functioning.

Guiding questions

- What was your general experience with the tool?
- Can you give an indication on how often the tool was used and for what activities?
- Did you find any surprising insights? If so, which ones?
- How did it fit in your daily workflow? Is anything missing?
- What parts did you like or dislike?
- What parts were easy or hard to understand?

8.1.2 Tasks

A set of tasks was defined by making several tasks from Chapter 3 (Analysis) actionable, which are listed below. The user performed these during the second part of the review session. The tasks are intended to investigate the actual understanding of and capabilities with the tool. Looking from the user's first perspective view, a better understanding is formed of what the encountered difficulties and limitations are. The researcher monitored their behavior and asked the user to talk through the thought process while performing a task. To be able to quantify and systematically investigate the performance for each task, a set of metrics is measured. The metrics are influenced by the work of Lecerof and Paternò (1998), who looked into methods for effective usability evaluation through performing tasks. The metrics are success rate, percentage of users who correctly completed the task; time to completion, average time it took to complete the task; and error rate, percentage of users who encountered the same problem.

Usability tasks

T1 Identify one diagnostic gap in FC-003 (FC)

T2 Find FC-003_165 (FM1), what cluster does it belong to?

T3 For FC-003_167 (FM1), what FM1 is most similar to it?

T4 For FC-003_167 (FM1), what two symptoms are most differentiable from the cluster?

T5 For FC-003 (FC), investigate to what FCs it is closely related

8.1.3 Questionnaire

To finish off the domain expert review, a user experience questionnaire was set up. The questionnaire is adapted from the well-known standard USE questionnaire (Lund, 2001) that helps studies find quantitative insights into the application's performance. A selection of questions was made to better fit the visualization application that is under investigation here. In total we have 15 statements divided over four categories, namely usefulness, ease of use, ease of learning, and satisfaction. The particular statements for each category can be found below. For the answers a 7-point Likert scale was used, with the 'not applicable' option available as well. Therefore the potential answers are 1) strongly disagree, 2) disagree, 3) somewhat disagree, 4) neutral, 5) somewhat agree, 6) agree, 7) strongly agree and 8) not applicable.

Usefulness

- It helps me to be more effective
- It makes the things I want to accomplish easier to get done
- It is useful
- It meets my needs

Ease of use

- It is easy to use
- It is flexible
- It requires the fewest steps possible to accomplish what I want to do with it
- I can use it successfully every time

Ease of learning

- I learned to use it quickly
- I easily remember how to use it
- I quickly became skillful with it

Satisfaction

- I am satisfied with it
- It is fun to use
- It works the way I want it to work
- I feel I need to have it

8.2 Results

Many different insights were gathered from the participants, they are categorized as results from each part of the review session. First, we discuss the observations made during the entire session. In the interview subsection we discuss the general reception of the tool and list quotes of users. In the task subsection the measured metrics belonging to each task are presented with the help of charts. Finally, in the questionnaire subsection we show the results for the statements.

8.2.1 Observations

During the interview and execution of tasks, observations were made on what parts of the tool have lacking functionality or usability. A list of the major and minor concerns was compiled for both, which can be used as a blueprint for future improvements to the prototype. The major points for improvement are listed below, whereas the minors can be found in Appendix A.

Functionality

- Add support for Downs, it could be a switch between Faults and Downs in the UI. From the Downs viewpoint, the size of nodes in the graph can be related to the amount of downtime.
- Support a reverse symptom query, it finds all FM1s that trigger with the selected symptom (this is already possible with FMkb)
- Add a uniqueness metric to each symptom for the selected FM1, it shows whether the particular symptom representative for the FM1. This metric can support the user to get a better overview of which symptoms are actually useful and to filter out noise.
- Related to the previous point, the uniqueness metric can be used to automatically recommend expert rules for an FM1. It could compute and show potential symptoms that are significantly differentiating from the entire dataset.
- Add FM1 cluster notion to FMkb. It aggregates data for FM1s and shows the resulting statistics, like what can be done for individual FM1s now.

Usability

- Overlap between FM1s in the graph for an FM1 cluster can be more clear. What is meant with inner darker color and outer brighter color is not intuitive. One suggested solution would be to let the inner part shrink and make the outer border very thin, without changing colors.
- Move the horizontal scrollbar to the bottom of the Symptoms view. It makes more sense conceptually and is in line with where it is located in other tools.
- Change the FM1 cluster ID to something unrelated to the underlying data. It could be a letter following the alphabet, instead of the FM1 ID of biggest FM1 in cluster.
- A search box should be added to each data list (FM1s, FCs, etc.) that filters the IDs.
- Unable to investigate the FM1 type and FC colors at the same time, which could be desired.
- Create an elaborate user manual. It could be consulted for the more advanced features and prevent extensive training from being required. It should also explain the new concepts.
- Export to Excel feature (e.g. export FM1 cluster with symptoms). It allows for more functional flexibility and personalized presentation of the data.

8.2.2 Interview

In the interview, the users were overwhelmingly positive about their experience with FMvet. They were able to understand the data and visual language, and could benefit from this type of visualization in their daily work. The participants indicated that they were able to achieve their intended goals with the tool, such as getting a better understanding of the underlying data and database structure, finding diagnostic gaps and setting priorities on them, and to find pointers on how to improve diagnostic quality. Some users also indicated there are limitations and there is room for improvement. The most requested extra functionalities were support for Downs (next to Faults), additional filtering options, and a more powerful Symptoms view. Furthermore, three representative quotes from the interviews were put together below.

"I always missed this type of visualization, because only working with lists is hard. It is a lot easier to explore data graphically, it provides more insights and makes the way of working more fluent." (D&E)

"The tool empowers me to guide the team belonging to an FC to parts (of the diagnostics) that need improvement and to set their priorities. The tool could be used before a PI event to define goals for the upcoming weeks, and afterwards to check the impact made." (D&E)

"The tool is useful as a starting point, as it helps me to better understand the underlying data and gives pointers to further investigation. However, the symptoms view is not yet powerful enough for my work, certain additions are needed to make it a valuable part of my toolset." (CS)

8.2.3 Tasks

Three metrics were measured while the user was performing a given task, namely the success rate, completion time and error rates. Each metric is discussed in more detail with its associated data. Note: two participants worked together on the tasks. Therefore, there were a total of six participants for this part of the review.

Success rate and completion time

Success rate and completion time are shown with the charts in Figure 37. The success rate is represented as the percentage of participants which were able to successfully complete the task. The completion time is shown as a line indicating the minimum and maximum time in seconds that were needed to complete the task. Furthermore, the mean is indicated with a blue dot and the perfect time with an orange dot.

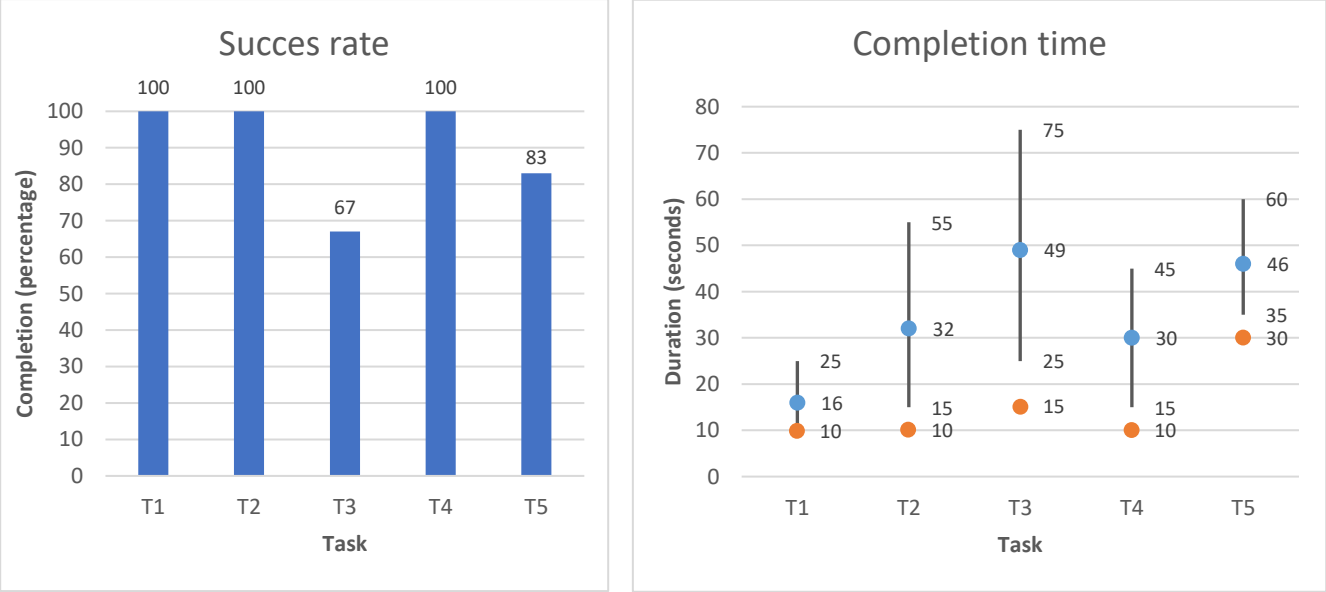

Figure 37: success rate and completion time of the tasks

Error rates

The observed errors and difficulties were written down during the participants' execution of each task. They are written down below. The number of participants that encountered the same error is listed between parenthesis.

