CHALMERS
UNIVERSITY OF TECHNOLOGY

Software Prediction Viewer

Improving understandability by visualizing future data over time

Master's thesis in Interaction Designs and Technologies

DUR ABUZAID

Department of Applied Information Technology
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Master's thesis 2017:NN

Visualizing Defect Prediction Models to Assist Software Resources and Efforts Distribution

Improving understandability by visualizing future data over time

DUR ABUZAID

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Applied Information Technology
CHALMRS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Visualizing Defect Prediction Models
to Assist Software Resources and Efforts Distribution

Improving understandability by visualizing future data over time
DUR ABUZAID

Supervisors:
Josef Wideström, Chalmers University of Technology
Jesper Derehag, Ericsson AB a Telecommunications Company
Staffan Bjork, Chalmers University of Technology

Master's Thesis 2017:NN
Department of Applied Information Technology
CHALMRS UNIVERSITY OF TECHNOLOGY
Division of Interaction Design
SE-412 96 Gothenburg
Telephone +46 31 772 1000

In collaboration with:
Ericsson AB, a Telecommunications Company.
Gothenburg, Sweden 2017

Software Prediction Viewer
Improving understandability by visualizing future data over time

DUR ABUZAID
Department of Applied Information Technology
Chalmers University of Technology

Abstract

Human brains process and retain information more rapidly when it is presented visually. Information visualization tools have been used by companies to assist analytical experts in gaining deeper insights about software data. The present study proposes a design for a visualization tool in the form of a program that uses, predicted/future data to amplify cognition by presenting raw data visually. Companies are currently interested in monitoring software prediction models that provide information regarding software defects levels to enhance understandability of the product status, find reasons for high levels of defects, and thus make informed decisions more quickly. This thesis was initiated in light of the interest of the Ericsson Company in Sweden to visualize software prediction model data. The methodology encompasses understanding software prediction models and users' needs, tasks, and environments. Subsequently, I propose a design that visualizes the requirements regarding what future software data is needed. Finally, an evaluation of the design conducted with the end users to ensure that it fulfills the goal of the thesis, meets users' needs and in order to make further adjustments. A human- centered design (HCD) approach was followed to facilitate the proposal of the final design of Prediction Viewer and meet the goal of this thesis. The design approach was divided into three phases and the result of each phase was used as an input to the next phase, which encourages repeating these phases and frequently refines the solution. The user research involved 16 participants from Ericsson divided into three user groups, where each group represents a user level in the organization's structure (system, subsystem, and file levels). Of the three focus group sessions conducted with the three user groups, 18 groups of requirements were defined, representing their needs for a system that visualizes future software defects. The most notable reasons are understanding defect causality, software quality, release/assessment management, defects fix planning, test case selections, and highlighting the most error- prone & code smell areas. Given those requirements and needs, a semi-interactive prototype called Prediction Viewer was developed representing information regarding future software defects. Prediction Viewer is a web page that presents information regarding defects and defect priority levels over time. Development teams are interested in predicting the number of defects on system and system parts levels; the type of defects; the defect phase found during the testing phases; the answer code assigned to each defect; and the platform where each defect is found. All this information was shown to enhance awareness regarding software status, thus improving the ability to make informed decisions. Finally, a feedback session was conducted with participants from each user group in order to evaluate the prototype, the importance of

the information presented, and the way it was presented. Most of the suggestions were then applied to the final design.

Keywords: Software, Visualization, Machine learning, Future data, Software prediction, Human centered design, User centered design.

Acknowledgements

I would like to thanks my academic supervisor, Josef Wideström of Chalmers University of Technology, who guided and supported this thesis by providing regular feedbacks and suggestions, thereby making this thesis possible.

This thesis work was supported by Ericsson, Sweden. I would thank Ericsson for giving me the opportunity to conduct this thesis with the help of development teams. I would also like to thank all the focus group participants for offering their time to make this happen.

I would like to express my sincere gratitude to my industrial supervisor, Jesper Derehag, for his support during the writing of this thesis by providing insights and expertise and for his patience, motivation, and enthusiasm that greatly improved this thesis throughout numerous consultations. He supported me throughout the entire process of researching and writing this thesis.

While preparing this thesis, I received help and guidance from the teacher Thommy Ericsson, who deserves my greatest gratitude. He provided me with a book related to information visualization, gave me the opportunity to discuss some design issues, and assisted me with design suggestions.

Dur Abuzaid, Gothenburg, February 2017

Contents

List of Figures

List of Tables

1
Introduction

One of a project manager's target concerns is to understand and improve software quality during the development lifecycle and to deliver a high quality product to the end users. However, during the software evolution cycle, several factors might introduce defects in the product that negatively affect the software quality. These negative aspects could include an increase in the size of the development team (Jones and Bonsignour, 2011) or a series of repeated software changes. Inspecting and fixing software defects in complex software have been the greatest known expense for some time, especially when defects are detected after software delivery (Harrower, 2003 and Jabangwe, Petersen, and Mite, 2013). An unexpected number of high severity defects found during the software testing period increases testing periods and might cause a delay in software deployment. mention, software maintenance accounts for the largest portion of the total cost of software development. Fixing bugs after project delivery costs as much as 100 times more than fixing them during the development process (Jones and Bonsignour, 2011).

Therefore, in order to achieve a high level of quality, most software companies have developed reliable measurements systems, such as software defect predictions models, to guide them in controlling the status and progress of an ongoing project in earlier stages (Derehag, Weyuker, Ostrand, and Sundmark, 2016). These models have been used by companies to predict a number of defects in an ongoing software development process before the software's release (Fenton and Neil, 1999 and Yang, Tang, and Yao, 2015). Most of these models are statically calculated based on historical information of software changes and a variety of software metrics to predict numbers defects (Yang et al., 2015 and Gyimothy, Ferenc, and Siket, 2005). Moreover, some of these models provide a wider variety of information about functional, nonfunctional and structural software quality which in turn helps steer development, resource allocation, and maintaining efforts efciently (Jones and Bonsignour, 2011 and Rawat and Dubey, 2012),), reduce development time, and thereby reduce software expense (Rawat and Dubey, 2012). In this thesis, the defect prediction model focuses on measuring the functional quality of software source code.

There is a need for a design opportunity to better interpret such sensitive information as that which results from these prediction models. One way for the human brain to process many data points more quickly is through graphic display. Abuzaid and Titang (2013) mention that representing data in graphical format reduces the analysis time and provides extra help for the development team member to grasp a vast quantity of numerical information easily and quickly without a need for under-

standing the complex measurements of the model. Making insights based on a very large list of numerical data is not feasible. Therefore, an appropriate visualization of the result of these prediction models would enhance predictors experience by providing them with an effective and simpler way to gain a holistic view.

In academia, there have been a number of debates regarding about software metrics visualization techniques to enhance understanding of software complexity and quality (Abuzaid and Titang, 2014 and Erdemir, Tekin, and Buzluca, 2011). However, to the best of my knowledge, there is no particular approach for visualizing the present and future predicted defects of software source files over time.

1.1 Research Questions

The aim of this thesis is to explore the standard defect prediction model through collaboration with development teams at Ericsson AB to propose a visual approach that maps information-related defect predictions with optimal visual elements to enhance the analysis of a large number of software source files, thereby supporting the development team in everyday work activities and decision-making. This empowers software managers, executives, and quality assurance personnel to gauge quality trends, preempt future defects, identify the factors that influence the quality of the software, derive a list of tasks, and track the project regarding quality and goals targets during the software evolution. The focus of this thesis is on the fields of software defects prediction and information visualization fields. The final design should enhance the analysis of software defects, therefore the following research question was defined:

How to visualize predicted defects of software source code over time?

In order to answer this main research question, the following supportive questions have also been defined:

- If we can predict defects, what would the development team use this ability for?
- How should we visualize the use cases gathered from the development team?
- What information is needed to build the visualization?

1.2 Scope

To limit the study scope, we focus only on 2D interface techniques. Through a discussion with the supervisor, we have determined that presenting information about software metrics (input of the defect prediction model) will not indicate the causes of defects, but rather the quality of source code, which is not the goal of the present study. Excluding information-related software metrics is beneficial: it facilitates the adoption of other prediction models in the future by other stakeholders. Interaction

mechanisms could be used in this study depending on the amount of data gathered and the complexity of the final design.

1.3 Stakeholders

One of the primary stakeholders of the present study is Ericsson AB, represented by an industrial supervisor and a development team. Their main goal for this study is to identify the types of predicted information needed to facilitate decisions during software development process and to effciently prioritize tasks and allocate resources in order to deliver a quality product more quickly. Their input, feedback, and cooperation were therefore required throughout this project.

The designer of this study is another primary stakeholders; they are concerned with improving software development teams' working habits by developing a visualization tool that presents predicted/future data. Moreover, she is concerned with identifying challenges in visualizing software predicted information and design process.

The last and most important stakeholder of this project is the target user of the tool. Their participation and feedback have impacted the design decisions and the final design of the visualization. The way in which the information is presented has been influenced by their experiences and stories regarding other visualization tools they have used.

One of the primary stakeholder of this thesis work is Ericsson AB represented by an industrial supervisor and a development team. Their main goal of this study is to identify types of predicted information needed to facilitate the decisions regarding during software development process and efficiently allocate resources to deliver a quality product faster. Therefore, their input, feedback, and cooperation were required throughout this project.

1.4 Structure of Thesis

This thesis paper consists of nine chapters. Chapter One is an introduction of the thesis problem and objectives. In Chapter Two, I discuss the contextual framework of this thesis and describe the defect prediction process and defect detection process at Ericsson AB. IN Chapter Three, I present definitions and descriptions of software visualization, visualization tools, and software developers cognitive processes in data analysis. Chapter Four is a presentation of the design methods followed in this study. Chapter Five includes a description of the process of the three focus group sessions conducted to collect requirements and methods at the designing and evaluation phases. In Chapter Six, we present the results collected from the three focus groups. Chapter Seven presents the visualization tool developed, Prediction Viewer. In Chapter Eight we discuss the methods and the final results, and we

provide recommendations and future work. Finally, in Chapter Nine we present the conclusions drawn from the result of this study.

2

Contextual framework

2.1 Defect definition

Software defects is a well-known term used by software engineers to describe software errors, bugs, flaws, mistakes, and faults in software source code or/and design that might cause unexpected behaviors and in some cases lead to software failures if not recognized and mitigated immediately. Software defects usually arise during software evolution due to a misunderstanding within the development team. Given the expansion of the software market, software engineers must deliver high-quality and reliable products to gain users' confidence and satisfaction. Many measurements have thus been proposed in various kinds of literature to reduce defects in software products, such as software quality metrics and software defect prediction. Azeem & Usmani (2011) argued that it is impossible to prevent the injection of defects in a software product. However, they could be reduced. Some defect types could be ignored depending on their severity. Unfortunately, severity levels defers across industries: they depend on the bug tracking tool used to report defects during the development process.

2.2 Product metrics as predictors

Software metrics are measurements computed using software or product properties as an input to the measurement. They enable software development teams to gain insight by providing a wide range of information about software quality, performance, and cost, thereby, improving software quality. Li (1999) divides software metrics into two categories: product and process metrics. Li (2006) uses four categories of software metrics: product metrics, development metrics, deployment metrics, usage metrics, and software and hardware configuration metrics. In this paper, we focus on the product metrics and development metrics used by (Li, Herbsleb, Shaw, and Robinson, 2006). Product metrics measure the properties of the product being developed, such as size and complexity, while process metrics measure the properties of software development processes such as code changes and process effciency

2.3 Description of a general defect prediction process

Several defect prediction approaches have been proposed in the literature to facilitate defect inspections and subsequently plan for software quality improvements (Han, Jiang, Zhang, and Sun, 2013, Kim, Zimmermann, Whitehead Jr, and Zeller, 2007 and Fenton and Neil, 1999). Prediction models require defects and metrics collected from existing software development efforts as an input to the model and then calculate the total number of defects as an output. Some of these models only estimate the present number of defects hidden in the software source code while others predict the present and future number of defects in the software source code files. Li et al. (2006) suggest predicting a defect rate that shows the total number of defects distributed over time rather than only predicting the number of present defects of software files. Most of the literature runs the model before the testing phase, while some emphasize the use of the model during the development phases. Li et al. (2006) classify prediction modeling into four categories:

- Relationship: the model aims to establish a relationship between the predictors (metrics) and the field defects.
- Classifications: the model aims to uncover whether the number of defects is above the limit.
- Quantities: the model aims to predict the total number of defects.
- Rates of occurrences over time: the model aims to predict the defect occurrence rate.

2.4 Setting

In this exploratory study, we examine the defect prediction model called machine earning, which is being used by a large multinational organization, Ericsson AB. They offer communication technologies and services and operate in around 180 countries. Ericsson has recently begun using a defect prediction model and aims to enhance their software quality by proposing a visualizing technique representing defect prediction information for better assessment, exploration, and analysis of future defects. This visualization would help the development team to:

- Reduce errors rate in their software before delivery,
- Decrease the number of errors discovered by testers after product delivery, thus improving user satisfaction
- Identify areas of improvement,
- Assist project progress,
- Plan for defect detection activity based on predicted defect magnitude, and
- Preempt defect discovery.

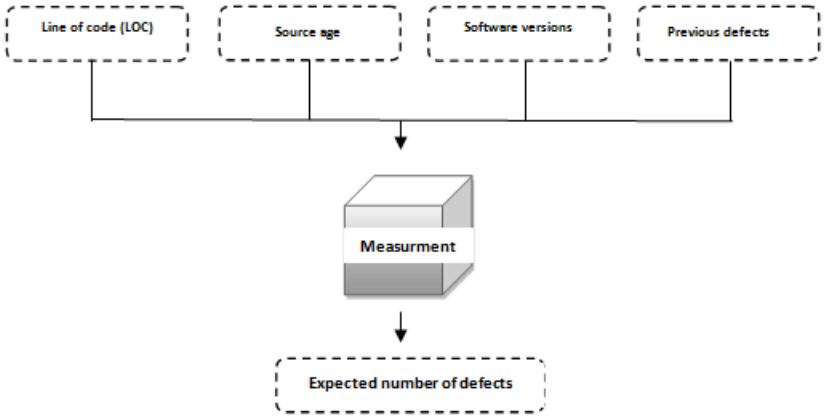

Figure 2.1: Software prediction model used by Ericsson

For a meaningful defect prediction, four input parameters are involved in the model as predictors for the number of future defects, namely line of code (LOC), software versions, age, and previous defects. The model performs some measurements and the result is the total number of predicted defects per software file (see Figure 2.1). The model measures defects for each software file, which means it produces a list of numbers where each number corresponds to a software source code file. Each file could also have several numbers of predicted defects representing the future. Developers can predict the future of a software file by changing the value of the age variables (input of the model) to predict the number of defects at a certain point in the future.

3

Theory

In this chapter, the theory, a definition, and concepts that this study is based on are presented.

3.1 Software developers' cognitive strategies

This study explores the cognitive design elements that exploratory tools are advised to support in order to enhance software comprehension and reduce cognitive overhead. Many exploratory tools have been proposed in the field of software engineering to assist programmers in their daily work activities. However, few of them are used in practice. Therefore, (Storey, Fracchia, and Muller, 1999) point to the importance of understanding the cognitive processes that programmers use to construct mental models of software structure and base the visual design on them.

In the following section, a list of various cognitive models software that maintainers uses to create mental models of the source code are discussed. Moreover, we discuss the cognitive elements guidelines to guide the development of exploratory tools and thereby enhance the comprehension of software.

Top-down program comprehension approach: Programmers trace source code from top to bottom. This approach is used when programmers have some concerns about the program nature and structure.

Bottom-up program comprehension approach: In this cognitive process, software maintainers read the source code from bottom to top. The programmers combine and divide statements into higher levels of abstraction to attain program comprehension and construct a mental model.

Knowledge based understanding model: In this model, programmer uses any of the two previous approaches to understand the program and build a mental model.

Systematic and as–needed program: Programmers systematically trace the source code files in detail to gain global understanding of the program or focusing on understanding a particular part of the source code related to the task at hand. The latter is called an as-needed approach.

An Integrated metamodel of program comprehension: This method refers

to programmers who frequently switches between the following three comprehension models: (1)top-down model, (2) program model, and (3) situation model.

The following section describes a hierarchy of cognitive design elements to take into consideration when developing exploratory tools.

Several factors that influence the developer's understanding process and cognitive performance include the developer's expertise or program complexity and size. Therefore, this study also proposes a hierarchy of cognitive design elements and guidelines elicited from the various cognitive models that the software maintainers propose above. The first branch of the tree focuses on introducing design elements to enhance the comprehension process of each cognitive model. For instance, an exploratory tool that supports bottom-up comprehension strategy includes a graph that provides access to the lowest level unit in the program and shows the relationships between them. The second branch focuses on reducing the developer's cognitive overhead . This occurs by (1) facilitating navigation mechanism, (2) suggesting orientation instructions, and (3) reducing disorientations that usually occur when developers trace distributed codes among several files.