T1 No substantial errors

T2 Tried to use FM1 ID filter (which is used for filtering data, not for searching FM1s) (3),
Cluster ID is confusing and misleads users (2)

T3 Difference between absolute and relative overlap is unclear (4),
Confusion between inner and outer circle (which visualize overlap in the graph) (2),
Thought position in graph was related to overlap and thus similarity (1)

T4 Hard to find horizontal scrollbar in Symptoms view (5)

T5 Switched the sorting on the FC list to FM1s (instead of Faults) (5),
Hard to locate the neighborhood filter in the UI (1),
Hard to locate the FC list in the UI (1)

8.2.4 Questionnaire

The questionnaire resulted in quantitative data for each of the statements. The results are displayed as bar charts in Figure 38, 39, 40, and 41. The bars are left-centered and red-colored if the answers are negative, or right-centered and blue-colored when they are positive.

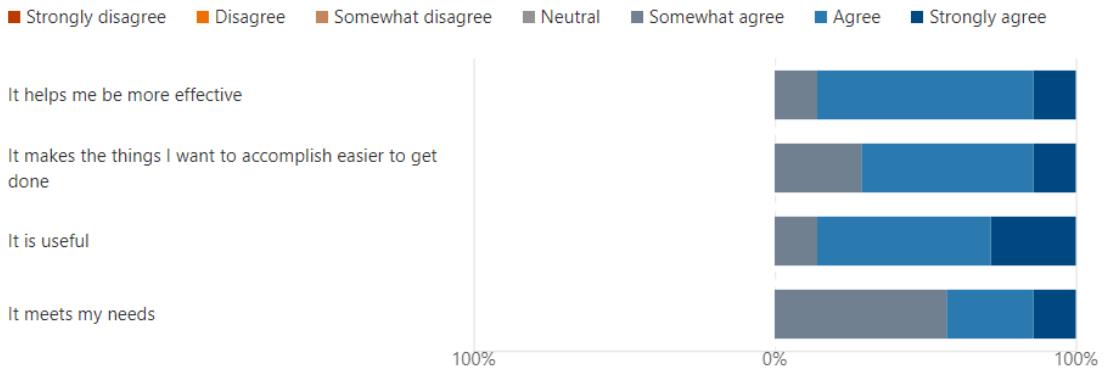

Figure 38: questionnaire results on usefulness

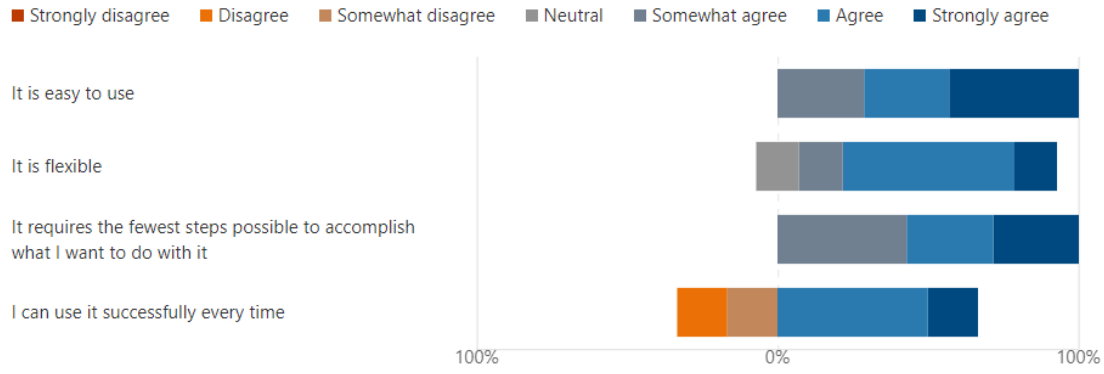

Figure 39: questionnaire results on ease of use

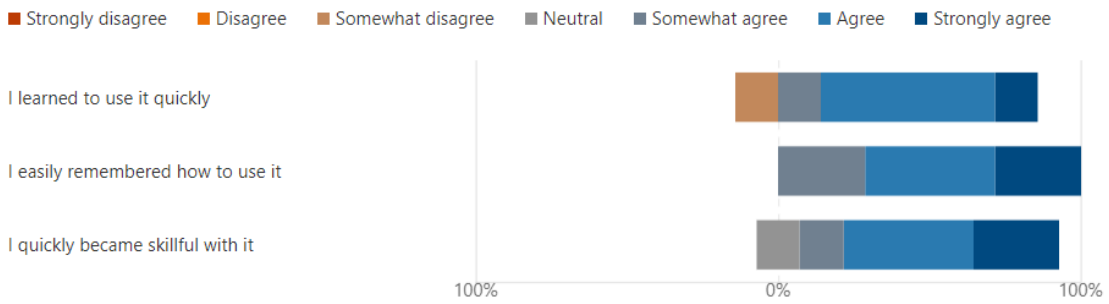

Figure 40: questionnaire results on ease of learning

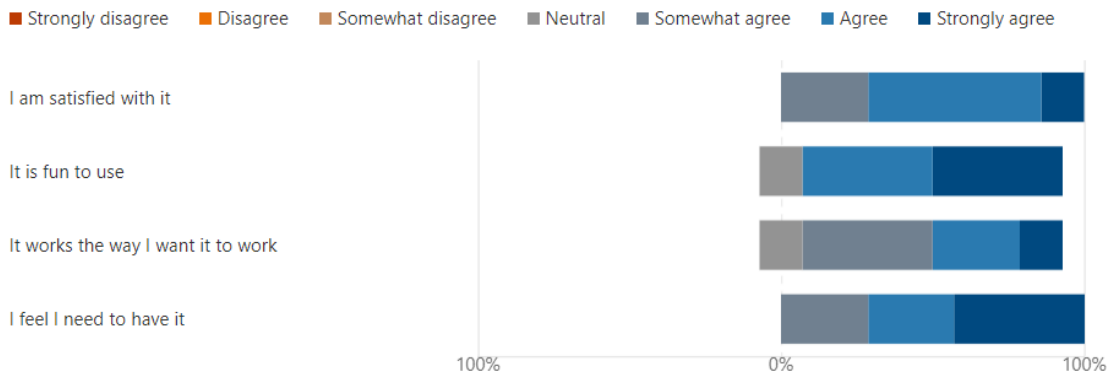

Figure 41: questionnaire results on satisfaction

8.3 Interpretation of the results

We consider the results from the observations, interview, tasks and questionnaire, to evaluate whether the tool was designed appropriately, and what parts need a second iteration.

8.3.1 Interpretations

Observations

During observation of the users, more information about the usage was gained and troubles with the tool were gathered. Generally, the observations showed that the users were able to correctly perform all intended basic tasks. There were also some unexpected results that were not encountered in smaller user tests before. The difference in skill levels and time spent with the tool during the week of usage, were influencing the user's capabilities with the prototype. The extensive list of both functionality and usability issues shows what parts need more effort. Several newly introduced conceptual definitions were hard to grasp, a user manual and better training could help with understanding them. The manual should contain analogies to everyday concepts and explanations of the visual language of FMvet.

Interview

The gist of the interviews was that the users were pleasantly surprised by the capabilities of the current prototype. They indicated that their intended goals were achievable and made easier. On another note, the users indicated that the tool was fun to use and motivated them to look at the underlying data more

closely. So far, only limited unexpected insights from the data were found by the users. To make the tool even more useful in the future, more iterations are needed to tailor to specific use cases. The main part that should be more powerful is the Symptoms view, with the addition of reverse queries and rule recommender features. Next to that, a way to focus on Downs would be especially helpful to Customer Support.

Tasks

Most tasks were completed by all users, with an average success rate of 90 percent. The completion time varied widely between the participants, indicating that some users had become more skillful than others in the limited time available for testing. The mean completion time was on par with our self-estimated effort and timing, and is quicker than what would be possible without FMvet. The error rate shows several errors surfacing that were not envisioned when the tool was designed. Luckily, the common errors can be updated in future versions of the tool, ensuring that all basics tasks are completable by future users.

Questionnaire

The qualitative questionnaire data shows overwhelmingly positive reactions to each statement. The usefulness and satisfaction statements received especially high scores. That indicated the users are willing to use the tool in the future and it was beneficial to their work. Only the statements ‘I can use it successfully every time’ and ‘I learned to use it quickly’ got slight hints of disagreement. This can be traced back to the setup being relatively hard for non tech-savvy participants, no user manual being present and the demonstration being a lot to process in one session.

8.3.2 Future features

The participants’ troubles and observations during the review sessions were translated to potential future features. Next generations of the prototype could incorporate these features to reach more goals and improve the tool’s functionality and usability. The improvements were partly known already, but could not all be incorporated due to time constraints. When given extra time, a more polished and easier to use prototype can be created. The list of the proposed future features can be found in Appendix B.

Chapter 9

Discussion

In this chapter we reflect upon the process, final visualization and domain expert review. Improvements were made to some parts of the diagnostic process, but certain prototype limitations hindered the tool to be used to its full extent. Next to that, pointers to future work are presented.

9.1 Improvements and limitations

FMvet brings functional improvements to the diagnostic process and to the workflows of its target groups. The FMkb interface had successful results by adding simple visualization aspects, such as line charts and bar charts. In our work, visualization takes a more central role and the details (data lists) come later, following Shneidermann's mantra. The users indicated this visual approach was useful, easy to use and motivating. The tool provides new views on the data, which makes it easier to find and share interesting insights for all types of users and high-level tasks. In the domain expert review we found that the intended goals were achievable with support of the tool. Furthermore, new conceptual definitions were developed and defined during this work. They can become integral parts of the diagnostics process and be used in other tools, increasing the generation of new insights and promoting consistency between tools.

In the beginning, exploring the data through database queries and simple visualizations helped shape the concepts and to discover interesting parts of the data to look into. The iterative design process contributed to having a mix of curiosity, focus and progress throughout the project. Having a clear initial focus with only limited facets worked well. Later on the focus was broadened by new iterations, adding extra functionality with each step. This way of working helped keeping the base functionality intact and made sure all use cases were incorporated. Overall, the project provides a successful application of visualization on real world data.

Furthermore, there is also room for improvement of FMvet. Improvements can be made on the conceptual level, as well as on the current implementation. Conceptual definitions could have been more clear in the tool, with the help of visualization and explanation. Several iterations went over minor parts of the visual language, but they did not always have the correct outcome. For example, we decided the overlap data for an FM1 cluster would be best shown with darker inner circles. This was proven to be a mistake by the participants' confusion during the task executions, thus plans for a better visual representation were included. On another note, there were implementation issues which hindered the prototype from being easily available to more users. It was an intended decision made in the beginning to allow for easier development. A deeper dive into the final setup earlier on could have prevented this from happening, but was unlikely to be spotted on time. Unfortunately there were external factors which limited the progress and caused time delays. The FMkb team had technical issues for several weeks, causing the API call for Symptoms to be implemented behind schedule and not in time for the user evaluation. The issue was partly overcome by preloading a chunk of Symptom data and allowing the user to manually request more Symptom data whenever needed.

Real integration with FMkb was considered out of scope from the start, to isolate this study from FMkb's work. However, limited integration was helpful to demonstrate and try the tool with live FMkb data. We ran into practical complications, such as a server technology mismatch and the manual entry of user credentials. These interim workarounds meant that users evaluating the tool experienced a tricky first-time setup and authentication. Next to that, the tool was not 100 percent polished and can use quality of life improvements. The combination of these things may have limited the user's ability to use the envisioned tool to its full extent. Furthermore, the prototype demonstration was quite information-dense, no extensive user manual was available, and more configuration settings could be added. The domain expert review was performed in a structured manner and led to valuable insights for next generation improvements to the tool. However, the sample size and test duration were relatively small and short. The participants did represent the major target groups well, limiting the impact of those shortcomings. More information could have been gathered by more extensive testing, but the most important deficiencies were caught in this user evaluation.

9.2 Future work

Future features were discussed in the previous chapter, but we did not cover the bigger structural and conceptual improvements yet. A list of the most relevant improvements is given below.

- The full integration with FMkb has the highest priority on this list. It allows all FMkb users to automatically consult FMvet, without needing extra installation or permission procedures. In Section 6.2 we discussed what the best approach for full integration is. The integration should be relatively easy and quick for an FMkb developer, as the general structure is in line with FMkb's software.
- The implementation of more data filters will allow the user to better specify which data they are interested in. Especially the Fabs filter and FM1 IDs filter should be prioritized, as they were requested by CS colleagues in interviews. It would support the top-down approach, were the user starts investigating data with the starting point of one Fab or one FM1 only. Synchronizing all data filters between FMkb and FMvet also result in higher consistency between the two, thus improving ease of use and ease of learning for FMkb users. This integration step requires changes to the FMkb API, thus it was postponed till developer time is available.
- The addition of a new layer to the conceptual design should be considered. This layer focuses on the relation between the FCs or between the SUs. By aggregating the overlap data between all FM1s of two FCs, the relation between those two FCs can be analyzed. This is particularly useful for finding unusual connections and for checking what is happening there. The visualization of this problem should look like a rough mapping of the machine, which can be compared to the design blueprint. It might be helpful to graphically display the FCs onto an image of the machine, as it helps understand what parts of the machines are related. This type of visualization can also be used by inexperienced people at ASML to get a better understanding of the complex machinery.
- The domain expert review and talks with Customer Support colleagues usually had a particular focus on the Symptom data belonging to FM1s. A new iteration could give this part of the tool a more central role. First, it would be very helpful to indicate the currently active expert rules for each FM1. Second, new features could be included such as: support for automatic recommendation of expert rules; reverse queries from Symptom to FM1s;

and uniqueness values for the Symptoms of an FM1. As this is a very complex task, it might be considered to create a standalone rule recommender tool.

- As was seen in this work, visualization can be a powerful tool to get more insights from the available data. It enables new views on the data, makes analytical work more satisfying, and helps to set priorities on the tasks at hand. Other types of ASML's data can benefit from effective visualizations and allow for better analysis of the machine's performance. Our advice is to look at which data is most suitable for visualization and follow a similar iterative design process for those use cases.

Chapter 10

Conclusion

In this work, the Failure Mode visual exploration tool (FMvet) was presented as a viewing tool of diagnostic data. More specifically, FMvet supports the diagnostic experts with diagnostic data creation, quality control during data release, and quality control through data cleanup. New concept definitions were introduced to grasp conceptual properties of the diagnostic data, such as FM1 overlap, FM1 clusters and FM1 neighborhood. A prototype was constructed to try out the final designed visualization, using real data from the FMkb API for data loading. New views on the data are presented through the tool, such as an overview of the FM1s in an FC, specific FM1 clusters consisting of overlapping FM1s, and organized Symptom data belonging to each FM1. To investigate whether this tool is successfully employed in the daily workflow of users, a domain expert review was performed. The experts indicated that the visual aspects helped to get a better understanding of the underlying diagnostic data and made their usual workflow simpler and more pleasant. Therefore, we hope FMvet can assist in improving the diagnostic quality (one fitting solution for one failure) and diagnostic hit rate (all failures should have a solution) in the future. In the end, we touched upon how FMvet supports the diagnostic process, but also discussed the current limitations of the visualization tool. Several pointers to future work are included, and we recommend the use of visualization as a tool that can benefit ASML in the future.

References

1. Alsallakh, B., Aigner, W., Miksch, S., & Hauser, H. (2013). Radial sets: Interactive visual analysis of large overlapping sets. IEEE transactions on visualization and computer graphics, 19(12), 2496–2505.
2. Alsallakh, B., Micallef, L., Aigner, W., Hauser, H., Miksch, S., & Rodgers, P. (2014). Visualizing sets and set-typed data: State-of-the-art and future challenges.
3. Alsallakh, B., Micallef, L., Aigner, W., Hauser, H., Miksch, S., & Rodgers, P. (2016, February). The state-of-the-art of set visualization. In Computer Graphics Forum (Vol. 35, No. 1, pp. 234-260).
4. Behrisch, M., Bach, B., Henry Riche, N., Schreck, T., & Fekete, J. D. (2016, June). Matrix reordering methods for table and network visualization. In Computer Graphics Forum (Vol. 35, No. 3, pp. 693-716).
5. Bollobás, B. (2013). Modern graph theory (Vol. 184). Springer Science & Business Media.
6. Borgo, R., Kehrer, J., Chung, D. H., Maguire, E., Laramee, R. S., Hauser, H., ... & Chen, M. (2013, May). Glyph-based Visualization: Foundations, Design Guidelines, Techniques and Applications. In Eurographics (STARs) (pp. 39-63).
7. Burch, M., Vehlow, C., Konevtsova, N., & Weiskopf, D. (2011, September). Evaluating partially drawn links for directed graph edges. In International Symposium on Graph Drawing (pp. 226-237). Springer, Berlin, Heidelberg.
8. Collins, C., Penn, G., & Carpendale, S. (2009). Bubble sets: Revealing set relations with isocontours over existing visualizations. IEEE transactions on visualization and computer graphics, 15(6) 1009–1016.
9. Craft, B., & Cairns, P. (2005, July). Beyond guidelines: what can we learn from the visual information seeking mantra?. In Ninth International Conference on Information Visualisation (IV'05) (pp. 110-118). IEEE.
10. Fuchs, J., Isenberg, P., Bezerianos, A., & Keim, D. (2016). A systematic review of experimental studies on data glyphs. IEEE transactions on visualization and computer graphics, 23(7), 1863-1879.
11. Ghoniem, M., Fekete, J. D., & Castagliola, P. (2004, October). A comparison of the readability of graphs using node-link and matrix-based representations. In IEEE symposium on information visualization (pp. 17-24). IEEE.
12. Heer, J., & Boyd, D. (2005, October). Vizster: Visualizing online social networks. In IEEE Symposium on Information Visualization, 2005. INFOVIS 2005. (pp. 32-39). IEEE.
13. Herman, I., Melançon, G., & Marshall, M. S. (2000). Graph visualization and navigation in information visualization: A survey. IEEE Transactions on visualization and computer graphics, 6(1), 24-43.
14. Lecerof, A., & Paternò, F. (1998). Automatic support for usability evaluation. IEEE transactions on software engineering, 24(10), 863-888.
15. Li, J., Martens, J. B., & van Wijk, J. J. (2010, April). A model of symbol size discrimination in scatterplots. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 2553-2562).