3.2 Defect prediction models

Companies are interested in the field of software prediction. Many studies have extended or proposed models other than the standard model (see Section 2.4) for predicting defects in software products for various purposes. Rawat and Dubey (2012) evaluate compatible software defect prediction models introduced in previous studies and discuss possible improvements for each model. Project managers use defect prediction models to assist them in time management and cost reduction. Fenton and Neil (1999) and Rawat and Dubey (2012) show that defect prediction models reduce defects magnitudes.

Hewett (2011) proposes another defect prediction model to predict errors in software source code and estimate the repair time needed to support test management and task prioritizing. While (Li et al., 2006) share their experience in selecting defect prediction models at ABB for planning better risk management, managing resource allocation effciently, and enhancing future product testing. Gyimothy (2005) explains the model's measurements of detecting proneness to defects and shows a relationship between object-oriented metrics and fault proneness.

3.3 Software visualization

Many papers introduce, evaluate, and improve existing defect prediction models. However, to the best of our knowledge, there are no papers that address visualizing present and future defect prediction magnitude in software source code. Some papers focus on visualizing detected bugs (reported bugs), while (D'Ambros and Lanza, 2006 visualize detected software bugs along with software evolution. This

integrated information helps characterize the evolution of software artifacts such as documentations and test cases. In this visualization, rectangle shapes are used to represent different versions of the software; the color of the rectangle indicates the level of bug severity in a certain artifact during the period of software evolution; and the border color of the rectangle indicates periods of software stability.

Toroi et al. (2012) visualize the defects distribution of three companies using a histogram that shows the total number of defects against the type of defects (functional, requirements, or interface). This visualization assists software engineers to prioritize bug fixing based on the type and facilitates inspection activity. The study shows that high defect magnitude was in the functional aspect of the software product.

3.4 Information visualization

The human brain tries to perceive data in the environment by searching for patterns in the data (Ware, 2004a. Moreover, Storey et al. (1999) claim that humans attempt to develop their mind map when exploring complex data to easily remember and retrieve this information in the future. In fact, humans gain more information through vision compared to other senses (Ware, 2004a. Furthermore, humans spend less time and effort receiving a great deal of information through visual exploration. Based on this, many visualization tools have been proposed to support developers' awareness in their daily activities (Sedlmair, Isenberg, Baur, and Butz, 2011) and (Margaret Storey, Čubranić, and German, 2005). Information visualization is a graphical representation of complex and abstract data in a structural way to amplify and reinforce humans' recognition, analysis, and exploration of data, thus increasing the understandability of certain phenomena. Companies uses visualization tools to investigate and deepen their understanding of particular data.

The benefit of information visualization is not to create an attractive display of the data, but rather to (1) facilitate searching for data through building patterns, (2) help form hypotheses, and (3) permit the detection and perception of unexpected emergent properties (Ware, 2004a. Ware (2004) proposes two categories of visualizations, sensory and arbitrary. In the case of sensory visualization, the information is represented in a way that matches the human perceptual process, so it is understandable and does not need to be learned. However, arbitrary visualization requires some learning effort because the information presented has no perceptual biases.

An increasing number of information visualization tools have been proposed in the field of software engineering to support the analyzability and exploration of data. Erick (2002) demonstrated a visualization called "ADVIZOR" to represent information regarding code changes using several windows. Another visualization tool called "softChange" aims to support the development team in developing insights on the evolution of the software (German and Hindle, 2006). Sarma et al. (2003) propose a tool aimed at increasing awareness of distributed software development

teams called “Palantir”.

3.5 Designing information visualization tools

Information overloading is a fundamental issue: companies are struggling to cope with the ever-growing amount of data gathered. Therefore, interest in visualization tools has increased so as to make these volumes of data accessible. The role of graphical representations is to accentuate important data properties and relationships between different data items, which in turn helps observers use their own visual perceptions to analyze data more quickly and easily.

One of the main challenges in designing exploratory tools is representing information overload in limited space. Therefore, this is a tedious task; designers should be creative in mapping information with visual representation so that it conveys the meaning of the whole data set and decreases the demand on the observer's memory. Additionally, designers should limit the amount of information users perceives using dynamic actions while somehow keeping the observer aware of the whole data set.

Carr (1999) proposes a design guideline for designing information visualization applications. These guidelines provide suggestions on how to represent abstract information in a graphical form that is easy to understand. Moreover, the guideline suggests that designers develop visualizations where the information is organized in a way that allows the user to immediately, easily, and quickly recognize, analyze, and make sense of them. The first point in the guidelines is to decide whether the visualization is an optimal choice: a visualization should be considered only when there is a large amount of data where the user goals are not easily achieved. Moreover, the design should support specific users' tasks rather than general visualization. Additionally, consider the data type when choosing visualization method. Carr (1999) discussed seven visualization methods along with their advantages and disadvantages which are: 1-dimentional, 2-dimentional, 3-dimensional, temporal, network, hierarchal and multi-dimensional. Finally, Carr, 1999 suggests only using a 3-dimentional view when it is necessary to map data to a physical representation.

Additionally, Carr, 1999, Kang, Gorg, and Stasko, 2011 and Pousman, Stasko, and Mateas, 2007 suggest different dynamic actions to be used with each of these visualization methods to enhance exploration and facilitate the study of the data presented. Navigation and zooming do not eliminate the need for data filtering. Navigation and zooming do help the user focus on certain data items by concentrating on a particular area of the visualization. However, filtering helps eliminate a whole class of scattered data points and focus only on the interesting data items. Additionally, another way to manage an overloaded amount of data and limited display space is to consider multiple views. Auxiliary windows should be updated with an additional subset of the data as soon as the main window is manipulated. Moreover, designers could use windowing techniques to represent several graphical views of the same data. Moreover, it is important to evaluate the visualization with the users and consider their needs. Finally, designers should consider the user's

technical abilities while designing complex visualizations.

Diehl (2007) introduces similar conceptual aids in classifying visualization by determining the scope, content, form, method, interaction, and effectiveness of the visualization to be designed. The concept is be used in the present study to gain a better understanding of future visualization characteristics.

4

Methodology

Proposing an integral visualization requires looking into several previous studies which have proposed interdependent phases to aid in designing software visualizations. Diehl (2007) designed an approach that starts with data acquisition, analysis and finally suggests visualization. A human centered design approach proposes a similar designing approach with the focus on understanding human factors, such as human behaviors, tasks, and environments, and establish requirements to develop a usable interactive system. To guide this study, the nonlinear HCD approach which will be used particularly by iterating through the following design stages proposed by (IDEO, 2015):

1. Develop a general understanding of the topic, participants, company and design methods and process.

2. Inspiration and eliciting information regarding the current defect detection process at the company. To do so; two information gathering methods used which are interviews and focus groups.

3. Ideation and Creation of a low fidelity prototype based on the information gathered from 2. At this stage, several iterations on the low fidelity prototypes created and evaluated based on some information visualization heuristics as suggested by (Forsell, 2012) and possibly with developers.

4. Implementation and developing a high fidelity prototype result out of the final low fidelity prototype. This prototype used in the final evaluation session as it helps in communicating the idea better than low fidelity prototype and evaluate user interface elements

5. Evaluation and presenting proposed design concept to the stakeholders through conducting focus groups.

The ISO-standard for Human-centered design (ISO 2010) proposed six principles that should be followed in all HCD approach. These principles are activities focused on different perspectives that designers should follow to produce an effective interactive system. In this study, five of these principles were followed: Understanding users, their environments and tasks were taken into consideration in the design. To address all user needs, I iterated and refined the designed idea based on user evaluation and other evaluation methods. The user's feedbacks on the design idea were taken into consideration and were applied to the final prototype. The remaining principles were addressed except one principle which is a multidisciplinary team as I work individually in this thesis project.

Human centered design approach can be viewed as an umbrella method containing the following two approaches which will be considered during the design process of this study.

Activity Centered Design (ACD) and User Centered Design (UCD) are both methods that encourage user's involvements (Norman. 2006). The difference is ACD is focusing on the activity developers perform in daily work to determine product functions. While, UCD is focusing on the developers needs and desires (Williams, 2009 and Gulliksen, Lantz, and Boivie, 1999).

I believe, in this study, a combination of ACD and UCD approaches would help in designing effective visualization based on understanding developer's needs and activities. To apply these two concepts, focus group sessions will be conducted to help in understanding the development team daily activity, tasks and problems. The identified problems will be viewed as the developers needs.

IDEO (2015) proposed several methods that could be used at the three stages of HCD to aid in eliciting information from the users such as, focus groups, semi-structured interviews and build your own methods at the inspiration phase. Focus group process proposed by (Kontio, Lehtola, and Bragge, 2004) was used at the inspiration phase.

5

Process

Different methods were used at each of the designing stages mentioned (see Section 4). The methods were inspired by the Field Guide to Human Centered Design (IDEO, 2015) and (LUMA, 2012)). In this chapter, I describe them, along with their benefits and impact on this study.

5.1 Study Subject

The standard model used by Ericsson AB (Derehag et al., 2016), focuses on improving the quality of the software by predicting the number of future defects in the software source code. The study subjects could be any member of a software development team who is interested in improving the software. They could be project managers, software testers, or software developers at the company who are involved at any stage of a development process. In this study, three user groups are included as representatives of each organizational level (see Section 5.3.2).

5.2 Data source

Three data sources are involved in this study, namely the defect prediction standard model, members of a development team at the company, and previous research papers. The defect prediction model is a defect measurement system used by Ericsson AB (Derehag et al., 2016).

The defect prediction model is an essential source of data source in this study. It uses several software metrics as predictors to measure the future/expected numbers of defects in the software files. The model takes four predictors as inputs, which are software versions, age, previous defects, and lines of code (LOC). It contains information to be presented, such as predictors or number of predicted defects and software versions. However, this information might not be relevant to the development team and might not provide a complete picture for the development team to base their decision on. Therefore, there is a need to validate this concern, especially when the development teams are not aware of the model's benefits and do not have enough expertise on the topic To do so, we first involve members from software development roles (as suggested by Giacomin, 2012) to understand and explore their daily activities (as suggested by Norman, 2006), namely the defect detection process. In this study, members of a software development team working at Ericsson AB, are

involved as informants in order to examine the relevance of the information extracted from the model by understanding their daily activities. Furthermore, I explore any additional information needed to enhance defect prediction exploration and analysis. Previous research papers and tools related to information visualization, specifically software visualization, visualizing uncertainty, and visualizing software granularity, have been used as inspiration during the designing phase.

5.3 Inspiration

5.3.1 Interview

A discussion has been conducted with the supervisor at the company in order to acquire first-hand information regarding the challenges in current practices and the purpose of defect prediction visualization. Some issues were raised regarding (1) how to visualize present and the future numbers of defects of large software artifacts files, (2) how to define the time scale (months or releases), (3) how to present values to indicate the uncertainty of the predicted values, and (4) whether we should visualize the granularity of software files (system, subsystem, and files).

5.3.2 Focus groups

The main aim of the focus group sessions is to address the four issues mentioned previously (see Section 5.3.1). Studying the current practice at the company would increase the advantages and usability of the visualization. Therefore, the purpose of using this method is to gather qualitative insights through group discussions. Most of the developers participating were not familiar with the defect prediction models concept. Therefore, to reveal insights for the analysis stage that were not visible, this method is used to enhance understanding of the model and users' needs, and to inspire and build upon each other's ideas. Moreover, it is an opportunity to gain a more precise understanding of data analysis habits, requirements, and goals and the current tools used by development teams to detect and report defects in their products. These focus group sessions have been planned based on the guideline and tips provided by (Kontio et al., 2004) and (Rennekamp and Nall, 2000).

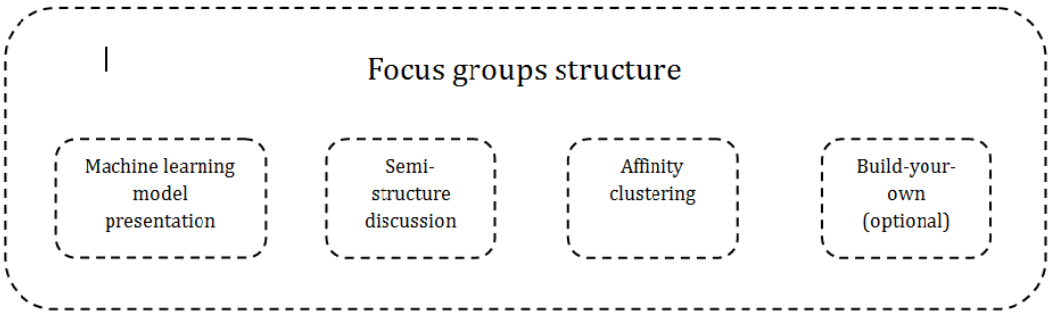

Figure 5.1: Focus group methods

5.3.2.1 Objectives and selection of participants

Software development teams consist of members with different goals, views, work habits, and experience. Therefore, in order to account for most of these experiences and collect detailed information, three focus group sessions were conducted at the site in Sweden for approximately two hours, where a group of relevant participants sharing the same work background (working on the same hierarchy level at the company) were involved in the same session. The participants were chosen based on the following work pyramid at the company, Figure 5.2.

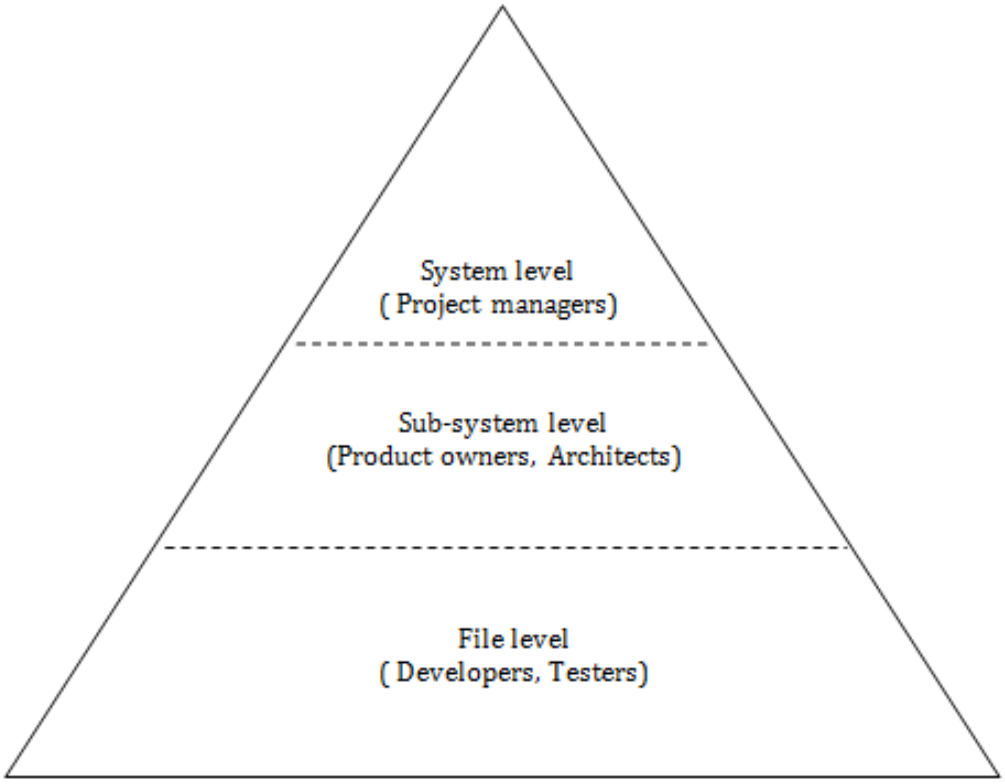

Figure 5.2: Organizational work structure

The pyramid consists of three levels that are vertically and horizontally interconnected and interdependent. At the pyramid's peak (system level teams), project managers work to analyze defects and data, release readiness and product quality reports, and predict the product delivery period. At the file level, the development teams are less concerned about analyzing defects; rather, they focus on analyzing codes and resolving technical issues. Therefore, the development team's needs for defect prediction increase toward the top layer of the pyramid and decreases toward the bottom layer of the pyramid. Management teams mostly work on data analysis and defects managements because they need this information for planning and resource allocation. However, developers working at the file level work less frequently with data analysis;, rather, they work with implementing and maintaining products.

In this study, three focus group sessions have been conducted beginning with the least relevant user group for this study which is the developers at the bottom level (file level), then the architects at mid-level (sub-system level), and finally managers at the top level of the pyramid (system level). Each session consists of four to six participants.

In the first session, we have invited developers working on the file level, which is on fairly low-level granularity, such as software developers and testers. Developers at file level are responsible for feature implementation and trouble report (defects) fixing. Since they are not concerned with analyzing defects, they are the least relevant user group of this study. The idea is to conduct the sessions from low-level granularity toward the highest level of the pyramid so that there is a chance to improve in the next two sessions and to gain a basic understanding of the working structure and terminology used. However, due to time constraints and scheduling conflicts, a focus group session with the user group at sub-system level was postponed. Therefore, for the second session, we have invited a project management group working on the system level. Upon reaching the top level of the pyramid, more analysis is needed for decision making. Managers at the system level are therefore the most relevant and important user group for the design of the exploratory tool. In the third session we have invited architects, a line manager, and product owners, all of whom are responsible for the middle level (sub-system level) in the pyramid.