16. Lund, A. M. (2001). Measuring usability with the use questionnaire12. Usability interface, 8(2), 3-6.
17. Maguire, E., Rocca-Serra, P., Sansone, S. A., Davies, J., & Chen, M. (2012). Taxonomy-based glyph design—with a case study on visualizing workflows of biological experiments. IEEE Transactions on Visualization and Computer Graphics, 18(12), 2603-2612.
18. Podtelezhnikov, A. A. (2008). Proteins and DNA: Venn diagram of amino acid properties. https://sites.google.com/site/apodtele/aa_venn_diagram.jpg.
19. Pretorius, A. J. (2008). Visualization of state transition graphs. Technische Universiteit Eindhoven.
20. Roberts, J. C. (2007, July). State of the art: Coordinated & multiple views in exploratory visualization. In Fifth international conference on coordinated and multiple views in exploratory visualization (CMV 2007) (pp. 61-71). IEEE.
21. Shneiderman, B. (2003). The eyes have it: A task by data type taxonomy for information visualizations. In The craft of information visualization (pp. 364-371). Morgan Kaufmann.
22. Van Der Wijst, M. G., de Vries, D. H., Brugge, H., Westra, H. J., & Franke, L. (2018). An integrative approach for building personalized gene regulatory networks for precision medicine. Genome medicine, 10(1), 1-15.
23. Van Wijk, J. J., & Nuij, W. A. (2003, October). Smooth and efficient zooming and panning. In IEEE Symposium on Information Visualization 2003 (IEEE Cat. No. 03TH8714) (pp. 15-23). IEEE.

Abbreviations

An alphabetic list of all abbreviations used in this thesis.

CS	Customer Support (department)
D&E	Development & Engineering (department)
DDF	Deterministic Diagnostic Flow (advanced diagnostic solution)
EV	Event
FA	Fault
FC	Functional Cluster
FM1	Failure Mode 1
FMkb	Failure Mode knowledge base
FMvet	Failure Mode visual exploration tool
MTTD	Mean Time To Diagnose
MTTR	Mean Time To Repair
PCS	Problem Cause Solution (old diagnostic solution)
PL	Product Line
SDT	Smart Diagnostics Tool
SU	Subsystem
SY	Symptom
Vis	Visualization

Appendix

Additional documents were used in the process of creating FMvet, but they did not fit in the main body. A table of contents for the appendices is displayed below, followed by the documents afterwards.

A List of minor potential improvements

The list of minor potential improvements following from the domain expert review is given below. They are categorized as functionality or usability improvements.

Functionality

- Visualize the diagnostic quality metric (including the user feedback loop).
- Apply a symptom threshold for filtering out noise (by setting a lower and upper bound).
- Show the expert rules belonging to FM1s in the symptoms view.
- Update the FCs and SUs lists based on selections made in the graph and other lists.
- Aggregate and visualize the Fault types for FM1s and FM1 clusters.
- Indicate which FM1s were (intended to be) replaced by other FM1s.

Usability

- Highlight selected FM1 in FM1 cluster list of ids in the cluster.
- Implement the Fabs filter, it provides individual Fabs' insights and is in line with CS's way of work.
- Add a button to PCS (Coach database) and DDF (DDF editor) in the FM1s list when applicable.
- Make the FM1 type color legend clickable (for fading out or highlighting certain types).
- Add a filter on the down duration (only include Faults with a certain duration).
- Create a way to (temporarily) exclude certain FM1s from the dataset.
- The indicator of FM1 selection and FM1 cluster selection in the graph can be represented in a better way. For example, the FM1 selection as a box around the node (instead of a circular border that can cause confusion with the inner circle), and the FM1 cluster selection as a glowing gradient in the background.
- Show FC descriptions as popups in the FC list.
- Provide a way to select an FM1 cluster from the graph.
- Deselect and select all buttons should only be visible when they cause changes (e.g. when a FM1 or FM1 cluster is selected).
- The zoom reset icon is not intuitive and should therefore be improved. It looks too much like an undo or rotate canvas button.

B List of potential future features

The list of potential future features is displayed below. Each feature is categorized under the heading that best descriptions the type of improvement and part of the tool it belongs to.

General

- Change theme color to match FMkb and to not be red (can be viewed as negative).
- Add more (FM1 cluster) meta data (e.g. affected machines, impact, etc.).

Filters

- Support custom date range filter.
- Support search by ID in all data lists.
- Implement the FM1 entry field (enter single or multiple FM1s to include).
- Implement the Fabs filter.
- Color Mode selection should be a three-state switch instead of multiple checkboxes.
- The differentiation between data filters and visualization settings should be more apparent.

FM1s list

- Show overlap with the selected FM1 for all FM1s in the list (in black when it is higher than overlap threshold, in grey when it is lower).
- Show the overlap column header only when an FM1 is selected.
- Fix node size in the FM1 list when zoom level is non-standard.
- Make two separate lists for the primary (filtered) and secondary (neighborhood) FM1s

FM1 neighborhood

- Primary and secondary colors for, respectively, selected and neighboring FM1s.
- Create a separate FM1s list for neighboring FM1s.

Graph

- Only update parts of the visualization which have been added, removed or changed, instead of redrawing the entire visualization each time the data has been changed.
- Make nodes in the graph draggable to other positions, for when the initially calculated location is clashing with other nodes.
- Show link color gradient legend

Global settings

- Make the settings persistent through sessions via a browser cookie.
- Add a setting for the gamma increasement of the node size and edge color intensity.

Preprocessing

- Remove FM1s with 0 Faults.

C Figures of the final visualization

This appendix features a selection of figures showing all aspects of the final visualization. Different selection, filters, and settings are applied to show potential usage states.

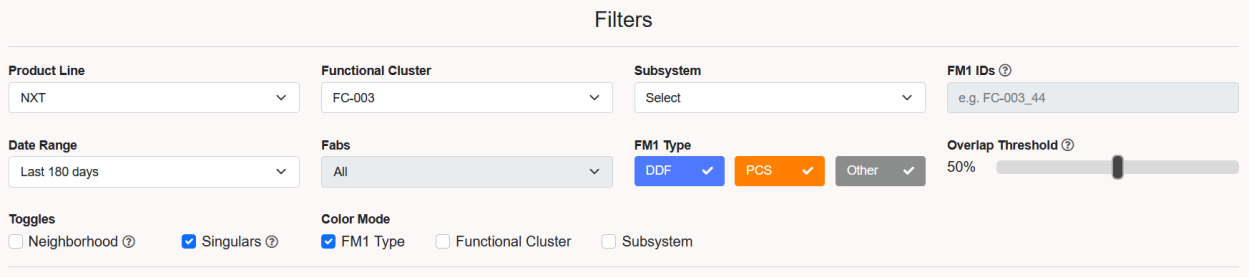

Figure 42: default filters section

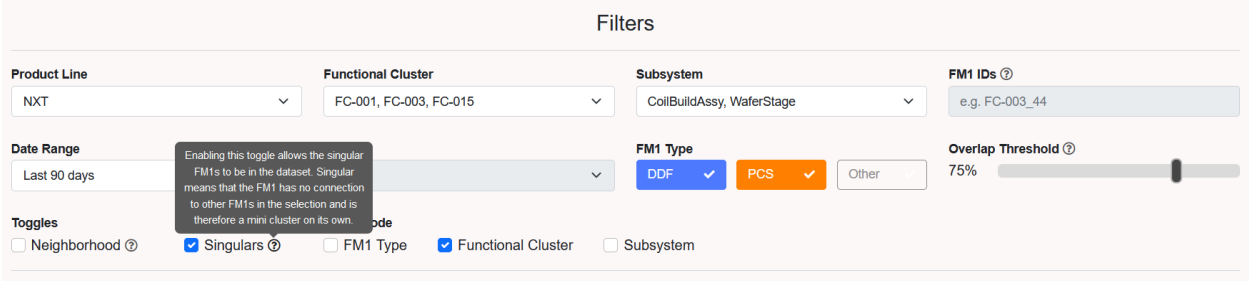

Figure 43: filters section in different state

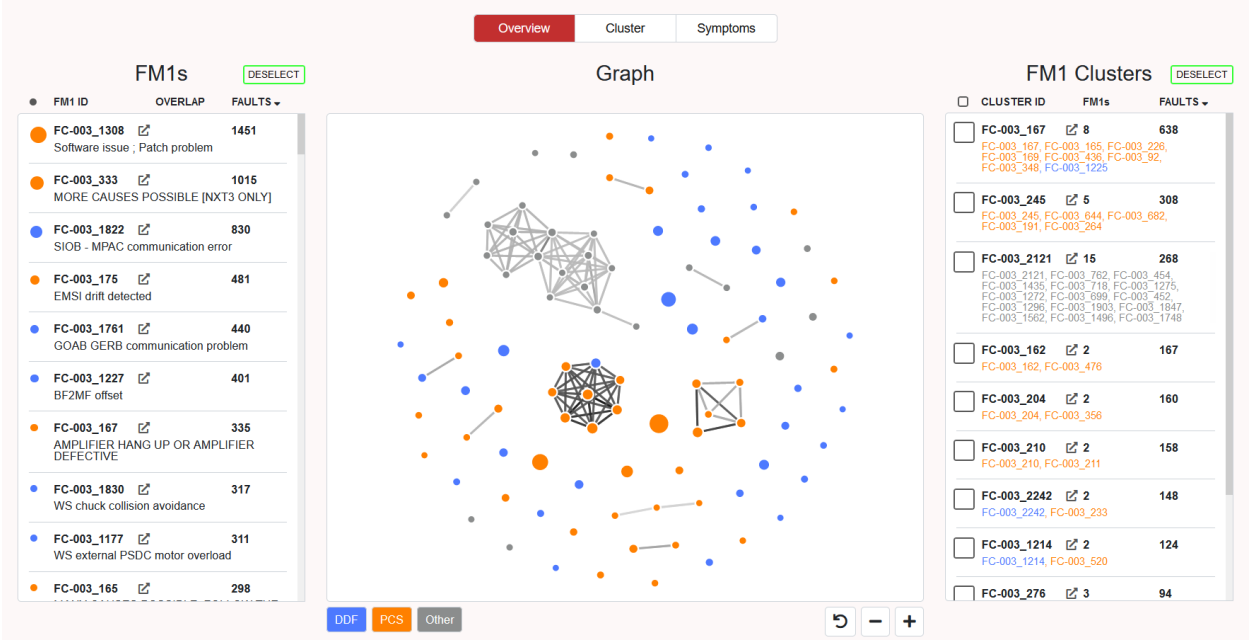

Figure 44: overview view default state

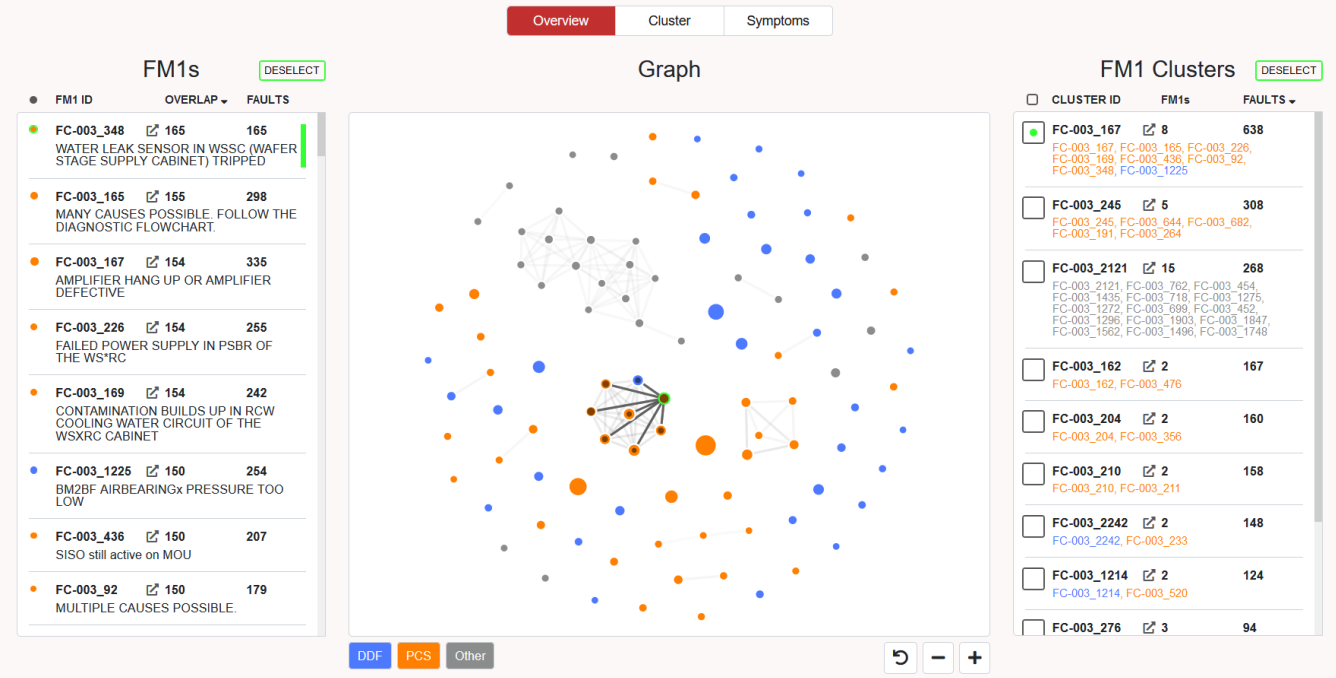

Figure 45: overview view with an FM1 selected