5.3.2.2 Planning the focus group session

Each focus group session consists of three complementary activities and an optional activity. The first part is a presentation of the defect prediction model, while the second activity is a semi-structured discussion focusing on understanding the current process, understanding participants' role at the company, and preparing them intellectually to give constructive feedback during the following activities. The third and fourth activities have been used to elicit and summarize the participants' needs and insights using the clustering technique and build-your-own method. In the following paragraphs, a detailed description of the structure of each session and the methods used is given.

Each focus group session begins by welcoming the participants, introducing the aim of the session and the activities to be done. The participants are reminded that the session is recorded and encouraged to share their opinions and to think loudly during the session.

Presentation of prediction model
The session began with a presentation of the prediction model aim, input, and output. Participants were encouraged to ask questions about the model as well.

Semi-structured discussion
The second part of the session was a semi-structured discussion where the question area was defined previously to stimulate and encourage the discussion These ques-

tions focused on eliciting information regarding the current defect discovery process and development process at the company (see Appendix A.1). The participants were first asked to introduce themselves and their roles, while at the same time I asked questions regarding their roles and activities.

Affinity clustering
The third activity aims to validate the information discussed earlier with the industrial supervisor. I chose affnity clustering techniques (LUMA, 2012) to summarize the information discussed during the semi-structured discussion and the elicit information needed regarding defects for decision-making. The participants were asked to spend a few minutes writing their answers on a sticky note and imagining that the prediction model presented earlier is working perfectly and that they can predict any type of information. The participants were encouraged to write any software information they would like to predict, such as defects properties, characteristics, or trouble reports that are needed for decision-making during the development process. Additionally, information for determining (cost, time, effort, replan) or indicating a high defect magnitude that might lead to software failures. Participants should write each insight on a sticky note. Once everyone was done, we discussed each response loudly and posted it on a white board. While the posting is being done, similar insights were grouped at the same time. When all the notes were discussed and similar insights were placed in one category, a name was given to each category. Participants were allowed to add more notes while we discussed the notes and while grouping them.

Build-your-own (optional)
The last activity was optional and was used in the sessions based on two factors, time availability and the amount of information gathered from the previous activities. The aim of this activity is to evaluate the type of defect information presented and the visual elements mapped to this information of the proposed visualization. A set of ten cards were distributed to the participants so each participant had ten cards. Each card contained a piece of information related to defects with a graphical representation. The cards were intended to help participants to visualize the information, be inspired, and give more feedback and insights. The participants were asked to read each card and to edit or change the information or graphical elements on the cards. Once the participants reviewed all the cards, we discussed each card and classified each as insightful or superficial.

5.3.2.3 Conducting the focus group sessions

The three sessions were conducted according to plan. However, some issues affected the way each session was performed. In the following paragraphs, reflections on the three sessions are presented.

First session: designers and testers at file level
In the first focus group session, five participants participated in the workshop, three of whom were software programmers, while the two other were testers at the file level.

The workshop started by welcoming them, followed by a short introduction of the content and the aims of the workshop. Then came an introduction of the prediction model for 15 minutes by the industrial supervisor. The next part of the session was a semi-structured discussion. Participants were asked to introduce themselves and their roles at the company while I interrupted and asked some questions that were prepared previously or questions raised during the discussion. Participants were encouraged to ask questions and raise their voices or give their opinions at any time during the workshop. In order to summarize the participants' thoughts, the clustering technique and build-your-own methods were performed in the last hour.

The former activity did not take a long time in this session compared to the other two sessions. As expected, the participants found it difficult to imagine the uses of the prediction model; one participant said, "Sorry, but I don't know how this might help me." However, some participants added more notes at the end of the activity after listening to each other's ideas. One of the participants had no notes, and the other had only one note. I would say that only three participants out of five was very interested in finding a potential use of the model. Finally, we grouped these thoughts and gave a name to each category. Three of the five participants were active. Some of the participants were confused and did not know how such a model would benefit them. However, I still wanted to determine some potential uses and how this might benefit them. Participants are not aware of the benefits and cannot see how predicting the future or the analysis tool to be developed might help them better perform their daily work.

The latter activity aims to evaluate the type of defect information presented and the visual elements mapped to this information. It focuses on evaluating the type of information to be presented in the visualization. The participants were asked to select, add, or change information on the cards and propose a visualization out of these elements on the cards that are needed for decision-making. They were then asked to group the cards into insightful or superficial cards to be discussed later. Fifteen minutes was not enough time for this activity. This method was important because non-active participants found it easy to imagine some potential uses, which inspired them to give more input to the previous activities.

The result of the sticky notes were various groups of important information needed to assist developers at the file level. More information may be found in Table 6.1)

Second session: Project managers at system level
In the second focus group session, four participants participated in the workshop from the system level; they were a department manager, a project manager, a release management leader, and a system architect. The workshop started by welcoming them, followed by a short introduction about the content and the aim of the workshop. Then came an introduction of the prediction model for 15 minutes by the industrial supervisor.

The session started with three participants; the fourth member attended after the presentation of the prediction model ended, while the other participants were introducing themselves. Even though only four participants joined this session, a great deal of input and insights were elicited. Teams at the system level are concerned about quality, time management, and resource allocation at the organization. They work daily with data analysis and base their decisions on facts and planning for the future. However, they need to continually replan because they are predicting the future based on their present intuitions. A participant said that we need to have a good intuition. They were interested in the model because now they can predict future based on values and data from their product, not based on their intuition, which helps them to plan and distribute resources early on in order to prevent sudden events that cause delays. During this session, participants were active, excited to know more about the model, and gave a great deal of input during the semi-structured interview and the affnity clustering (sticky note) activities. Therefore, we excluded the last method, which is the build-your-own cards activity. It was interesting that the participants decided to continue with the workshop activity instead of taking a break. Even though we gained 15 minutes, we could not perform the last activity. Conducting a focus group with four participants were sufficient when only one moderator is participating. The whole session was recorded and many questions were asked during the discussion. There was an opportunity for every participant to participate and express their opinions.

The main results from this session focus on predicting the quality and quantity of the defects earlier for resource allocation and time management. More information can be found in Table 6.1.

Third session: Product owner and architect at sub-system level
The third and last session was with product owners and architects from the subsystem level in the organizational work structure (see Figure 5.2)), where the teams work as a connection between the system and file levels. The teams at this level are concerned with code structure, quality, and refactoring within the subsystems, especially the subsystems they are responsible for. In this session, we had a surprising number of participants: there were six participants, including two design architects, two test architects, line manager, and an architect. It was therefore difficult to keep track of the time because of the number of participants; we thus decided to remove the build-your-own activity and instead focus on the affnity clustering (sticky note) activity and the semi-structured interview. Some participants spent a long time discussing their own issues and interrupting others by providing examples. Furthermore, they could not stop discussing possibilities for implementing ideas even though we had mentioned several times to assume that the prediction model is working correctly. Two hours for this session was insufficient given the number of participants. Due to the number of participants, we spent 30 minutes instead of 15 minutes in the model presentation since each participant had several questions regarding the model. The semi-structured discussion also took more than 30 minutes; in turn, we needed to decrease the time of the break. Moreover, having one moderator supervise six participants and keep track of time and instruments was a challenge.

The main results from this session focus on predicting the quality and quantity of the defects earlier for resource allocation and time management. More information can be found in Table 6.1.

5.4 Ideation

At this stage, an initial design element is elicited from the previous stage (inspiration). Several methods proposed by (IDEO. 2015) will be used at this stage.

- The ideation phase starts by using the download your learning method, gathering all the information from the inspiration phase, and writing it down.
- followed by generating ideas by finding a theme and patterns from the data collected from the five focus group sessions.
- At this stage, information about design for single or multiple visualization panels is visible.
- A method called "How we might" was used to ensure the viability of the ideas and completeness of functionality and that all the stakeholders' needs and activities were addressed.
- Create scenarios out of the participants' stories.

5.4.1 Download learning

Transcribing video and voice record process
During the inspiration phase in Section 5.3.2, video and voice tapes were used to record the discussions. All the video/audio was converted to text based on the guidelines provided by (Guidelines, 2011). These transcribed documents were then sent to the stakeholders to validate their inputs during the focus group session and to answer a few questions that were raised while copying the records. Since the participants might not have time to do this, I made the review easier for them by associating each participant's name in speech with a unique text color in the transcribed note (see Appendix A.2). Moreover, to reduce reading time and encourage stakeholders to clarify the documents, all jokes, clarification of the activities during the session, implementation questions regarding the model, and the presentation of the model have been excluded from the transcribed document. Even though I tried to make the notes short and focused on the relevant discussion, each note was between four to six pages. Therefore, I highlighted the new questions raised during transcribing the documents with red color in order to be quickly recognized by the participants. However, only 2 of 15 participants have read, answered questions and clarified their text.

Two different devices were used to record all the discussions during the sessions. Both the voice recorder on my phone and video tabbed using a camera were tested before the first user encounter. However, the voice recorder sometimes stops working while the camera can only record half an hour and I have to manually press the recording button every half hour, which makes me miss recording some of the discussion at the file and sub-system level meetings. The missing parts of the sessions

were clarified in the transcribed document.

During the sessions, some participants attempt to push their ideas and stop others from giving their opinions. For this reason, the text notes give a second chance for the participants to express their needs and concerns freely. Also, it is another opportunity for them to change their minds or to support others' ideas. This could happen because they might have had the chance to think or discuss the topic after the session with their colleague and more questions, ideas, or opinions have been raised.

Analyzing affinity clustering method
The results of the affinity clustering used in the three focus group sessions were organized in a table (see Table 6.1). Three lists of requirements have been gathered from the three user groups of this study.

These three lists of requirements were organized into a table for further analysis. It consists of three columns where each represents a user group and their needs. Each row accounts for a category of similar needs gathered from the affinity clustering method during each focus group sessions and assigned to a group with a defined name.

In total, there are 22 categories of requirements representing more sub-features of needs. Each group consists of one to six different sub-requirements. The following steps were applied to reduce the complexity of analyzing and designing these requirements:

Grouping: Since the affinity clustering method was conducted three times with the different user groups, some groups of requirements were similar but given in two categories using different names.

Organizing and sorting: These groups of needs were then classified into two categories, either relevant to data prediction or irrelevant. Two irrelevant categories were named Application Programming Interface (API) and reduce information, which discusses visualization ideas and design suggestions. Some sub-requirements were mapped incorrectly and did not match that category's purpose. These requirements were moved to a suitable category.

Prioritizing: Given all the previous steps in minimizing the load of requirements, it was necessary to prioritize relevant requirements by importance. The user needs were divided into three levels, high level, low level, and future work.

- High-level requirements are demands related to the prediction model and related to predicting the future during software development with the aim of preplanning and resource allocation at early development phases.

- Low-level requirements are requirements related to predicting the future. How-

ever, the company does not have enough data to run a calculation. Low-priority requirements are these needs that we might look at if there is an opportunity to engage in one of the existing views that does not increase the complexity of the design (see Section 6).

- Future work requirements are the requirements that are not feasible or not possible to predict; however, these requirements are still related to prediction and would return value to the development process. These users' needs were not involved in this study but rather noted for future investigation and as opportunities for further studies.

Concept mapping Concept maps aid in visually illustrating the relationship between different data entries and organize thoughts. Concept maps help build a thorough understanding of the users' needs and similarities between various user needs. Identifying similar needs narrows down the problem and facilitates design ideas. Three user groups participated in this study; without creating a concept map, it would therefore have been difficult to remember them or to understand the connection between the requirements and the corresponding user group quickly during the design process. The concept map therefore helped structure the thoughts and improve our understanding of the users' needs by visualizing them using circles and a line connecting the relationships between the needs.

During the ideation phase, I faced several challenges. One of the challenges is which type of graphs are suitable for presenting uncertain data and information over time. Another concept map was developed to address the design challenges and propose ideas. It draws the relationships between the design challenges and different design solutions. Concept map was also used to save various design ideas, cycle through the design solutions, the optimal choice.

Statement starter
Upon examining the requirements in each category, I realized that some requirements were different in purpose but similar in their visual elements. From a design perspective, several conclusions can be made from a group of requirements. One visual might therefore lead to several conclusions. These requirements were combined into a statement. Some requirements were mentioned several times in different ways by different user groups with different work perspectives. Designing these views requires almost the same information; combining these requirements in a statement would reduce the complexity of the design and reduce the number of requirements, which would in turn contribute to minimizing the complexity of data perception.

Similar statements were then combined and prioritized based on importance in order to facilitate the design process. The requirements mentioned by all user groups were also at the top of the list, so statements that cover different user groups' needs are at the beginning of the list while other features mentioned less frequently are at the end of the list. Furthermore, the list was used as a validation document at the end of the ideation phase to ensure that the information mapped to the visual design

addresses all needs.

Another list was created for low priority requirements. This list used at the end of the ideation phase to investigate the possibilities of including them in the design concept without increasing the perceived complexity.

Statement starter lists were written on paper. It was wise not to digitize the list so that it is reachable and design does not become distracted by scrolling down and having many steps required to find a particular statement, such as opening folders and files to view the document. The list was used frequently during the ideation phase; each time a design idea proposed it was compared against the list and other requirements that could fit in that particular design view were defined.

Customer journey
At each of the three organizational levels (see Section 5.3.2), many employees collaborate and merge their knowledge and experiences to deliver a product with a quality that matches the company's reputation. At each of these organizational levels, many employees have different backgrounds, working habits, and goals. During the focus group sessions I met with representatives have different needs, it was difficult to remember each participant's goals for using prediction visualization. Even though the transcribed documents were used as reference, it was too long to be referred to during the ideation process. The customer journey method was therefore used to summarize the transcript and for rapid access to information.

The customer journey is divided into three column and three rows. Each column represents one of the organizational levels from the top of the hierarchy down to the bottom. Each row represents the goal of the work and purpose of using a prediction an analytical tool visualizes predicted data and needs to address these goals. The needs were translated to information related to prediction.

The need of such a prediction tool increase as in the higher levels of the organizational structure. For instance, employers at the system level have greater demand than designers at the file level.

5.4.2 Generate ideas

At this stage, all gathered information was listed, sorted, and prioritized. There were three ways to proceed at this stage; each presented certain benefits and challenges.

The first way consists of using pre-existing 2D graphs such as pie charts, bar charts, or histograms. These diagrams are commonly used, so no learning effort is required. Moreover, many tools allow for generating these diagrams, so there is no need to design them. However, the amount of data does not fit in existing diagrams.

Second, to make all this information fit in a view, I considered developing a new visualization using visual attributes. The purpose is to be able to fit more information in a view by applying Gestalt law, a set of a design principles for information visualization. These principles help humans brain to easily perceive and recognize patterns out of big data (Ware, 2004b). It is related to associating data with objects, color, size and other characteristics in order to create a visualization that is easy to recognize and perceive patterns in. The idea behind inventing a new type of visualization is to be able to fit more information in a view without the need to create complex views and tools. However, presenting all prediction information in a view just created more complex views which demand more memory because of the amount of related information. The information was connected; different combinations reveals more data patterns and insights, thus several views was needed.

The third way is to search for existing analytical tools and insert the data to produce a visualization. There are many visualization tools with different interaction and graphing functionalities. These tools are limited in their types of views and interactions; some cannot be further developed or programmed. Consequently, the requirements and design must be adapted to the visual elements that these tools provide. It is difficult to consider many of the user requirements and usability requirements, so there are trade-offs and different challenges. Existing tools use diagrams that were not recommended, such as stacked bar. Given the time frame of this thesis, this was not studied in depth. However, Existing visualization tools require deep analysis to know whether they are able to fit the design of Prediction Viewer and if it addresses the design challenges and requirements of data prediction. This method of visualization was therefore not explored. To the best of my knowledge, this thesis is a first attempt to visualize predicted data, so it would be of greater benefit not to constrain ourselves to the visual views that these tools provide.

5.5 Prototyping

The designing tool will be used to prototype the final idea is determined at this stage.

Listing the design challenges helps structure the brainstorm of the possible designs. The list was used as a hint to solve a puzzle. Each challenge can be imaged as a piece of a puzzle. To produce a complete picture of the design, I used a trial-and-error approach by moving the puzzle pieces until they all fit into the right places. Moreover, I used the concept map to show the relationship between requirements and another concept map to show the common requirements between the user groups in parallel with brainstorm ideas. The most common charts and a few other charts suggested by the stakeholder were placed beside those two concept maps at this stage for inspiration. They are also used as rules to control the design and validation documents for the proposed design. Additionally, they were used to validate my design against stakeholders' and users' needs and constraints.

The schematic diagramming method proposed by LUMA, 2012, was used to outline

the structure and the components of the tool. Initial ideas were sketched on paper using color pens to quickly explore alternatives and easily modify and combine the sketched ideas and develop an initial design. Paper sketching helped in the transition of ideas from the mind onto paper, allowing participants to capture thoughts and rapidly extend ideas (Rohde, 2011 and Egger, 2000). 2D charts largely present users' requirements intuitively and clearly; however, visualizing uncertainty level was a challenge. The views are condensed when presenting uncertainty levels, especially for those views presenting detailed information. Therefore, we come to the conclusion of having several views presenting a group of related information or users' needs.

When the initial design was sketched and the charts types were chosen, the information was placed and presented clearly; a paper prototype was created using papers, colored pens, and sticky notes. It presents the user interface elements and the content. Generally, it was used to communicate the initial design workflow and the user needs with the stakeholder. Sketches and paper prototyping methods are quick to develop and easy to iterate through and improve. However, one particular advantage of paper prototypes is that they are inexpensive and quick to create. Thus, it is easier for the stakeholders to criticize and propose changes (Sefelin, Tscheligi, and Giller, 2003). Paper prototype facilitated discussion with the stakeholders and aided in improving and ensuring understanding regarding users' needs and the tool being designed.

When the initial design meets the stakeholder's expectations, a semi-interactive prototype was created using a prototyping tool called Axure (Axure Software Solutions, 2002–2016).The purpose is to show a realistic mock-up and consider the space between elements and elements' sizes. It helped to explore the tool's dynamic navigation in depth as well as its look and feel (Maguire, 2001). It was also used as an acceptance test to validate the final design idea and the distinct functional boundaries later in the evaluation phase.

5.6 Evaluation

Evaluation is an essential phase in any design process. Researchers can obtain feedback about the design, guidance for redesigning, and improvements preventing errors that might appear in the final design and measures if the goals and objectives of the design have been achieved.

Several evaluation classifications exist to assist in choosing the optimal evaluation approach and when to perform the evaluation based on several key characteristics. Bowman et al. (2002) propose three essential characteristics to be identified before choosing the evaluation methods in virtual reality, which are relevant to be used in this study. The three characteristics are: resources available (whether representative users will be involved or not), evaluation context; and type of the data to be produced (quantitative and qualitative). Another classification assists evaluators in determining at which designing stage to conduct the evaluation session (Andrews, 2006). The first is formative evaluation used during the design development phases

to receive feedback about the design and proposed room of improvements. The second is a summative evaluation to be used at the end of the development phases to test the final design and assess whether stakeholders' objectives have been achieved or not.

This study promotes iterative works and evaluation. Two evaluation types were used, formative and summative.

Continuous formative evaluations were conducted at the end of each designing phase with the industrial supervisor and domain expert. The purpose is to detect usability, analytical, and content issues at an early stage of the designing process that the representative users do not need to experience. This method is quick to plan and perform since it does not require end users' involvement and only requires deep analysis of the design by a domain expert.

The output of each phase was evaluated with the industrial supervisor, who is considered an end user and domain expert as well. The applicability of the requirements gathered during the data collection phase were discussed and refined with the supervisor. Additionally, the initial sketches and design challenges were communicated; based on his suggestions, a paper prototype was developed. The paper prototype was described by presenting some scenarios and moving the sticky notes, a result of user actions.

Finally, a summative evaluation where an analytical evolution was performed with the end users. Heuristic studies are well known and used in the HCD field to evaluate the usability of products (Forsell, 2012). Most of the heuristic studies concerned with evaluating the graphic interface and the controls of the system, however, do not evaluate the analytical and exploratory part of the visualization. Therefore, they might not be applicable for evaluating information visualization designs. Forsell (2012) proposed a modification guideline for evaluating information visualization with representative users. The aim of this evaluation session is to examine the proposed visualization promotes exploration and analysis of defect proneness files and how it might support developers' daily work practice. A high fidelity prototype of the tool was presented with different user scenarios. These scenarios were elicited from the three focus group sessions. During the presentation, participants were encouraged to interrupt with questions and opinions regarding the tool. The participants were supposed to evaluate the system against the 11 information visualization heuristics proposed by Forsell, 2012. However, due to time limitation, I instead asked the users to give a positive and a negative feature of the system.

6
Users' needs

In this section, the data collected are presented to answer the supportive research question of this thesis work, "If we can predict defects, what would the development team use this ability for?" Also, the design and process challenges defined throughout the thesis work are presented. All interpretation of the data collected and data exclusion process was discussed with the stakeholder at the company.

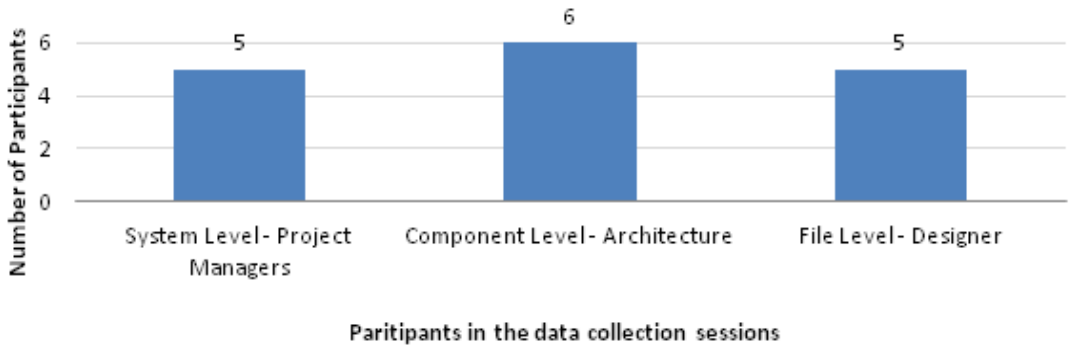

Figure 6.1: Participants in the data collection sessions

This thesis work was conducted in collaboration with Ericsson AB and the data was collected from 13 participants working at different levels in the organization (managers, architects, and developers) (see Figure 6.1). These participants collaborated through three focus group sessions, including a stakeholder, who participated in all the data collection and the evaluation sessions. Each session lasted two hours.

In total, 22 groups of requirements were derived from the three focus group session (see Figure 6.2). Each consists of different sub-needs/features. The three user groups frequently mentioned some of these requirements and some were unique. Some were repetitive or related but required relocation into a different group, and some were not related to data prediction; as a result, the gathered requirements needed further adjustments.

To better interpret the gathered requirements and reduce its complexity, the data was analyzed and reorganized; and similar groups were merged in collaboration with the supervisor/stakeholder at the company. Some requirements were excluded for several reasons: either they were not related to data prediction or not related to information visualization, or the requirements lack mathematical calculation or data,

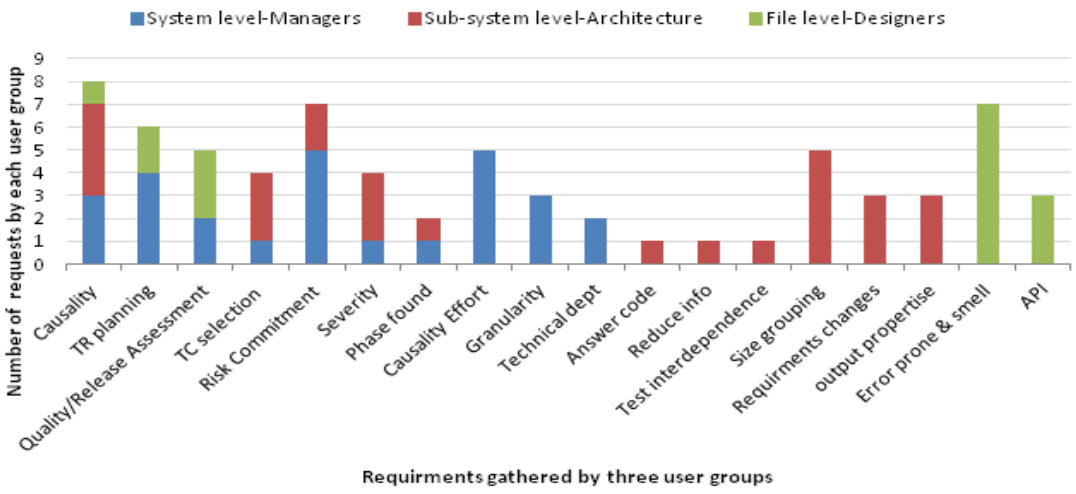

Figure 6.2: Requirements gathered by the three user groups

which is necessary to be able to run a calculation. Four groups of requirements were repetitive: thus, 22 groups of requirements were reduced to 18 groups. Ten of them were relevant to the thesis scope and considered in this thesis (see Figure 6.3), while eight of them were not related to the study scope. Thus, they were excluded. Below is a detailed description of all the requirements gathered and reasons behind the acceptance or rejection (see Section 6.1 & 6.2).

In total, ten groups of requirements were considered important and taken into consideration in the present study. Figure 6.3 shows that one group of requirements (causality) was mostly mentioned by the three user groups. However, five groups of requirements (phase found, quality/release assessment, test case selection, severity, and defect planning) were mentioned by two user groups. User groups at both system and sub-system levels in the organization were interested in understanding the phase in which the defects were discovered, the severity of the defects, and the number of defects affecting test cases (phase found, test selection, and severity), while project managers at system level and designers at file level were interested in predicting the quality level and the amount of defects for early planning and resource management (quality/release assessment and defect planning). These six groups of requirements were common between the user groups; thus, it should be easy to find them and access them in the tool.

Error prone and code smell and defect planning were the two groups of requirements mostly mentioned by participants (see Figure 6.3). Error prone and smell have seven features requested by the developers. On the one hand, developers are interested in predicting the future of defects, mainly in the areas (sub-systems/files) of their interest. On the other hand, defect planning was also mentioned seven times by both managers and developers. They were interested in viewing defects rates over time and viewing defects quarterly.

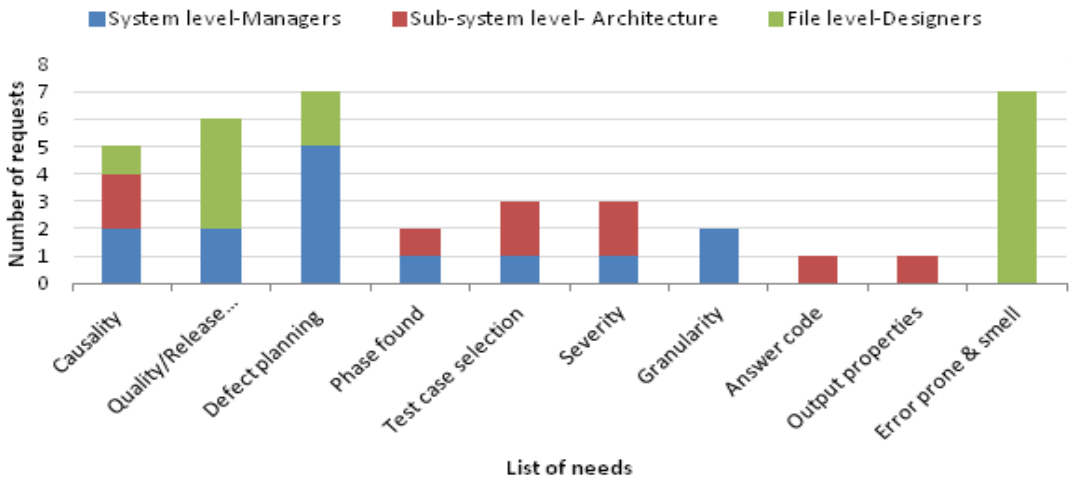

Figure 6.3: List of the considered needs/Requirements

Project managers were also interested in defects prediction for an early understanding of the reasons of a high number of bugs and determining which areas have more defects, allowing them to estimate the effort needed to plan and estimate delivery time more precisely. These have been derived based on the following groups: causality, defect planning, and granularity. Most of the groups of requirements were identified by the project managers at the system level (eight groups of requirements), while architecture at sub-system level took the second place (six groups), followed by the developers (four groups). Participants at sub-system level identified more requirements than project managers; however, many of these requirements were not relevant. Developers usually do not use analysis tools as frequently as teams at system and sub-system level. Therefore, they found it difficult to define requirements and imagine how such as tool might aid them during their daily work.

As seen in Table 6.1, many groups of requirements were defined by project managers and architects, while developers identified fewer requirements. There are two reasons they have fewer requirements: the first is that such a tool mostly is relevant for system level employees as it is already a part of their work, mainly to iteratively analyze the work progress and status and plan for improvements. As developer at the file level do not regularly work with planning, they found it difficult to image how such a tool would improve their work.

6.1 Relevant users' needs

This section presents all the considered requirements gathered and the purpose of each group of requirements, Figure 6.3. The ten groups of requirements considered in this thesis work are presented in Table 6.1, where the correlation of the common needs of user groups can be seen. A detailed description of these ten categories of requirements and their purposes is presented in the following paragraphs:

Causality is an important group of needs because since it the common needs defined by the three user groups (project managers, architects, and developers). This group is presented in the first row of Table 6.1. The user groups would like to know the causes of the past, present, and future numbers of defects reports. They have hypothesized several potential causes of errors, such as an increased number of contributors, frequent code changes, or introducing new features. They are also concerned whether the number of design tests created for testing system parts would increase or decrease the future number of defects reports. Some of these causes, such as a probability of hidden undetected number of faults, frequent code changes, and introducing new features were not taken into consideration as there is no way to acquire data on that to predict the future at present. The actual causes are found and their questions are answered through the visualization tool created.

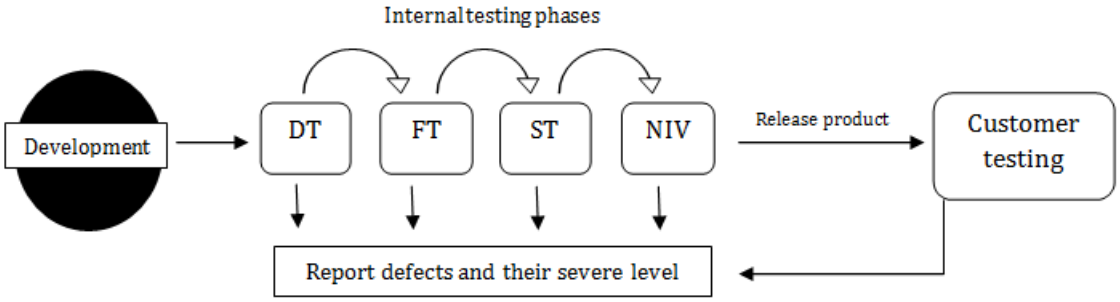

Figure 6.4: Defect discovery phases

The phase found group of requirements was identified by the project managers and architects. They would like to know at which stage in the development process each defect was discovered. There are two main resources of detecting and reporting defects. The first is internal: testers at different internal test phases detect and report defects. There are four sequential phases of detecting defects internally before product release (see Figure 6.4). The first testing phase is a real-time test during the development of the product called a designer/developer test (DT). The second testing phase is functional verification (FT), which consists of testing the functional area or requirements of the entire system. The third testing phase is Non- functional verification (ST), which is testing the nonfunctional requirements. The fourth and final internal testing phase is network integration and verification (NIV), which refers to testing the entire cell phone network by testing the product characteristics on the entire cell phone network rather than a single function. The latter is external defect detection reported by the customers on the release part of the software. However, developers were interested only in customer defects per area. On the one hand, the developer would only like to understand the quality of the files or areas they are responsible for. On the other hand, the architects would like to understand the overall picture. They are interested in understanding defects detection phases both internally and external by customers.

The quality/release assessment group of requirements was identified by project managers and developers. The project managers are interested in understanding the total number of defects of the next project release and release branches to estimate a sufficient amount of resources and a delivery time of that particular release. The developers are concerned with showing predicted information versus the actual number of defects found. Showing the predicted number of defects on release branch would help to understand the situation and avoid the need for any additional time before the product release. This group of requirements was considered in the present study.

The defect planning group of requirements was identified by project managers and developers. They would like to use the prediction tool to plan ahead and avoid surprises that usually happen before the release period. A project manager said, "The unexpected amounts of detected defects are frequently detected before release period which causes delay of the release, thus; they cannot give a precise delivery time to the clients." Therefore, they would like to view the number of current and expected defects over time for early preparation. The developers also suggest showing predicted number of defects versus actual defects found to estimate the accuracy of the prediction model and make the decision more precisely. There are various algorithms and studies conducted on how accurate the machine learning model is for predicting data. However, showing data over time addresses this issue. Presenting defects over time requires considering the internal development process. Project managers would like to view the detected errors about their release cycle (every three months or quarterly).

Test case selection is another group of requirements identified by both project managers and test architects. Test architects develop hundreds of test cases to determine whether a software part or a feature is working properly. Defect reports indicate the quality of the test cases coverage. Knowing that there are defects affecting a certain area of the software help test architects to expect which test cases will be affected, thus improving their test coverage. Furthermore, if no faults were found in a certain area of the software, it also indicates a lack of test coverage.

The severity group of requirements was identified by project managers and architects. They are interested in monitoring the severity of the defects found both internally and externally. It indicates the degree of the negative impact on the software under development caused by each detected defect. It also helps to estimate the amount of efforts and time needed to fix a defect. Different defect classifications are used by organizations and tracking tools. In the present study, we refer to the three classifications of defects (A, B, and C) known at Ericsson. Defects classified as A or B means they affect the functionality of critical data of the software, while defects classified as C do not affect the functional area but are rather associated with errors in the layout. Project managers and architects are interested in the early prediction of the number of future defects on levels A and B. On one hand, the project managers aim to plan and support the development team with the needed required resources and avoid planning for overtime as how it is, in the current process. On the

other, the architects wish to prioritize the redesigning and refactoring complex parts of the software source code and improve readability. Moreover, project managers give more attention to defect reports submitted by customers than internal ones as long as internal defects do not block the development process by having a high number of severe defects (A or B). Therefore, they were also interested in predicting severity of customer defects (defects reported by the customers) and internal defects (defects reported internally by development team).