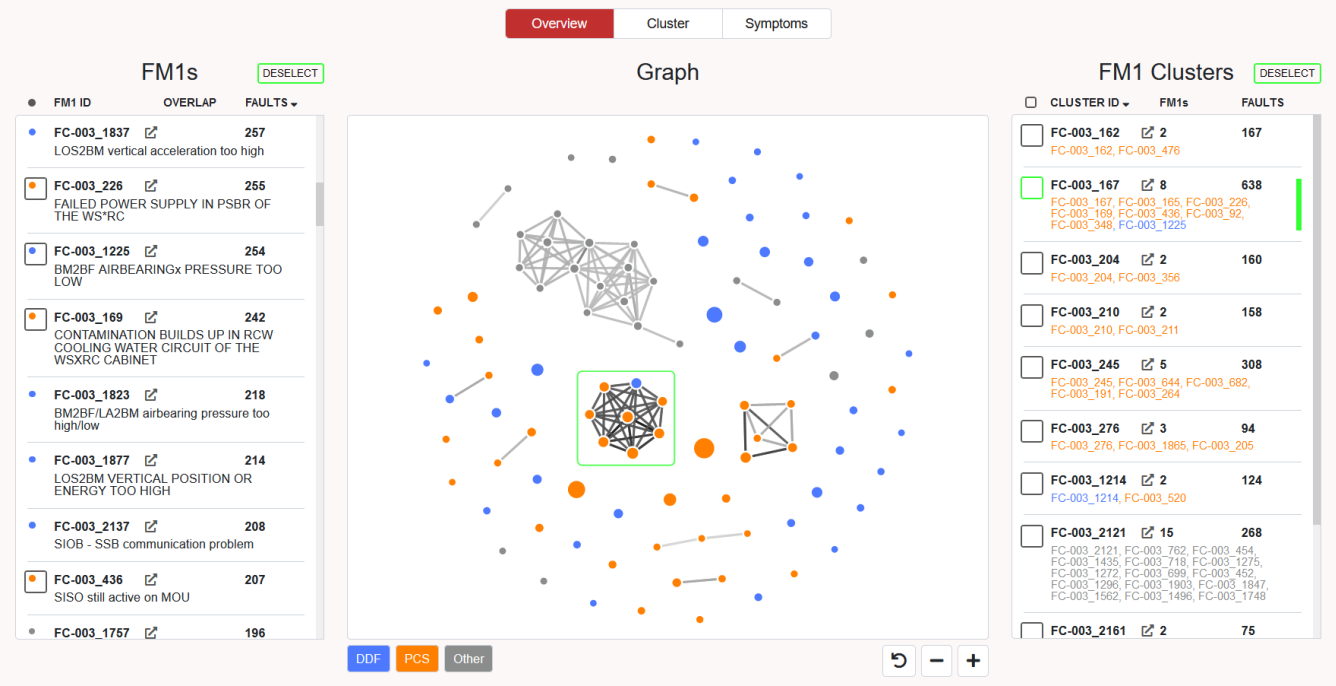

Figure 46: overview view with an FM1 cluster selected

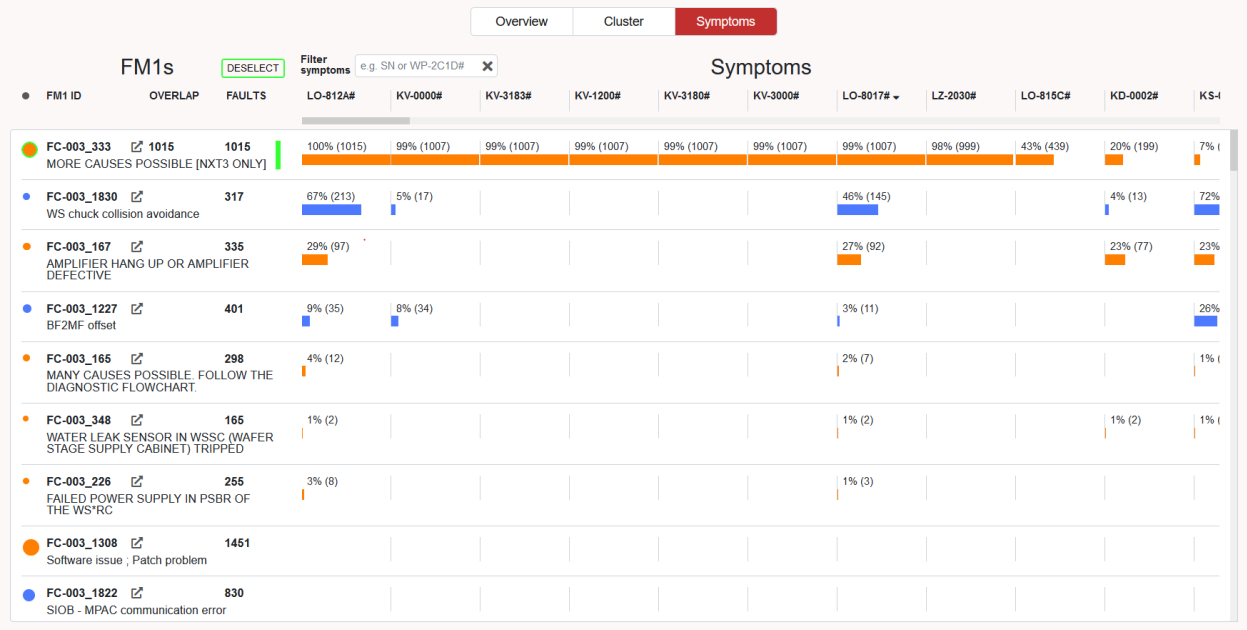

Figure 47: symptom view with an FM1 selected and sorted

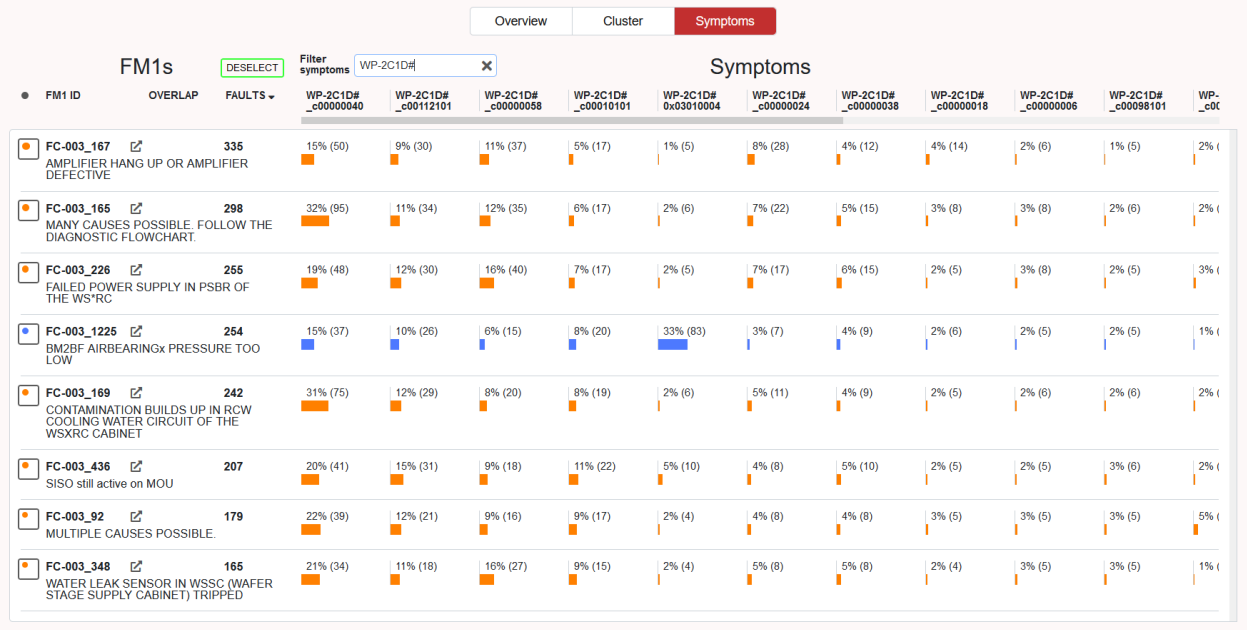

Figure 48: symptom view state with an FM1 cluster selected and a symptom filter applied

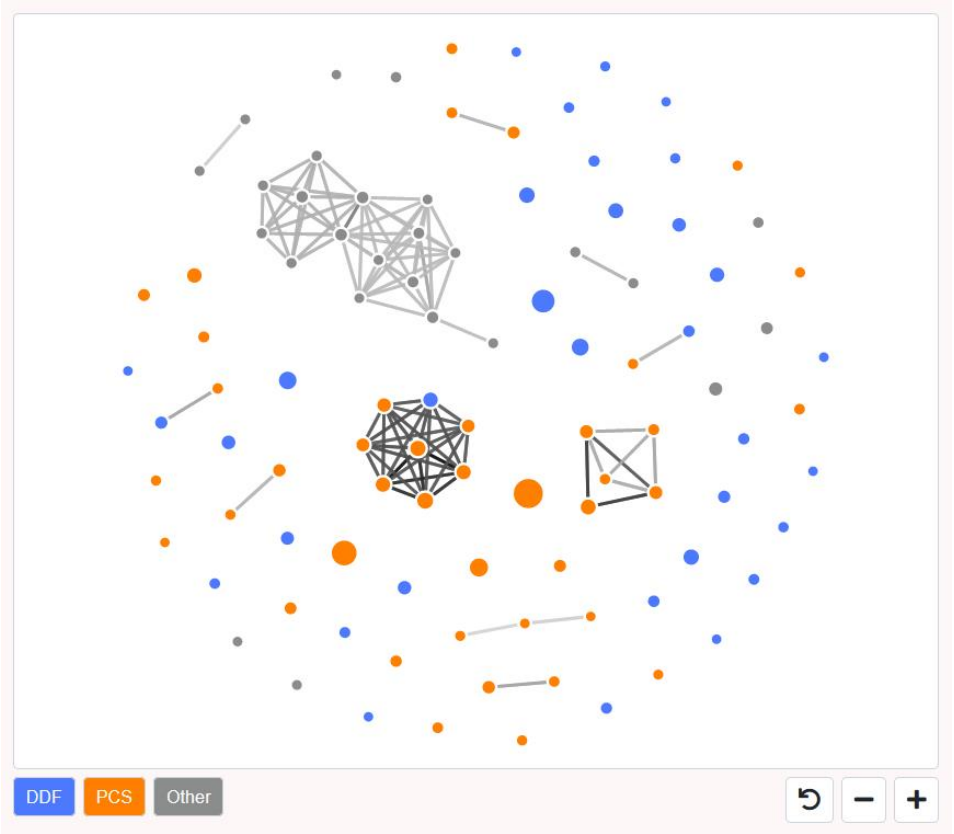

Figure 49: a close-up of the graph

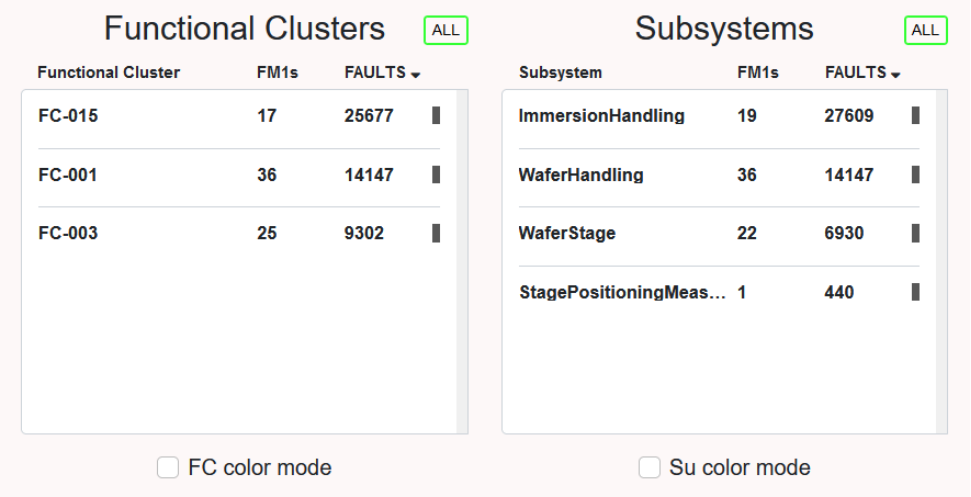

Figure 50: default Functional Clusters and Subsystems lists

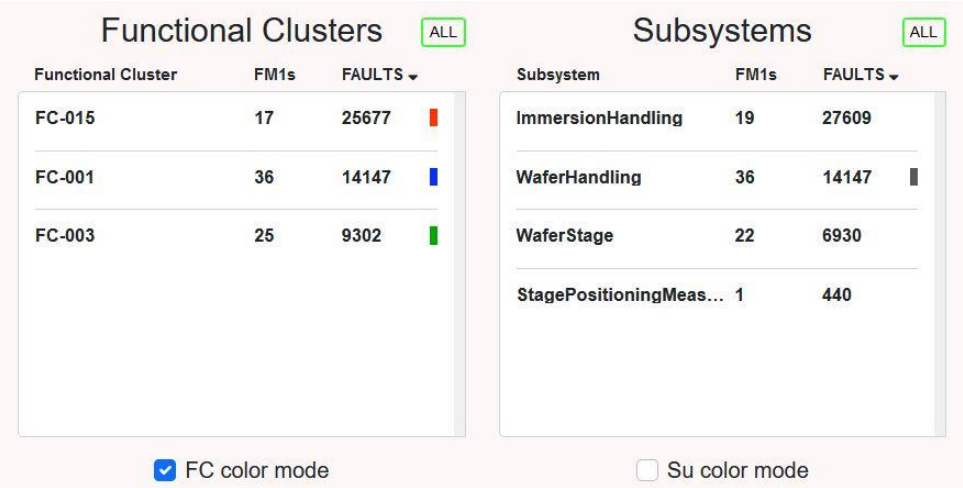

Figure 51: Functional Clusters and Subsystems lists with FC color mode and one SU selected

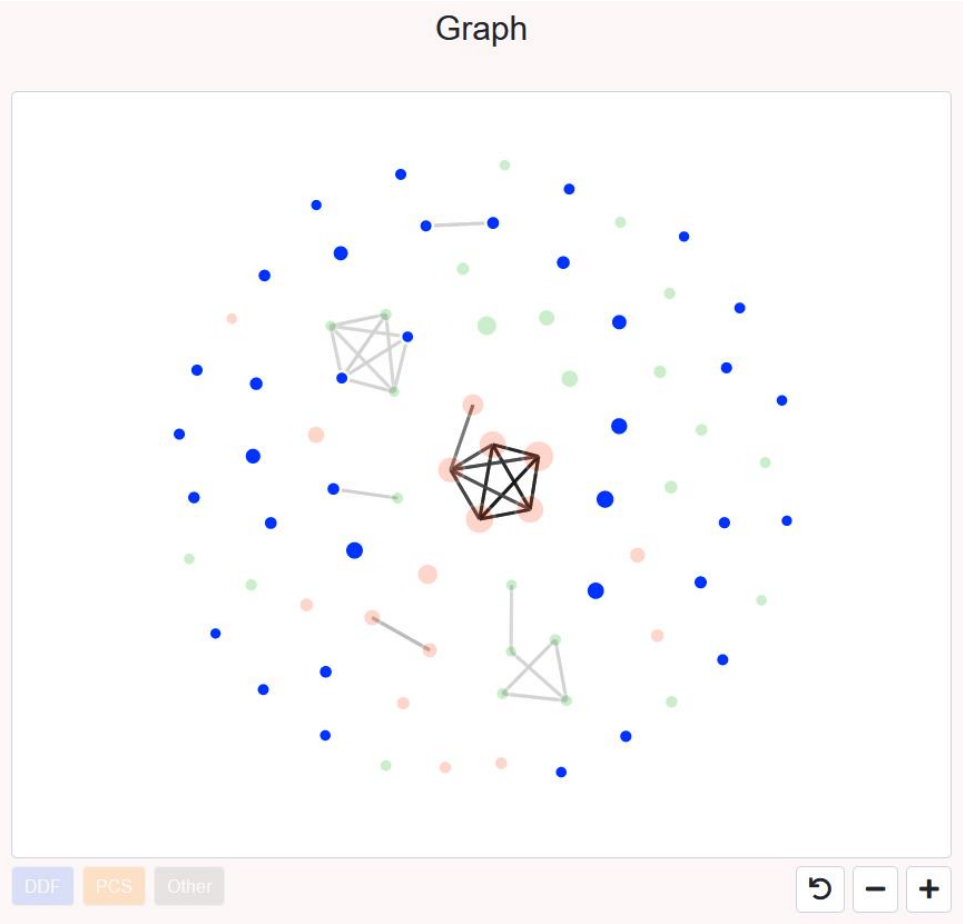

Figure 52: the graph corresponding to setting the Functional Clusters and Subsystems lists as in Figure 51

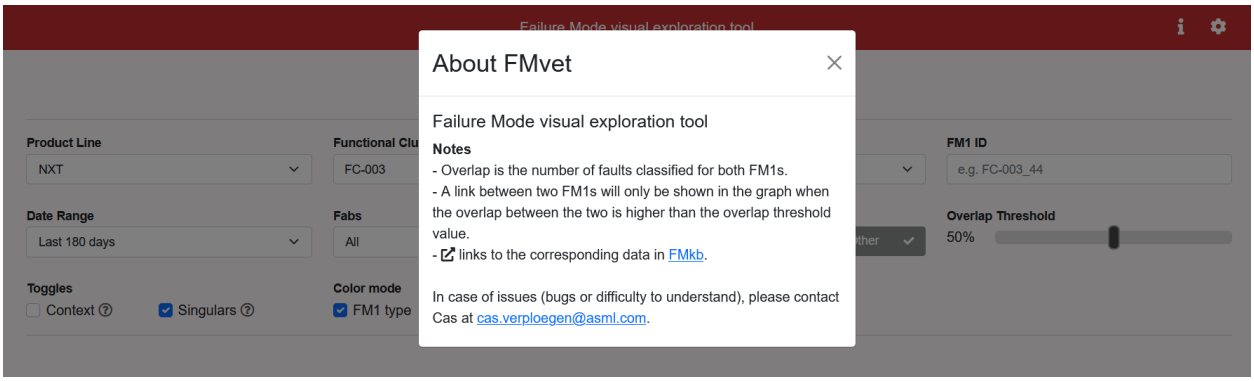

Figure 53: information popup

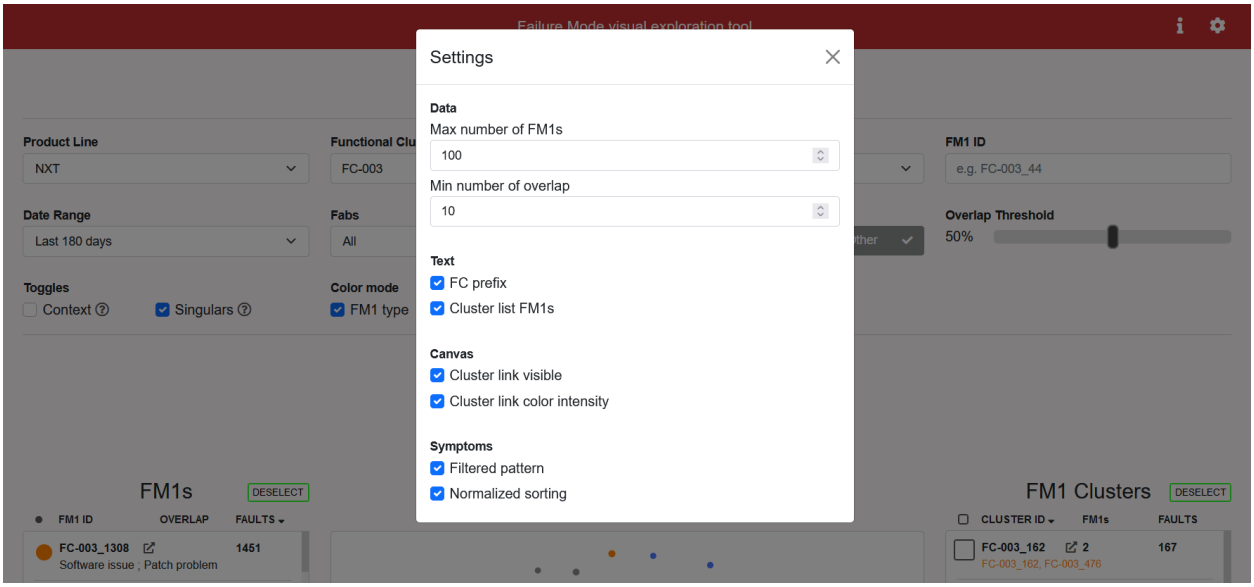

Figure 54: settings popup

D Specification FM1 overlap query

The specification of the FM1 overlap query, as used by the FMkb developer to integrate API support for requesting overlap between a set of FM1s.