The output properties group of requirements was defined by both product owners and architects at the sub-system level. They are interested in visualizing predicted output properties of the machine learning model, such as efforts, density (number line of code (NLOC) of changes for a defect), and correction complexity. However, only the total number of defects and defects density have been considered in the present study.

The granularity group of requirements describes the need to show a number of defects against software files structure. Software files are usually structured as a tree of software source code folders. The system level at the top of the tree consists of several sub-systems. Each sub-system contains up to hundred software source code files. Project managers identify this group of requirements because they are concerned with identifying the most defective areas of the software earlier in the development process to provide proper resources. Additionally, they are interested in viewing the correlation of customer defects (defects found by the customers) with the severity of the defects.

The answer code group of requirements was requested by the architects. When a defect is assigned to a team or a developer to be analyzed and fixed, the team report back (answer) by identifying the causes of defects and what the team should do about the defect. The answer code is the decision given for each defect; there are three answer code levels (A, B, and D). The answer code "A" is assigned to a defect that requires an immediate fix for the next product release, while "B" is assigned to a defect that need to be fixed in a future release, and "D" is assigned to a defects that are not faults but rather a misunderstanding of how the product is supposed to work or when a developer was unable to figure out what a fault is due to lacks of logs. Architects are mainly interested in predicting the number of defects having a D-answer code in the future.

The error-prone and code smell group of requirements was identified by the software developers; they are concerned with identifying the most error-prone areas of the software under development. There are different conclusions that can be made out of identifying most error- prone areas, such as determining an number of tests needed in each area and indicating weakness in the design and the need for more refactoring and redesigning. This gives an overview of the quality level of the software artifacts and estimates how much efforts needed to fix and maintaining defective areas.

6.2 Rejected user's needs

This section presents a description of the nine excluded groups of requirements and elaborates on data exclusion.

The causality effort group of requirements was identified by project managers at the system level. They need to estimate the amount of effort and time needed to fix the present and the expected future defects. Turn-around time (solution time) is another feature in this group of requirements where the users are interested in watching the life cycle of the submitted defects, particularly the time each defect spent at each defect life cycle state. A defect life cycle varies between companies and depends on the defect tracking tool used. The defects life cycle starts once a defect is assigned to the first state, "submitted", and ends when the fix has been verified and assigned to the last state, "closed". This feature helps managers optimize the defect life cycle. Additionally, project managers are interested in monitoring the number of defects submitted by the different development teams. This group of requirements has been excluded for two reasons. The first reason is that predicting or estimating the human effort (in person-hours) needed for fixing faults is not possible at present. The second reason is, that the other three features, such as monitoring turn-around time defect life cycle and defects submitted teams, are not related to prediction.

A technical debt (TD) group of requirements was identified by project managers. It is also known as a design debt or code debt. TD refers to software source code files that are complex to perform any changes on and complicated to maintain. It is caused when developers that are under development pressure take the easiest path to implementing the features but not the best approach to building them. It is thus more difficult to perform changes on code files. Other causes of TD include lack of collaboration and parallel development. Early TD detection would reduce the amount of maintenance required and decrease the complexity of the code. However, unaddressed TD increases the code complexity and slow implementation of new features, thus perhaps causing delay of the project delivery. One of project managers' main concerns in reducing TD is by predicting the amount of the submitted defects, indicating a technical debt to identify candidates' files or code for redesigning. This group of requirements has been excluded since there is currently no data for it in order to run a calculation.

The requirements changing group of requirements was requested by architectures at the sub-system level. They would like to estimate the number of changes on statement of compliance (SOC) report that is reviewed and updated at the end of the product release cycle and before delivery. The SOC contains all information regarding requirements changing and any additional requirements or features supported by the next product release. Architectures are interested in estimating the amount of requirement changes on the next SOC to be delivered. This group of requirements was excluded since there are no features mapping data to calculate the expected amount of changes for the next release.

The test interdependency group of requirements was identified by architects. Software products pass through five testing phases in a sequential order, namely: design tests, functional verification tests, system verification tests, and network interface virtualization (Figure 6.4). There are different development teams working on each test phase. This indicates that some tests cases could be replicated at different test phases, especially when collaboration between the different teams is insufficient. Running all the test cases takes days for each test phase. Therefore, the architects are interested in observing a correlation or overlapped test cases to prioritize test cases fixing, ensure test coverage, and save time. This group of requirements was not taken into consideration since the architects do not know where the overlap is; thus, there is no data with direct mapping between test cases.

The risk commit and defect analyzing groups of requirements identified were similar in purpose and merged under the name risk commit. On one hand, risk commit was identified by project managers. They wonder whether a single source code commit to the version control system would introduce a risk of identifying more faults into the source code files. Therefore, they would like to predict the probability of having more faults due to a fault fix occurred in a certain source code file when the developers commit changes. On the other hand, defect analyzing was identified by developers at the file level. They would like to predict the probability of detecting other faults when fixing a fault. Additionally, developers are concerned in predicting the number of hidden faults by a certain fault. We found that the purpose of both risk commit and defect analyzing groups of requirements are similar; they were thus merged. The purpose of this group of requirements is not related to prediction visualization but rather coupled with a commit check on a single fix level and not the holistic view. Information visualization is about data mapping and creating patterns, and this group of re-quirements is not about predicting the future and understanding the holistic view but rather predicting a source code single commit at a certain time. The tool stands alone, while this feature needs to be aggregated to the development tools and in real-time development. However, this could be carefully studied for future consideration.

Size grouping group of requirements was requested by software architects. The purpose of this group of requirements is predicting the number of defects affecting each functional areas/requirements (robustness, mobility, stability, and session) of the software. They are interested in mapping each defect to one of the functional areas to understand the quality of each functional area. This group of requirements requires deep analysis and careful thinking before it is considered in future projects as there are no connection between the functional areas/requirements and the physical location of the source code. A participants said, "Let us say a defect is affecting the robustness functional area. However, it's hard to pinpoint or map it physically in the source code were robustness handled. This group of requirements could be postponed to future work since we don't have the data for it today. We could kind of estimate functional area based on files modifications. When a file is modified, it impacts certain test cases which have a particular functional area designated to them. Those files could be grouped and belong them to a specific functional area.

However, we can't make a concrete conclusion out of this since a functional area might span in many files and sub/system. Additionally, assume that we mapped functional area with defects, it's hard to direct them to a devel-oper or a team as you can't pinpoint them physically to a certain source code location".

The API group of requirements was requested by the developers at the file level. They would like to be able to implement API as a public API so they can script API's to test their ideas with the predicted data. Additionally, they would like to have eclipse plug, as it features such as color code and code snippet assist them in API maintenance and identifying risky code or indicating code smell. This group of requirements was not related to data prediction, thus it was not considered for the present study.

The reduce information group of requirements was identified by the developers at the file level. It was linked to the design and presentation of the data. The users wanted to have a simple tool without complex interactions and information that is easy to grasp. This was taken into account during the designing phase of the tool; however, it has been removed from the requirements list because it is not related to data prediction.

6.3 Users' feedback

This section presents the results derived from the evaluation session. The participants mentioned several suggestions for improvements during the evaluation session of the semi-interactive prototype. These improvements were classified into four categories, related to the understandability and the presentation of the data, the interaction with the data, or to the additional features category.

A minimal number of suggestions were related to the understandability of the data presented. This could be because the way the data is presented or the terminology used to describe the data gathered. For example, using the word "weight" describing a percentage value of the input parameters or features of the model was confusing in the causality view. Also, some labels used to describe the purpose and the content of the view were not clear. Therefore, the participants suggested changing the initial name of action buttons from platform to found on platform to better describe the view content.

Moreover, the participants provided some suggestions related to the presentation and organization of the views in the prototype. For instance, they suggested aggregating the phase found and the severity views into one view, since they have almost the same data characteristics, thus reducing the amount of views and complexity of data processing. Additionally, they suggested changing the time scale to only one and three months, while removing the 12 months data scale, as it is less reliable to predict data one year ahead.

Table 6.1: The three user's groups requirements categories gathered during the focus group sessions

	Project managers (System level)	Product owner's/ Architect (Sub-system level)	Designers/Testers (File level)
Causality	-Show input to the model. DT (UT, CT). -Show number of contributors (on system part level)	-Show reasons for peak/change. -Show number of contributors or changes. -Show new customers or features. -Show which measures would reduce fault density more quickly? UT, Refactoring, combination or others.	-Show why a function, file or component has a certain number of predicted defects.
Phase found	-Show defects phase found	-Show internal defects Master branch(FT, ST, DT, NIV) -Show external/customer defects	
Granularity	-Show everything on system level and system part level -Show customer defects prediction related to severity		
TC selection	-Show test case selection(prioritization)	-Show target affected (Platform1, Platform 2, simulator or hardware) -Show horizontal or vertical (System only or platform dependent)	
Error prone & smell			-Show how error prone an area is? For planning how much to test in that area. -Use case: Project manager Show prediction vs. actual on product level -Show smell, top 20 bad files or areas for re-design. -Show component (files), functions sorted after predicted TRs in coming year. Show interesting files that changes defect magic value. Show expected customer defects per area Areas or high level causing problems Show per file defect prediction
Severity	Show severity of the defects	Show severity levels	
Defects planning	-Show quarterly prediction system level and system part level. -Show overall defects planning (defects races with overtime) -When the defects are found (in relation to release cycle)		-Show rate of prediction over time. Are we doing the night tough? Show expected number of new defects are relevant for scrum planning.
Quality/release Assessment	-Predict release quality assessment -Show prediction of number of defects on release branch		-Show threat level per file -Show types of faults that has high likelihood of occurring
Answer code (AC)		-Show differences in AC levels between files and subsystem.	
Output properties		Show defect density per file	

In the design of the prototype, many interactions options were avoided, as the users requested to reduce interaction complexity and the time spent to learn the tool functionalities. However, interaction options suggested by the users were taken into account. During the evaluation session, participants suggested additional interaction options in the defects over time view. They wanted to identify the related sub-systems or source code files by color. Color makes the data stand out and makes it easier for the users' eyes to follow and recognize (Ware, 2004b). Additional detailed on demands options were requested on the treemap view, such as show defective files having particular value range and exclude other files from the treemap view that are out of the specified range.

Additional features were proposed by the participants, such as being able to export the views and aggregate the data of some of the sub-systems for developers who are interested in their own areas only. Also, they suggested adding 'events overlay' to each view. The events are management changes that happened during the development process that might indicate reasons for changes in defects rate in the next period. Some features were not related to prediction; hence, they were excluded as they do not fall within the scope of this thesis. These features encourage developers to work better. An end user who participated in one of the data collection session

said, "It would be great if the tool could show improvement trends so I can feel good about my progress".

Generally, throughout the feedback session, positive remarks were given regarding the final design and data presented with minor changes. Participants mentioned that the time to render the page should be fast and we should keep consistency with the graph choices used. They also noted that having the action button menu on the top of the page helped to get the user attention and that the design of the interface guided actions easily. Having the window frame holding the graphs in the center of the page, below the action button, helped to identify the view. A participant mentioned that he did not request a view for defect phase found during the data collection session. However, he found it useful and it might have an impact to his work. This has also proved that it was a good decision to establish a universal tool to be used by all the user groups from different organizational levels rather than creating an individual tool for each user group. A line manager mentioned that the tool would help estimating the work to be done, which also shows that the purpose of the prediction viewer was perceived. One of the positive pieces of feedback expressed by a participant was that "having tabs on one page is more usable than having each view open in an individual new page." Another positive point that a participant mentioned is that "the action orientation makes a bit more steering what you want to do? By looking at the top action button menu, you can easily see what you can do with the tool".

7
Results

Software companies' main concern is to understand and improve software quality during the development life cycle and to deliver a high-quality product to the end users. They are interested in visualizing predicted/future data regarding the quality during the software underdevelopment for early improvements. In this chapter, the results are presented to answer the research questions of the present study, namely, "How should the use cases gathered from the development team be visualized?" and "What information is needed to build the visualization?". The semi-interactive prototype is presented, which illustrates the stakeholders' needs to enhance development performance and decision-making through data prediction.

7.1 Prediction Viewer

In this section, we present the final design of a software prediction tool that addresses the users' needs.

Information overload is a fundamental issue; companies struggle with the ever-increasing amount of data. Therefore, people's interest in visualization tools has increased in hopes of making these volumes of data accessible. Graphical representations accentuate important properties of the data and relation between different items, which in turn helps viewers to use their visual perception to analyze the data faster. Therefore, the present study aims to enhance software development and process quality by proposing a visualizing technique to represent defects prediction information for better assessment, exploration, and analysis of future faults. This visualization would help development teams to perceive predicted information easily and quickly, such as defects magnitude, the error rate in their software before delivery, and error discovery after product delivery; and to identify areas of improvement and preempt defect discovery to enhance their work performance and quality, thereby improving customer satisfaction.

The final design (Prediction Viewer) idea has many opportunities for improvements and features that were postponed to future work (see Section 8.2). Moreover, many additional requirements could be explored that are unrelated to prediction. However, they might increase awareness of the future status of software or help in making decisions with the predicted information, which could be explored in the future. No usability tests were conducted on this prototype due to time constraints. The focus of this thesis is limited to future data and particularly to software future defects:

what information the developers wish to predict the future and how to visualize this information. In the present study, we only considered user needs that can be predicted and computed with the existing machine learning models (see Section 6.1).

7.2 Prototype

Prediction Viewer is an online tool for employers at Ericsson AB to visualize high volumes of information gathered from different databases, resources, and the results of the standard machine learning model. The present study is intended to find a way to visualize the extracted data from the machine learning model; however, during the preparation of this thesis, we found other machine learning tools that can be used to gather a broader variety of information. Therefore, we took the opportunity to understand the users' needs and the purpose of data prediction in the software field to attempt to create a visualization tool in this sector. Furthermore, there are many ambitious requests in software data prediction that cannot be predicted at present due to the lack of algorithms to run a calculation, such as expected mankind efforts and time needed to fix each defect.

In the following section, I introduce the Prediction Viewer as an online tool that presents the number of past, present, and future defects in the system under development, allowing developers to plan ahead and define a precise software release time. The design below presents approximate values to the actual values of the defects; the prototype was evaluated by the end users and modified based on that evaluation.

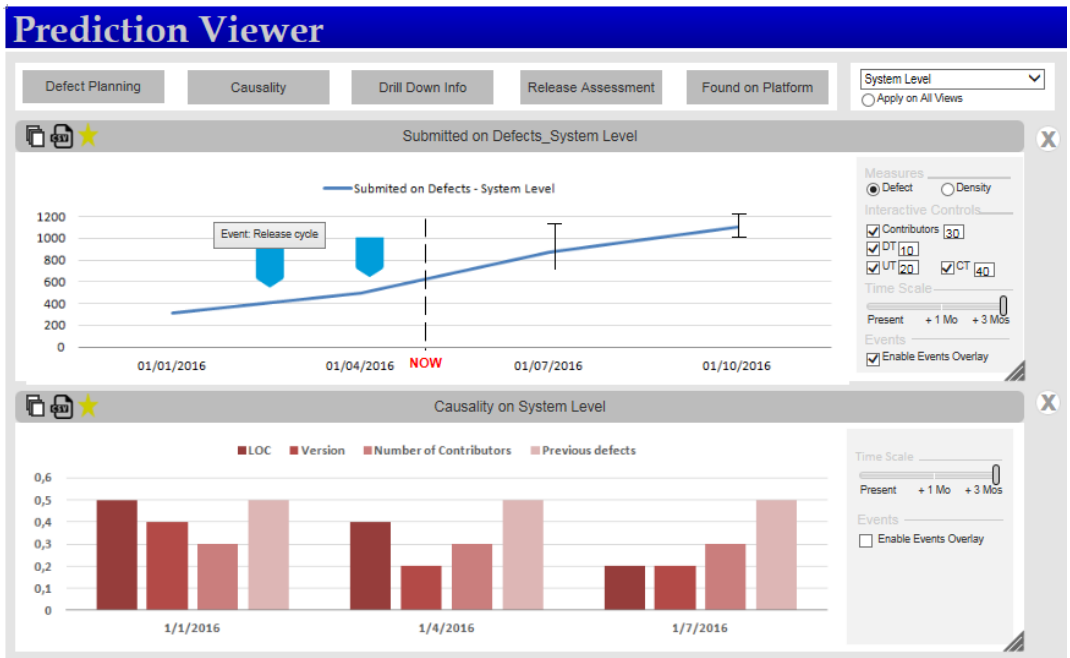

Figure 7.1: The Prediction Viewer main page

The design of the home page consists of two main areas. The first area is the action button menu on the top of the page and the second is the window frame area.

The action button menu consists of two menus. The left-hand side menu, occupies most of the top area, which consists of five action buttons; these represent the main category of the user requirements described in Table 6.1. Each button has a name describing the set of data that will be visualized in that view. The right-hand side menu consists of a drop-down menu, where the users select a sub-system file of interest; data is presented according to the user's selection. All the views visualize defects on the system level (defects found on the entire project) by default. The menu position is constant during the user's interaction with the tool.

The window frame area is located in the center of the tool directly below the action button menu (see Figure 7.1). The window frame area presents a number of different views; each view has different menu options. There are five different windows, one for each action button. Each view displays the data differently using different graphs. A combination of both the name of the view and the selected sub-system file(s) is given on top of each window frame. Additionally, there are three constant icons on the upper left side of the window frame. The first icon from the left is for cloning that particular window frame view. The users can always clone a window and change the datum for comparison purposes. The next icon gives users the ability to download the views in CSV file format as requested during the feedback session. The star icon is to save all the choices that users make during the interactions with the tool for future use. The user does not have to remember the changes or selections they made on the menu of each view each time they visit the page. They can save the changes by pressing the star icon after changes or selections are made. It is possible that users would like to reduce the number of views opened in the view area. Therefore there is an exit button on the upper right-hand side of each window frame to hide that window from the view area. The views can be expanded by dragging and dropping the gray rectangular icon at the right bottom of each view. Moreover, if only one view is visible, then it should occupy all the screen below the top menu. Otherwise, it minimized each time a new view is opened and added to the view area. To open a view the user needs to click on one of the action buttons on the menu at the top of the page.

7.2.1 Time scale

All the views have a constant time scale option and show the data over time (past, present, and future). The vertical line in the middle of the graph shows the defects found at the present time, (see Figure 7.3). As the users have requested, there are two time intervals (one month or three months) that are used to represent data related to defect prediction. Project managers would like to view the amount of defects every three months, which is similar to the development release cycle length. Therefore, the three months time interval was used as the default time scale to represent a set of data on all the views. However, the Scrum users at the file level who have small iterative and incremental development cycles can change the time scale of the views to one month.

7.2.2 Event overlay

The development teams would like to monitor the change management events that happened during software development, such as changes in the version control system, change release cycle, or integration. The events should be presented in correspondence with the time in which they occurred. Events might indicate reasons or changes in the defects rate for the coming months. Vertical tags would appear on the top of the graphs once the user selects the event overlay check box from the menu option on the right-hand side of the view (see Figure 7.3). Each tag represent an event that occurred at a certain time during the development process. There is an event overlay option in all the views.

Two views are always visible by default when users visit the Prediction Viewer pages: causality and defect/density views. These two views were derived from the causality group of requirements as seen in Table 6.1. They are considered significant as they were the most common needs requested by the three user groups (see Figure 7.2). Below is a description of the causality view followed by the defect planning view.

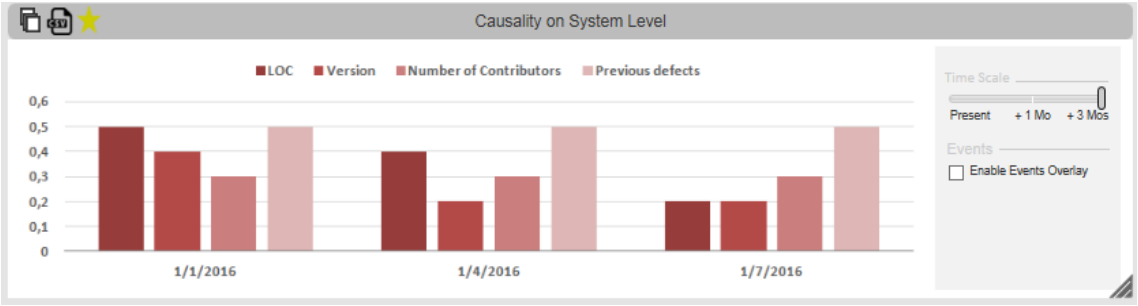

Figure 7.2: Causality view presenting the value of the four defect predictors over time

7.2.3 Causality view

The causality view presents the four predictors variables involved in the machine learning model, which are line of code (LOC), software versions, age, and previous defects. These predictors are used to predict the future number of defects. Additionally, they are perceived as the potential causes of defects. The values of the predictors are the past defects were found in the software development, which is used to predicted number of future defects (Section 2.4). The user groups have come up with other causes of high number of defects (Section 6), some of which are covered in the defect/density view. The causality view presents the different sets of coeffcient values for each model rather than the causes of defects. The bar chart represents the coeffcient value, which is the features going in or needed for training each model. In Figure 7.2, the coeffcient values are shown in every three months, which indicates that three different models are running. Each bar chart individually represents the coeffcient value for each model that predicts every granularity. We therefore are showing the value of the three different equations. The users can view the value of

the predictors in one or three months, so they can correlate the causes of defects from this view to the total amount of defects in the defect/density view (Figure 7.3).

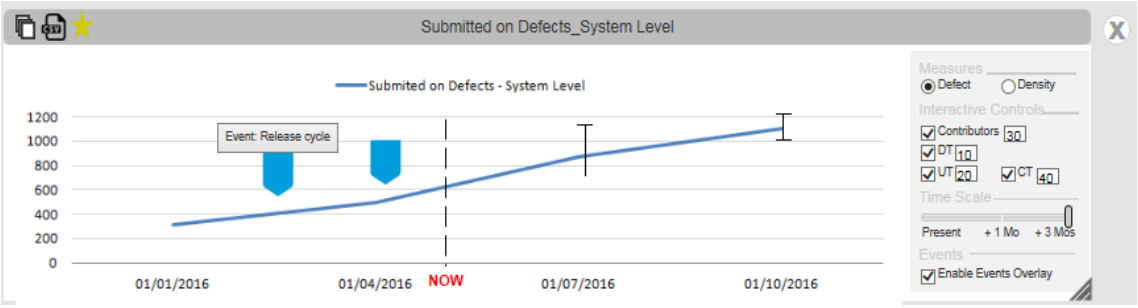

Figure 7.3: Defect/density view presenting the total number of submitted defects over time (Past, present and expected future)

7.2.4 Defect/Density View

The defect/density view consists of two types of graphs: a line chart, as seen in Figure 7.3, and a tree map, as seen in Figure 7.4. On the one hand, the tree map is used to show the quality level of all or several sub-systems/files for a certain period (one or three months). On the order hand, the line chart shows the total number of submitted defects or defects density over a period (all submitted and expected amount of faults) on the system level or for a certain subsystem/file over a specific period of time. Figure 7.3 illustrates the total number of submitted defects on system level during the past and present and the expected number of future defects. Presenting the data using a line graph helps show trends in the number of defects in the past, present, and the expected future clearly. Project managers are interested in viewing trends in the submitted defects and the expected number of future defects over time in order to understand the causes of defect amount increases or decreases and plan ahead to fix defects.

On the right side of the defect/density view line chart, there is a list of interaction options. By default, the line graph represents the total number of submitted defects, since this is substantial information and the most requested need by the three user groups. However, some users are also interested in inspecting defects density (defects per line of code) over a period of time. In this case, the user can select the density radio button from the menu. Additionally, users would like to understand the causes of a high number of defects and how to reduce the incidence of future defects. In this case, they can directly manipulate the line graph by changing the data parameters (Num. of contributors, DT, UT, and CT) and update the graph. The interaction controls on the menu show the present value of the number of contributors (development team), the type, and the number of tests performed. The users can manipulate the view by increasing or decreasing these values and interpreting the causes and the factors of the present and the expected number of defects of a certain sub-system/file. For example, if they have a high number of future defects,

increasing the number of unit tests (UT) could be a way to reduce them, while having many contributors could be a reason for a peak in a number of defects. Data manipulation helps in early understanding of the causes of defects and the factors that help to prevent or reduce them, thus overcoming complex situations at an early stage of the release cycle. In conclusion, data manipulation is a way to understand the factors causing an increase of the number of defects in a certain area in the system.

The future data is not certain as it a prediction result from using past data to predict the future. Therefore it is uncertain; an uncertainty icon s used to alert the observer of this uncertainty.

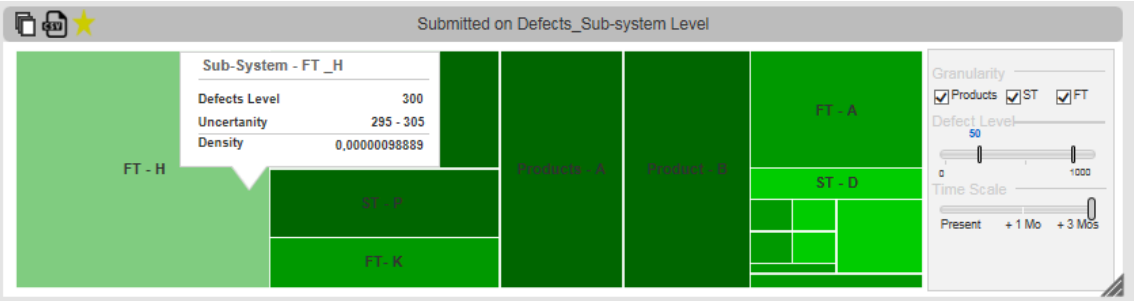

Figure 7.4: Defect\Defect/Density – Tree map view presenting the expected number of future defects of all the sub-systems.

The users would like to view the number of defects or defects density on all sub-systems or all files. In addition, users would like to view the number of defects in each sub-system in relation to defects severity level and in relation to the three system parts. This cannot be presented using a line graph as it requires more dimensions to represent more data types. Moreover, in this case the users are not interested in viewing the defects trends over a period time, but rather in making a comparison between software files. Department managers are usually responsible for several sub-systems within a project. They are involved in the decisions regarding the sub-systems they are responsible for, such as maintaining quality, providing support when problems occur, and managing these sub-systems by providing the needed resources on time. Additionally, developers at the file level would like to understand the quality of all the files within a sub-system they are working on. Thus, they would like to stay informed about the quality of their own sub-systems/files. A tree map graph allows for the representation of large data sets because it can present multiple data dimensions using the colors and size of the rectangles. Therefore, it allows developers at the system, sub-system, and file levels to view an overview, a summary, and the similarities of all the sub-systems or all files of a certain sub-system.

The defect/density view tree map graph is used in the prediction viewer tool so that users can view and compare the quality level of several or all the sub-systems/files. In Figure 7.4, a tree map is used to show all the subsystems on system level or of a product. Each subsystem is represented by a rectangle or a box and the name

of the subsystem is written inside the box. The box size represents only the future amount of defects or defects density of that subsystem, unlike the line graph, which presents data over a period of time. To view detailed information about a particular subsystem/file, users can hover on that particular rectangular box to view detailed information related that particular subsystem/file, such as level of uncertainty, defects density, and number of severe defects. To see the tree map view, the user should select the "all subsystems" or "all files" option from the drop-down menu on the upper-right side of the page, so the type of graph changes from a line graph to a tree map graph. Additionally, since it is not important for users to view all the subsystems over a period of time, the user can only view all the expected future defects of the subsystems. In the case where a user is interested in comparing the defects levels of several subsystems over time, this cannot be done through the tree map view or the line graph view. However, this can be solved by cloning the defect/density line graph view several times, changing the datum, and choosing a certain subsystem for each view from the drop-down menu on the upper right side menu of the main page. This type of graph was also requested many times by developers during data collection sessions and feedback sessions when they proposed their way of inspecting the data. We could have used bar chart but the number of subsystems is very large and each subsystem also has many files.

7.2.5 Drill-down information view

This is the most important view as it combines several groups of requirements (severity, phase found, answer code, and found on platform) with the intent of minimizing the number of options on the main page. It is a complex view because it consists of three sub-views and presents the most relevant data variables. The data set presented in this view has a tree structure. Additionally, these sets of data share the same data characteristics and data structure and could be presented in one view. During the feedback session, the users were not interested in correlating both data of phase found and severity; therefore both data of severity and phase found sub-views are presented in two different views. The bar chart is used in all the sub-views of this window frame.

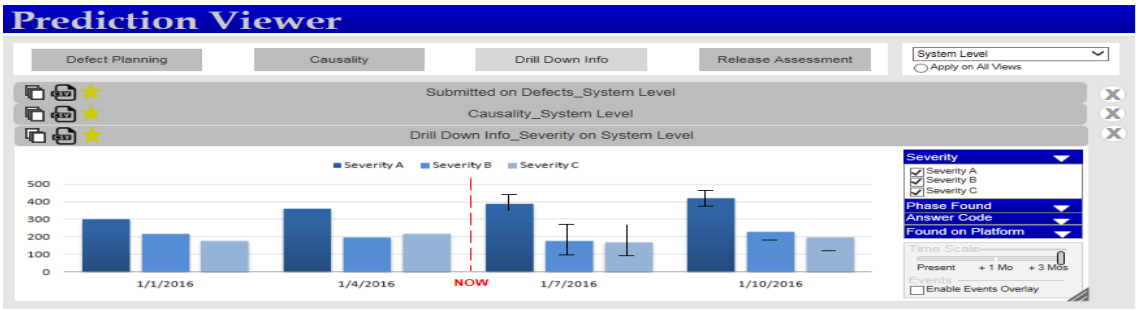

Figure 7.5: Severity view represents the total amount of defects on each severity level

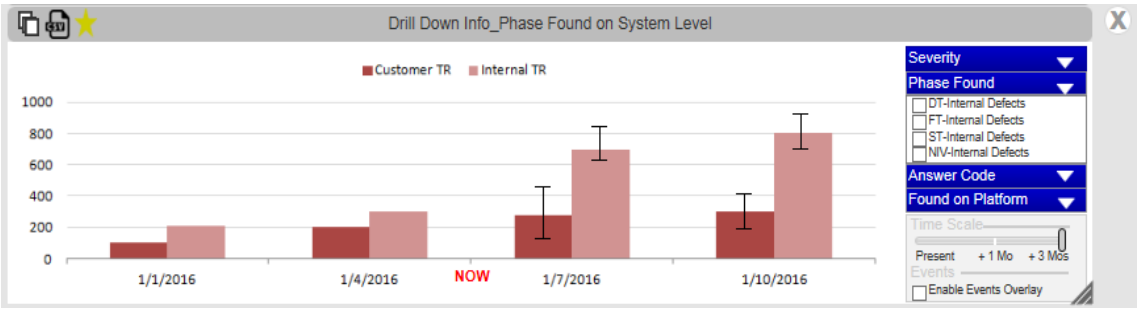

Figure 7.6: The phase found view represents the total number of defects found internally and by the customer.

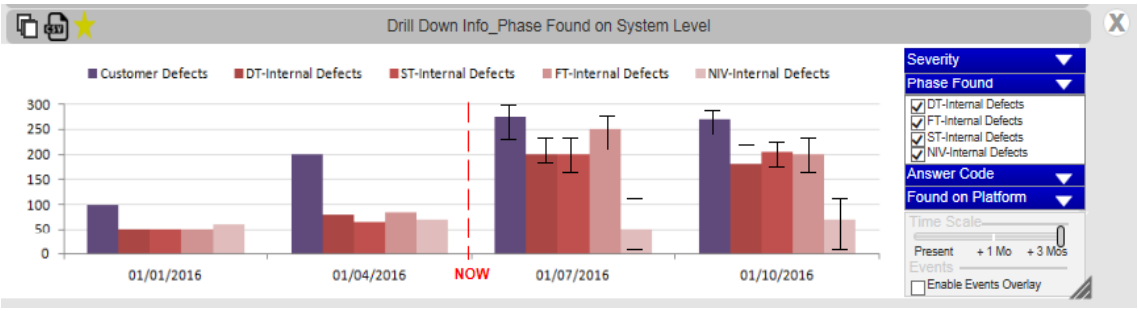

Figure 7.7: The phase found view represents the total number of defects found at each testing phase.

The first sub-view is the severity view; it shows the total number of defects that have a severe level (A, B, or C) (see Figure 7.5). Each bar chart gives the total number of submitted and expected defects on each of the three severity levels. Project managers are interested in monitoring the total number of defects with a severity level of A or B. In this case, they could uncheck the "severity C" box from the severity accordion menu to focus on the information they need.

The second sub-view is the phase found view; it presents an overview of the total number of defects submitted internally and externally(customer), (see Figure 7.6). In addition, the internal defects have sub-categories; these categories indicate the internal testing phase where each defect was discovered internally. Figure 7.7 presents the total number of defects found at each internal testing phase. To see this view, the user checks one or all of the check-boxes (DT, FT, ST, and NIV) from the phase found accordion menu. By default, the phase found view shows the total number of internal and external defects found over a period of time on system level, as in Figure 7.6.