Endpoint

/v1/fm1s/overlap

Input

Name	productlineCode
In	query
Description	A single product line code
Required	true
Type	string
Enum	["XT", "NXT", "NXE", "SRC"]
Example	"NXT"

Name	fm1Ids
In	query
Description	A list of Fm1 Ids separated by a comma (,)
Required	true
Type	string
Example	"FC-003_1225,FC-003_165,FC-003_165,FC-003_226"

Name	restrictTargetFm1s
In	query
Description	Whether target (fm1B in query) Fm1 Ids should be restricted to the fm1Ids set
Required	false
Type	boolean
Default	true

Name	minimumOverlap
In	query
Description	Minimum overlap threshold between two Fm1s
Required	false
Type	integer
Example	10

Name	startDate
In	query
Description	Start date of a range. Format: dd-MM-yyyy
Required	false
Type	string
Example	"09-10-2020"

Name	endDate
In	query
Description	End date of a range. Format: dd-MM-yyyy
Required	false
Type	string
Example	"09-04-2021"

Output

FM1A_ID	FM1B_ID	OVERLAP
FC-003_165	FC-003_165	21
FC-003_169	FC-003_169	18
FC-003_169	FC-003_165	16
FC-003_165	FC-003_169	16
FC-003_226	FC-003_165	15
FC-003_165	FC-003_226	15
FC-003_226	FC-003_226	15
FC-003_226	FC-003_169	12
FC-003_169	FC-003_226	12
FC-003_1225	FC-003_1225	7

Example query

```
SELECT
    fm1A.FM1_ID AS FM1A_ID,
    fm1B.FM1_ID AS FM1B_ID,
    COUNT(*) AS OVERLAP
FROM
    FMKB_L5.FM1_DIAGNOSIS fm1diagA
        INNER JOIN FMKB_L2.TASK_T f ON fm1diagA.TASK_T_FK = f.PK
        INNER JOIN FMKB_L5.FM1 fm1A ON fm1diagA.FM1_FK = fm1A.PK
            INNER JOIN FMKB_L0.FUNCTIONAL_CLUSTER fc ON fm1A.FUNCTIONAL_CLUSTER_FK = fc.PK
            INNER JOIN FMKB_L0.SUBSYSTEM ss ON fm1A.SUBSYSTEM_FK = ss.PK
                INNER JOIN FMKB_L0.PRODUCTLINE pl ON ss.PRODUCTLINE_FK = pl.PK
    INNER JOIN FMKB_L5.FM1_DIAGNOSIS fm1diagB ON fm1diagB.TASK_T_FK = f.PK
        INNER JOIN FMKB_L5.FM1 fm1B ON fm1diagB.FM1_FK = fm1B.PK
WHERE
    pl.CODE = 'NXT'
    AND fm1A.FM1_ID IN ('FC-003_1225', 'FC-003_165', 'FC-003_169', 'FC-003_226')
    AND fm1B.FM1_ID IN ('FC-003_1225', 'FC-003_165', 'FC-003_169', 'FC-003_226')
    AND f.AS_OF >= TO_DATE('20201009', 'YYYYMMDD')
    AND f.AS_OF <= TO_DATE('20210409', 'YYYYMMDD')
GROUP BY
    fm1A.FM1_ID,
    fm1B.FM1_ID
HAVING
    COUNT(*) > 10
ORDER BY
    COUNT(*) DESC;
```

E Specification symptoms query

The specification of the symptoms query, as used by the FMkb developer to integrate API support for requesting the (un)filtered symptoms of one FM1.

Endpoint

/v1/fm1s/symptomfaults

Input

Name	productlineCode
In	query
Description	A single product line code
Required	true
Type	string
Enum	["XT", "NXT", "NXE", "SRC"]
Example	"NXT"

Name	fm1Id
In	query
Description	A single Fm1 Id
Required	true
Type	string
Example	"FC-003_1225"

Name	minimumFaults
In	query
Description	Minimum symptom faults to return
Required	false
Type	integer
Default	0
Example	2

Name	filteredPattern
In	query
Description	Whether the symptoms should be pulled from the filtered or unfiltered pattern
Required	false
Type	Boolean
Default	false

Name	datetimeStart
In	query
Description	Start datetime of a range. Format: dd-MM-yyyy
Required	false
Type	string
Example	"09-10-2020"

Name	datetimeEnd
In	query
Description	End datetime of a range. Format: dd-MM-yyyy
Required	false
Type	string
Example	"09-04-2021"

Output

SYMPTOM	FAULTS
DN-1042#	7
WP-0404#	7
DN-1017#	7
DN-1043#_x00000004	7
WP-0400#	7
WP-2C06#	7
WP-2C1D#0x03010004	7
WP-0400#_x00000015	7
DN-1013#	6
SM-820.#_x00000009	6
SM-0009#	5
OE-1802#	5
TH-0002#	4
TM-1103#	4
TX-0003#	3
GF-2109#	3
MG-60C9#	3
GF-210B#	3

Example query

Filtered pattern query

```
SELECT
    sc.SYMPTOM_CODE AS "symptom",
    COUNT(sc.SYMPTOM_CODE) AS "faults"
FROM
    FMKB_L5.FM1 fm1
        INNER JOIN FMKB_L0.SUBSYSTEM ss ON fm1.SUBSYSTEM_FK = ss.PK
            INNER JOIN FMKB_L0.PRODUCTLINE pl ON ss.PRODUCTLINE_FK = pl.PK
        INNER JOIN FMKB_L5.FM1_DIAGNOSIS fm1diag ON fm1.PK = fm1diag.FM1_FK
            INNER JOIN FMKB_L2.TASK_T f ON fm1diag.TASK_T_FK = f.PK
                INNER JOIN FMKB_L5.UNFILTERED_PATTERN_IN_CNTXT upc ON f.UNFILTERED_PATTERN_IN_
CNTXT_FK = upc.PK
                    INNER JOIN FMKB_L5.FILTERED_PATTERN fp ON upc.FILTERED_PATTERN_FK = fp.PK
                        INNER JOIN FMKB_L4.SYMPTOM_IN_F_PATTERN sfp ON fp.PK = sfp.FILTERED_PA
TTERN_FK
                            INNER JOIN FMKB_L4.SYMPTOM_CODE sc ON sfp.SYMPTOM_CODE_FK = sc.PK
WHERE
    fm1.FM1_ID = 'FC-003_1225'
    AND pl.CODE = 'NXT'
    AND f.AS_OF >= TO_DATE('20201009', 'YYYYMMDD')
    AND f.AS_OF <= TO_DATE('20210409', 'YYYYMMDD')
GROUP BY
    sc.SYMPTOM_CODE
HAVING
    COUNT(sc.SYMPTOM_CODE) >= 2
ORDER BY
    COUNT(sc.SYMPTOM_CODE) DESC;
```

Unfiltered pattern query

Notice only the difference in the joins:

- FMKB_L5.FILTERED_PATTERN fp → FMKB_L5.UNFILTERED_PATTERN up
- FMKB_L4.SYMPTOM_IN_F_PATTERN sfp → FMKB_L4.SYMPTOM_IN_UF_PATTERN sup

```
SELECT
    sc.SYMPTOM_CODE AS "symptom",
    COUNT(sc.SYMPTOM_CODE) AS "fault"
FROM
    FMKB_L5.FM1 fm1
        INNER JOIN FMKB_L0.SUBSYSTEM ss ON fm1.SUBSYSTEM_FK = ss.PK
            INNER JOIN FMKB_L0.PRODUCTLINE pl ON ss.PRODUCTLINE_FK = pl.PK
        INNER JOIN FMKB_L5.FM1_DIAGNOSIS fm1diag ON fm1.PK = fm1diag.FM1_FK
            INNER JOIN FMKB_L2.TASK_T f ON fm1diag.TASK_T_FK = f.PK
                INNER JOIN FMKB_L5.UNFILTERED_PATTERN_IN_CNTXT upc ON f.UNFILTERED_PATTERN_IN_
CNTXT_FK = upc.PK
                    INNER JOIN FMKB_L5.UNFILTERED_PATTERN up ON upc.UNFILTERED_PATTERN_FK = up
.PK
                        INNER JOIN FMKB_L4.SYMPTOM_IN_UF_PATTERN sup ON up.PK = sup.UNFILTERED
_PATTERN_FK
                            INNER JOIN FMKB_L4.SYMPTOM_CODE sc ON sup.SYMPTOM_CODE_FK = sc.PK
WHERE
    fm1.FM1_ID = 'FC-003_1225'
    AND pl.CODE = 'NXT'
    AND f.AS_OF >= TO_DATE('20201009', 'YYYYMMDD')
    AND f.AS_OF <= TO_DATE('20210409', 'YYYYMMDD')
GROUP BY
    sc.SYMPTOM_CODE
HAVING
    COUNT(sc.SYMPTOM_CODE) >= 2
ORDER BY
    COUNT(sc.SYMPTOM_CODE) DESC;
```