The third sub-view is the answer code view; it represents the three granularities (A, B, and D) of submitted and expected defects (see Figure 7.8). Users can reduce the amount of presented data by removing one of the unnecessary answer code granularity options from the view. Reducing the amount of information in the views would give more space to show more data sets in the view. Moreover, it is more logical to combine the answer code view in the defect/density window frame for a use case,

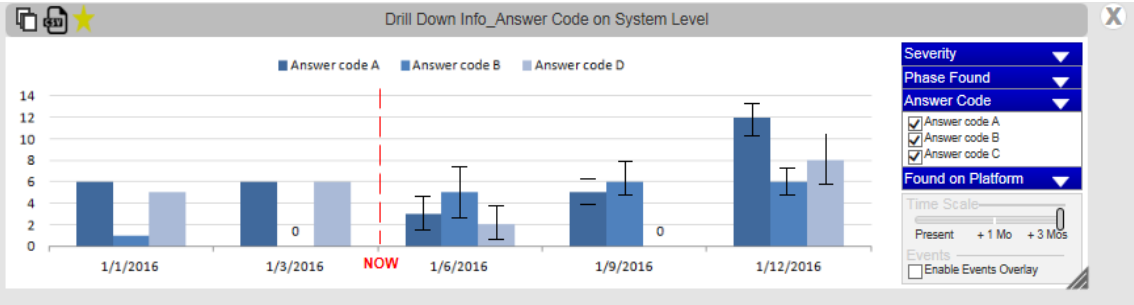

Figure 7.8: The answer code view represents the total number of defects assigned at each level.

such as determining how many defects have been assigned to an answer code from the total number of reported and expected future defects. However, the users agreed to have it as a part of the drill- down view in order to acquire more accurate data. The total number of defects is gathered from the bug tracker, while answer code data is gathered from the version control system. Thus it is not recommended to combine answer code with the same view as the total number of defects. The users want to see whether there is a difference in answer code between subsystems or files. Knowing that there is a misunderstanding of a functionality in one area would indicate the need for improvement. This thus reduces workload in analyzing reported defects that do not lead to something in the system. For instance, they might need to improve documentation or clarify a misunderstanding so that customers do not continue to report similar cases.

The last sub-view is the found on platform view. The defect/density view accumulates all detected and predicted defects of all the products of a department at the company. While found on platform view, it is a kind of drill-down information, which shows the detected and predicted number of defects found on a particular platform (simulation or product) in the next period of time. Figure 7.9 shows two products and two simulations and the number of detected and predicted defects on each platform over time. The phase found and found on platform views are somewhat similar. The phase found view shows the testing phase a fault was found at. It therefore indicates the type of testing that must be improved. However, the found on platform view shows the sort of platform an error was found on.

7.2.6 Release assessment

Ericsson does not submit the whole project at once at the end of the development; rather, they release a set of features continuously. During this development cycle, hundreds of developers work on different parts and features of a product. When it is time to deliver a set of features to the clients, developers regulate the defects on the release branch to condense efforts and allocate time for fixing these errors. To restrict the errors of a release branch, they make a copy of the project parts to be released, called the release branch, so it does not get affected and to isolate the release branch from the mainline development. It is a way to protect the release

branch from being affected by the continuous changes in the development of other features to be delivered in future releases. This view is a copy from the defect/density view line chart (see Figure 7.3). We decided to present it in an independent view for fast accessibility and to not mislead the analysts.

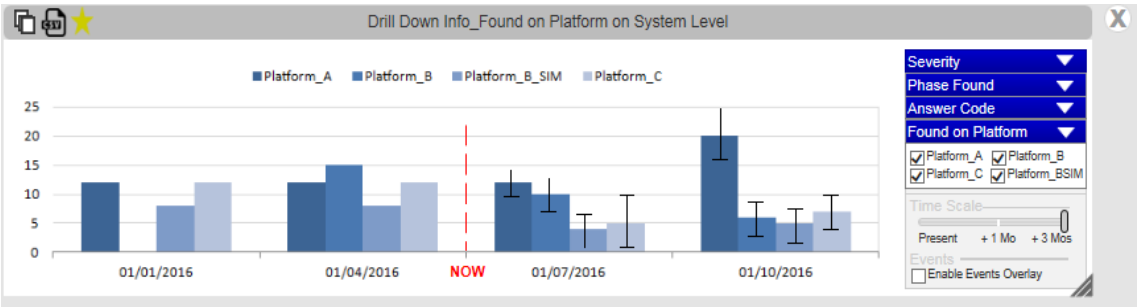

Figure 7.9: Found on platform view

7.3 Design decision

The design of Prediction Viewer was influenced by the users' needs, desires, and experiences with other data analysis tools. Additionally, some design challenges and design requirements guided the decisions regarding different design choices. Some of these challenges were due to the number of data sets and the varieties of data sets properties collected from the three focus group sessions. Few design requirements were elicited from the actual development teams that had experience and design issues with other visualization tools they use. Metrics Viewer is one of the tools frequently discussed and referred to during the three focus group sessions that affected the final design of the present study (Derehag, 2016). It is an open source visualization tool developed and used by Ericsson that presents an analysis of several software metrics measures of software system properties. In the following paragraphs, information regarding the ways in which these obstacles affected and guided the design of Prediction Viewer is presented.

7.3.1 Prediction Viewer

Three focus group sessions were conducted wherein the participants pointed to the usability challenges they face while interacting with some data analysis tools. These challenges were mainly in regard to comprehension, visual design, and interactivity. The tools present many functionalities and a great deal of information, which makes it difficult for users to remember them and their purpose. According to Giacomin (2014), a successful product design has to be intuitive in that it can be manipulated and the information presented can be grasped easily and immediately. Additionally, some interactive tools are not intuitive because they fail to prompt users' interests. Giacomin (2014) suggests we consider human factors/behavior, tasks, and needs

during the designing phase in order to design a usable and beneficial tool. Prediction Viewer was therefore designed based on a deep understanding of the users, their daily tasks and, desires as obtained through the three focus group sessions (in total, six hours). Since it is the first attempt to visualize predicted data extracted from the machine learning model, some of these requirements were excluded to focus on presenting essential information properly and to facilitate the process of designing a visualization of predicted data. Prediction Viewer's visual design presents the main functionalities in the main menu bar on the top of the page with a descriptive name that give an indication of the view content. During the feedback session, the participants appreciated the way the functionalities are presented. It helped them to see an overview of the tool functionality and what actions can be performed.

The participants were interested in understanding the causes of future defects peaks in particular areas of the system under development. The input parameters for the machine learning model are believed to be the causes (see Section 6). Additionally, the users mentioned other causes from their points of view, such as introducing new features and team size. The input parameters cannot be presented in the same view as the other causes mentioned by the focus group participants because the data properties differ. The input parameters for machine learning models are legacy data, so there is no future data and time scale needed. Additionally, the participants were interested in manipulating these causes and view changes on defect magnitude on the graph to get hints and estimate the amount of resources or effort needed. Therefore, the causality view presents the input parameters weighted values (see Figure 7.2), while the other causes were added to the defect/density view because users are interested in manipulating the defect magnitude in order to understand the factors of future defects peaks and plan ahead. During the feedback session, the participants did not understand the purpose of the causality view, especially what the weight on the Y-axis means, weight being the value derived from a differential equation that gives the rate of change of one measure to another needed to train the machine learning algorithm to predict several data sets. Since not all the user groups' members had the chance to attend the feedback session, this view was not removed but rather modified. Instead of displaying the "weight" values of the input parameters, percentage values were displayed for the users who have little knowledge about machine learning algorithm and statistical terms.

In total, 18 groups of requirements were considered in the present study: each group has two to five data features that need to be visualized. An insightful graphical presentation design makes this information meaningful by showing patterns in the data and uncovering hidden information in order to fill in the missing parts of a holistic view. The role of graphical representations is to accentuate important data properties and relationships between different data items, which in turn helps observers to use their own visual perceptions to analyze data more easily and quickly. Information visualization analysis tools have to facilitate searching for data by building patterns, help to form hypotheses, and permitting the detection and perception of unexpected emergent properties (Ware, 2004b). To do so, data visualization tools should support specific users' tasks instead of general visualization (Carr, 1999).

Severity is an example of a complex data set category. It has many sub-categories or information that need to be presented or aggregated differently for each user group to show different perspectives of the current development situation. The drill-down/severity view was designed based on our understanding of the development team's different tasks (task-oriented). Developers and project managers would like to view the predicted defect severity levels (A, B, and C), whereas software architects at component levels are interested in viewing both severities in relationship and the phase found of the defects, which also has some sub-categories that would make the view crammed with information. This implies the need for interaction techniques and data manipulation options. Thus, different user groups can manipulate and prioritize information and create patterns from the data that address their concerns. According to Yi et al. (2007), interaction techniques allow the users to adjust the representation from an overview of general information to more detailed information. Data filters can be used to filter information to only present necessary information that the user eyes can catch quickly, thus reducing the amount of information (Foundation, 2017). The severity view has several interaction filters so that the users can change the level of information presented from the total amount of severe defects on system level to a sub-system or file level. In drill-down phase, in the found view users can change the level of information from overview (internal and customer defects phase found) to more detailed internal phases (DT, FT, ST, and NIV).

7.3.2 Interaction

The stakeholders seek a tool that is easy to use, clear, and does not require effort to learn. The interaction mechanism was necessary to reduce data density, support the users' visual analysis, and to overcome analysis complexity. Additionally, challenges regarding presenting diverse range of data can be overcome and users' cognition can be further amplified (Yi, ah Kang, and Stasko, 2007). Interaction enhances the users' abilities to manipulate the data, allowing for better data interpretation as compared to a static data presentation (Yi et al., 2007). Even so, users emphasized the fact that they do not have time and they want a simple tool where the amount of interaction should be limited. Moreover, they reflected upon their experiences while using a software analytics toolkit (Derehag, 2016) because it is difficult to understand the functionalities provided and the data filter/options; thus, it takes time to figure out how to reach their goals. Therefore the aim is to design Prediction Viewer to promote recognition rather than recalling information by making information and interface function visible and therefore easily accessible. Budiu (2014) mentions the difference between recognition and recalling functionality. Recognition is to recognize familiar objects, while recalling requires information retrieval from memory. Prediction Viewer was designed in such a way as to promote users' recognition rather recalling because they are busy and they need to analyze great volumes of data very quickly. Whitenton (2013) proposes a design guideline to reduce cognitive overload, including reducing the amount of data and following

the human mental model in structuring the data views or the pages of a tool. In the design of Prediction Viewer, the most common information is presented in the default view, while viewing detailed information requires interacting with the tool and manipulating the information.

Furthermore, the structure of the tool and the views follow the existing mental model (top-down structure) of websites design as the guideline suggested by (Whitenton, 2013) and (Storey et al., 1999). The tool consists of two main components/areas: the top menu that contains all the possible information, and the content area that presents the main information the viewers might need to analyze and observe on a regular basis. On the one hand, the menu was designed to be intuitive. The buttons makes all the available commands visible at a glance for the users. On the other hand, the content area presents the main information by default. However, for detailed information, there are several data filtering options allow the user to manipulate the data of each view. It might be time-consuming for the users to manipulate the content in the views each time they interact with the tool and to remember changes in the filter options, especially for those developers who do not use the tool frequently. Avoiding interaction options was not possible given the users' different backgrounds, each of which presents different demands. Thus, the user can customize the tool showing only relevant information he or she needs. For instance, it can be customized to show only the sub-systems/files he or she is interested in. To do so, the tool saves the changes made on the tool for the next visit.

According to Ware (2004), the use of visual images increases data recognition. Despite the visually pleasing aspect and the aesthetic appeal of the icon, icons can be quickly recognized at a glance, making it easier to recall functionalities in complex systems (Harley, 2014). Metaphorical icons aid in reducing the complexity and increasing the usability of the tools. They are used to give the user clues on how a tool works and the expected result of the action performed. An interface metaphor is a visual figure representing data or information that helps in communicating functionality to the users. Moreover, these metaphor icons and interfaces do not require learning; users understand their behaviors from past real-life experiences by comparing them to real-life object. Several icons buttons are used in Prediction Viewer based on this concept, such as an exit button to close, hiding the unnecessary window frame, and PDF icons to indicate downloading and creating PDFs out of the active view. This is intended to help the users remember the functionality of the tool and retrieve information more quickly.

7.3.3 Choice of graphs

Carr (1999) and Diehl (2007) suggest deeply understanding the scope, content, and users' tasks and goals for an effective visualization tool. Since the amount predicted data is very large, a visualization tool is required. Defining the data types assists in choosing the optimal visualization method to represent the data sets (Carr, 1999). The types of the data sets gathered are discrete; however, they have several versions

over time. The presentation should aid and inform analysts and decision-makers from different backgrounds and roles at all the organizational levels. It was therefore challenging to present quantitative categorical data sets, which have diverse data properties, and their visual characteristic (uncertain future data) changes over time.

The data sets gathered through the focus groups have three structural levels: system level, subsystem level, and file level. Each of these data levels should have an overview of the data values and a detailed view at each of the data levels (drill-down data). For instance, a developer would like to view defects on a certain sub-system, while a project manager would like to view defects on all sub-systems to define error prone areas. Additionally, the user should be able to view detailed information, such as defects properties (severity, phase found, and answer code). The overview presentations of a total number of defects and defect properties were not difficult because the number of data sets is limited (see Figure 7.3). However, the main challenge is to present defect properties of 50 to 7000 sub-systems and files, while considering visualization of temporal data, wherein some of the data attributes change over time, affecting the visual presentation of the data. A bar chart, a bubble chart, or a flame graph could present all the data mentioned above. Despite the complexity of tracing and recognizing patterns and comparing future data, all these need to be considered, which has influenced our choice of graphs. When two graphs present information clearly, the graph with fewer interactions was chosen. Decreasing interaction is valuable for developers and managers who want to reduce learnability and effort while interacting with the tool. Additionally, I considered graph types that could present information on the three data levels clearly. In the following section, I present reasoning regarding graph choices used in the Prediction Viewer and why other graphs, such as flame graphs, stacked, and bubble charts, were not considered.

There are different ways to visualize information, including inventing new graphs and using existing well-known graphs. In this study, to reduce the learnability time of the tool and due to time limitations, existing graphs were explored. Numerous 3-dimensional (3-D) or 2-dimensional (2-D) graphs options have been used differently to present information by many existing information visualization tools, such as Tableau (Chabot, Stolte, and Hanrahan, 2003) and D3.js (Bostock, Ogievetsky, and Heer, 2011). Carr (1999) discusses seven visualization methods and their advantages and disadvantages: these are 1-dimentional, 2-dimentional, 3-dimensional, temporal, network, hierarchical, and multi-dimensional methods. In Prediction Viewer, three types of 2-D graphs are used, namely line charts, bar graphs, and tree maps. Three elements influenced the graph choices. First, the ability to present information over time where the data sets' properties vary (legacy data and predicted future data) over time. Secondly, the capacity to plot the uncertainty level of predicted information clearly without confusing the observer, or condensing the view with information. Finally, the ability to compare information and certainty values between files over time quickly.

3-D bar graphs have been found to be complex in presenting and comparing data

magnitudes. Some bars are hidden behind other bars. Rotating the bar graph could be a solution. However, it is time-consuming and cumbersome (Few, 2009). Carr (1999) suggests using a 3-dimensional view only when the data need to be mapped to a physical representation. However, we wanted to study the ability to present error bars of uncertain data using a 3-D bar graph, as it visual attributes allows the encoding of many data sets, such as the height, weight, color, and shape of the bar. The 3-D bar graph becomes more difficult to read when plotting an uncertainty level for each bar representing the number of expected defects in each sub-system/file. The 3-D bars overlap, which make it difficult to compare the reliability values of the data in each sub-system/file. Respectively, one might imagine how complex the view would be when presenting information over a period of time in 3-D bar graphs. It is confusing and not possible without interactions. 3-D bar graph was tested regardless of the other 3-D type of graphs because I expected to present massive data at once over time and solve the issue regarding presenting error bars. However, it was difficult to read when the number of data sets and data properties is large. Moreover, presenting information over time (every three months) was complex for the analysts to trace files over time, identify patterns and compare reliability between sub-systems/files.

The 2-D chart might be too plain and uninteresting. However, it makes more sense than a 3-D graph in terms of presenting predicted information. The following section discusses and presents how different 2-D graphs assists in visualizing predicted data. Two of the following graphs were tested to represent the prediction data based on the end users' and stakeholders' suggestions, namely tree maps and flame graphs. These two graphs were suggested by the end users perhaps because they are used to these types of graphs, so they would like to continue using them. They might also want to avoid the difficulties they have experienced while analyzing data using other visualization tools.

A staked bar graph is another 2-D type of graph which enabled us to combine several data sets and reduce the number of information and views. For instance, it allows us to combine phase found data (internally, externally/customer) with the severity levels data. However, during the evaluation session, we found that stakeholders were not interested in the correlation of these data sets. In this type of Graph error bar to present future data overlaps. Thus, a 2-D bar graph was explored instead.

Both 2-D line graphs and 2-D bar graphs were used to represent information regarding the number of defects and defective areas. Several aspects made these two graphs more suitable for the data sets than other types of graphs. 2-D line and bar graphs easily and clearly present high-level information about the defects levels on system level over time at once. All the information can be seen in the same graph, which facilitates data comparison. The defects values (essential information) are presented without the need for the user to interact with or become confused about available interaction options. The user do not need to interact or to perform an action to analyze or understand the data; they can gain a holistic understanding and easily recognize the relationships, the causes of the future defects, and the fu-

ture situation at a glance (see Figure 7.3 and Figure 7.5). Moreover, the user can perform a comparison between the present and the future situations. However, for low-level and detailed information, the user needs to manipulate the view and the data presented. As I realized, in the SWAT - Software Analytics Toolkit, (Derehag, 2016), the users need to perform an action first to become informed. The vast number interaction options needed before acquiring default information and the amount of information presented on one page increased the complexity of the tool. Thus the user becomes frustrated. The line graph is used to represent the overall number of defects (defect/density view) on the whole system level and for a certain subsystem or file (see Figure 7.3) while a bar graph is used to present low-level information about the defect, such as severity, phase found, and answer code (see Figures 7.5, 7.6, 7.8 and 7.9).

The line graph shows overall and high-level defect information on system level, thus to drill down to the sub-systems/files level, a tree map graph was used to show defects on all sub-systems or files of a certain sub-system at once. The defect/density view is the only view that has two types of graph to present defect and defect density on the three data levels. This is due to the increased amount of data on the sub-systems and files levels. The line graph is only used to present the total number of defects on the system level, so the data sets are limited. A tree map has been chosen to present the total number of defects on sub-system/files levels as it allows the presentation of values of a large number of data sets. The drawback of the tree map graph is in terms of presenting information over time. However, comparing several subsystems or a subsystem of interest over time can be done by cloning the line graph and change the datum to the sub-systems of interest. The user at sub-system and file levels is usually interested in comparing defects on more than one sub-system or file, while managers at the system level are interested in understanding the whole situation and detecting the most defective areas. The tree map shows the overview while comparing several subsystems need cloning line chart number of time.

The tree map provides more data dimensions to represent more data sets as compared to a 3-D bar graph and a flame graph. However, the same challenge was faced while attempting to represent the data using a 3-D bar graph. We found that visualizing uncertainty levels is very difficult using most of the graphs because of two factors: the amount of uncertain data to be presented and the need to visualize the value of the error percentage (certainty level) in future data. Additionally, the reliability level is not a constant number, it is un-known interval value and differs from file to another. Hence, using colors or patterns to encode the uncertainty level was not possible. Thus, in the tree map, visualizing uncertainty is only possible through a pop-up information window that appears when the users hover over a sub-system/file. This might be not the optimal solution since the users need to depend on their memory (recalling rather than recognition) to compare the reliability of several sub-system/files as the window disappears immediately when the mouse leaves.

The flame graph is usually used to visualize activity detection, software executions, and performance profiling. Many papers have discussed the benefits of the flame graph in enhancing the comprehension of software detection activities and Central processing unit(CPU) performance (Gregg, 2016 and Gregg, 2014). For the visualization of data sets and data properties in the present study, the flame graph is not the optimal solution. Presenting large quantities of data using a flame graph is confusing, especially when plotting uncertainty levels for software files that might reach up to 7000 files of a sub-system. Presents the expected amount of defects in software files over time and tracing each line representing a file is difficult. Moreover, comparing data reliability in the files of the future data at once was not possible.

Having a pop-up window showing level of data reliability level would require memorizing and recalling data.Indeed, it is challenging to represent uncertainty level or changes in data set properties in both flame graph and tree map. Interactions might reduce the amount of information, such as data filtering. However, the tree map graph communicates the defect values of a large quantity of files at once. It is much better at grasping the value of defective file than the flame graph, however, in a specific period of time. While it was possible for the flame graph to have a time scale, it was not convenient for tracing thousands of files. It is important to identify defective areas, then trace defect values for both project managers at system level and architects at sub-systems level. A tree map is therefore used in the Prediction Viewer to present information on a specific time period which the user can manipulate.

The bubble graph is the last 2-D graph explored because it allows for the representation of three dimensions of the data. It is known as a sub-type of scatter plot chart. This was also explored with the data sets of this thesis in order to compare its ability to present the data with the other graphs presented above. The bubbles or circles representing a sub-system/file might overlap exactly as in a 3-D bar graph, which introduces complexity in tracking and comparing files over time. Displaying the uncertainty level is only possible by having a hover pop-up information window. Presenting files over time would require more space than screen size. Regardless, the tree map wins this comparison as the rectangular representing the files do not overlap with each other and most defective areas can be easily recognized. The tree map is used to represent defect and defect density on all sub-system and files of a sub-system over a specific period of time, as in Figure (7.4). It is good to show a defect overview of all sub-systems or files of a sub-system before product release. The number of defects could indicate level or amount of work needed.

Finally, future data is difficult to visualize, especially when there is a large number of data sets and different data properties. This was not an issue when presenting information using the 2-D bar and line graphs to present high-level information on system level where the data sets are countable. The uncertainty level or confidence interval differs over time and from file to file. Thus, uncertainty cannot be encoded as color or pattern as it would require encoding values to many different colors. Using a wide range of colors to map numerical values makes the visualization and data

comparison more complex (Harrower, 2003). Lundblad (2015) mentions that the maximum number of color values used should be between 10 to 20 depending on the size of the chart. There are not enough colors to encompass all the possible values (confidence intervals values) and using a gradient color scale is not recommended because people are not able to recognize color variations in lightness and saturation (Ware, 2004a). The confidence intervals indicate the range within which a machine learning result of future data might be inaccurate. However, in this project, the confidence interval could range between any two real numbers, which makes it difficult to perceive the color difference between encoded to the confidence levels.

7.3.4 Time scale

Another element that aids in determining the graph choices is the time scale. During the data collection, we found that the work activity at each system level differs. Some teams work on a weekly basis while others work on a quarterly basis, and few developers were interested in longer term analysis, such as predicting on a yearly basis. The quarterly prediction (every three months) was the common time scale between the three user groups. Therefore, it was chosen to be the default time scale. The time scale is constant in all the views. Predicting on a weekly basis and one year ahead is excluded since the predicted data would be highly unreliable and there is not enough data to predict information on a weekly basis. Predicted data sets need to be visualized over time to compare how legacy data affects future data, thus eliciting high defect levels in a certain area. The comparison should be between legacy and future data on system level, a sub-system/file or between several sub-system/files.

7.4 Prediction Viewer Design

Wang et al. (2000) suggest the use of multiple views to facilitate data inspections and support investigation. There are several advantages of the use of multiple views; it enhances and improves users' performance and the discovery of unexpected data relationships, and it fosters pattern recognitions, data comparisons, and information exploration. Prediction Viewer consists of multiple views to represent different software future data for different users. The defect/density view represents the essential software future data, which are the predicted number of defects. Therefore, it is presented in a separate view, allowing all user groups to manipulate the data and choose the system part to analyze. The release assessment view has the same type of information and data sets; however, it is in a separate view for quicker accessibility. Project managers view it frequently in order to quickly adopt plans and allocate resources. Combining this with the defect/density view would increase the number of interactions options and confuse other user groups that are not interested in release assessment information. Moreover, project managers can easily confuse release assessment view and the main development branch because they look the same, which might lead to incorrect interpretations and conclusions. For these reasons, release assessment information is separated into an individual view.

Since this is an initial attempt to design a software prediction tool, it has been designed to be flexible. Users can select a view and change the datum to allow more data correlation; this is possible in order to address all users' needs. For instance, the users can view general information about the product (system level) and also detailed information of every sub-system and files. Additionally, they can filter the data and focus on certain data sets. The users can clone several views with a different datum to allow more data comparison. When the user uses the tool and know about these important features and data correlations, it helps them see hidden information and determine whether further investigation and improvements can be performed on Prediction Viewer.

7.5 Lessons learned and Suggestions

In this section, I present suggestions and tips for visualizing large uncertain data sets. Additionally, I propose ideas on how to improve the designing phase using focus groups, especially with more than one user group.

There are several elements to consider when designing for large data sets before start the data collection and designing phase.

User environment
Designers should start by investigating the end user environment and take the opportunity to investigate or interact with the different visualization tools the end users are using to perform daily work activities. When the developers complain about other tools they are using, a designer must understand what the user likes or dislikes about these tools and how frequently they use them. Designers ought to elicit tips and create a user suggestion list to guide them during the designing phase so as to not make similar mistakes and frustrate the users. Designers should use the created suggestion list to evaluate the proposed visualization tool against it.

Understanding the number of tools the end users have, the complexity of these tools and how many of them are used help determining whether you need to create a standalone visualization tool or simply an additional functionality added one of the existing ones. Thus, in order to avoid increasing the number of tools and overwhelming the users with an additional tool. Extending an existing tool that the end user is already familiar with would increase the usability of the tool because it will not be forgotten. If the end users have experience with many tools, then they have some design or graph type preferences. To enhance usability, consider users' preferences and experiences with visualizing data.

The tool platform
As mentioned previously, I recommend studying the option of complementing or extending existing tools rather than developing a new one. However, if none exist, under-standing the work environment and flow would help in determining which

platform to consider for the visualization. Prediction Viewer has been designed to complement an existing SWAT - Software Analytics Toolkit used at the company (Derehag, 2016); it was therefore designed as web-based. Also, developers at the company use visualization tools to communicate issues and facilitate discussion during their daily meetings by presenting them on projectors. Hence, mobile application and tabletop applications were not considered. If the tool requires interactions with the data and collaboration with analysts, I suggest a tabletop application.

Graph choices
Thousands of types of graphs are available that might distract the designer's attention at the designing phase. Start by investigating the users preferences graphs, data characteristics, and the purpose of the tools to limit the graph type to be used to the most relevant ones. One way to reduce the number of graph options to the most relevant one is by focusing on a graph dimension (2-D or 3-D). Use the information gathered regarding the data characteristics, user environments, and preferences when making such a decision. Busy developers usually prefer to use the graph types, interaction options, and the tool work flow and information structure they are familiar with. In this way, they do not need to learn how to elicit information or interact with the tool. Choosing a graph dimension makes it easier to pick the most relevant graph that has a data dimension matches your data set's characteristics. Explore these graphs and evaluate them with the end users to ensure user satisfaction. Different data sets require mapping them differently to graph types so that patterns and hidden data are show, which allows for better decision-making. Additionally, avoid using consistent graphs in the tool to present all the data sets; this might mislead the analysts because they look similar, which requires more attention. A consistent graph type used to present all the data might be confusing as they look similar; this might easily confuse the analyst if he or she does not pay careful attention. In this study, I use three types of graphs: tree maps, line charts, and bar charts. Even though the users requested a consistent graph type, data sets and requirements affected the choice of graphs. For example, line charts are not suitable for presenting detailed data compared to tree maps, such as in the case of presenting defect on each software file.

User group(s)
Presenting a large number of data sets over time is a significant challenge. It is a difficult for the designer to know which data sets he or she should begin to visualize and what information is important. Make sure that you allocate time to prioritize requirements with the users during the data collection phase. Usually, prioritizing gathered requirements is the final activity in the data collection session, which means that there is a risk that time could run out and this activity cannot be conducted.

In the present study, it was difficult to control the time of the focus group sessions; there was usually no time left for prioritizing the requirements gathered. If this is the case, use the mapping patterns method and users' stories to determine the most important requirements. If you have recorded the conversation, then you can always look for the most mentioned requirements or challenges from the users' stories in or-

der to begin the design with these. However, allocating 10 minutes at the end of the session to prioritize requirements would save time and ensure that the prioritizing process has not been biased by the designer opinion. To avoid bias in the present study, the requirements prioritization was reviewed by a stakeholder before starting the designing phase.

Additionally, we must consider prioritizing requirements when there is more than one user group. One way is to prioritize the end user group as well. Start small and with the most common requirements of the user groups involved. Once you have an initial design representing the most important requirements, start to add secondary requirements, which are those that are not mentioned by all the user groups. When a requirement is at the same level of importance, look for the one mentioned by the most relevant user group of the tool to be designed. It i's important to have constant feedback during the designing phase. When data sets are plotted, communicate these designs and the design challenge to the end user. Communicating the design challenges is as important as getting feedback on the design. Design challenges determine design decisions and help in reasoning through the design choices; they might be forgotten if they are not communicated to the end user during the design process.

It is recommended to design for broad user groups (Mace, 1997). Make sure to allocate more time for the study when you design for several user groups. It is not always possible to arrange meetings with end users, especially if they have tight schedules which make it difficult to collaborate during the various design stages. One suggestion in such cases is to start by visualizing a few data sets and then expanding them through constant evaluations. Have a few sessions on data collection, collect some data, and continuously design with the end user through data collection/evaluation sessions. Combine evaluation with data collection: evaluate and elicit more needs rather than gathering all the needs and spending time on that without having enough evaluation sessions. In this study, we had only one evaluation session, but there are many things I wished I had the time to evaluate, such as usability and the tool's ability to improve the developer's daily work. Combining data collection with evaluation is a better approach when it is not possible to arrange many sessions with the end users.

Another way to facilitate the designing phase is to elicit and design for the most relevant user group. This design may be used to facilitate the discussion with the other less relevant user groups, elicit more information, and complement the design. In this case, you may evaluate your design as you collect more requirements. Finally, conduct one final evaluation with all the user groups to evaluate the final design.

Having different visualization tools or viewers for each user group is inefficient and might be redundant. My suggestion is to use concept mapping to find the most common needs between the different user groups. These needs should be the default information of the system. Use interaction mechanisms for reveal the low-level information that is specifically needed for a user group and not information common

for the three user groups.

The end users of this thesis work suggest avoiding interactions with the tool. Avoiding the use of interaction options is not the optimal solution and not recommended with big and interconnected data and various user needs, as it will increase the complexity of data interpretation. There are therefore two things to be considered when grouping information in such a situation: first, group the data so that it encompasses hidden data, presents some patterns, and answeres user groups' questions. Many data combinations might uncover interesting information; eliminate these by choosing the data combination that answers most user groups' concerns. Second, when data combination or intended patterns are defined, consider simple interaction options. Even though developers are used to working with complex systems, they are frequently introduced to new tools and do not have the time to learn new tools or how to use or interact with them.

Uncertain data sets:
Presenting future data is difficult. Future defects is uncertain: it represents an estimated value of defects or data sets in the future. Visualizations should warn data analysts that the data is uncertain and the numerical uncertainty value should be clearly presented. The analyst should perceive the uncertain value at a glance to avoid making an incorrect interpretation.

There are not many papers that explore the different ways of visualizing uncertain data and the constraints of each approach (Nathan, 2013). Designers continue to seek the best way to visualize uncertain data with the ability to compare large data sets at a glance. These designers are looking for other ways than the traditional ones, such as box plots and histograms, which show distributions. Many new ideas that have more aesthetic value than box plots, such as the color gradient. Krusz (2013), Propose ideas on how to use the color gradient to show uncertainty for the non-statistician audience. I do not recommend this approach when a data set is very large because it becomes difficult to compare the certainty value between different data sets for the purpose of work prioritization.

Choosing the approach to visualize uncertainty depends on how many data dimensions are to be visualized and the data density to be given on the screen. Consider the other data dimensions that need to be presented when choosing an approach for visualizing uncertainty. For Prediction Viewer, the uncertainty value becomes another dimension of the data. Using an uncertainty bar graph was optimal; it reminds the analysts that the data is uncertain as well as giving them the ability to compare the certainty values of the data sets.

A stacked bar visualization was avoided even though it helps to show more data dimensions and reduce data density on screen. First, it is not able to plot uncertainty levels (error bar) as they overlap. Second, it is not capable of comparing value at a glance. Analysts have to pay attention in order to calculate the value of each stack. This graph type might be useful when the purpose is to compare portions regardless

of the actual value of each stack or when the data sets are limited.

There are many types of graphs. Uncertainty limits graph choices. In the case of future data, I recommend beginning by choosing graphs that are capable of presenting uncertainty. Not all types of graphs can present uncertainty values clearly.

In this thesis, I focus on visualizing the data sets in an intuitive way. Choose the graphs that developers are used to working with and avoid inventing new ones that require learning. I suggest avoiding using many data coding types; instead, use two to three that can be remembered. One way to use color gradients to indicate increasing or declining uncertainty values. For instance, color gradients could be used to show the change of the uncertainty level over time, Figure **??**. When the uncertainty increases in the next month, the error bar becomes red; it becomes green when it becomes more certain.

Timescale:
Consider the time scale when you have large data sets and data dimensions to be presented. Are you presenting only future data or data over time (the past, present, and future)? Make sure that the chosen time scale can present the uncertainty level of the data clearly without condensing the view.

8

Discussion

In this chapter reflections are presented on how the final design of Prediction Viewer answers the research question of this thesis. There is also a discussion of how the final design might influence the development process and the development teams at soft-ware companies. Moreover, reflections on how a human-centered design approach has supported and facilitated the flow of this thesis work are presented.

Prediction Viewer was designed to represent future data in the standard machine learning model (Derehag et al., 2016). However, the user research requires examining different users' needs and their tasks and environments. This was beneficial as the design of the Prediction Viewer was influenced by the user's tasks and environments. Prediction Viewer is web- based because it is intended to be integrated with another analytical tool (Derehag, 2016). Thus, Prediction Viewer is easily adapted to the development teams daily use and integrated into their daily work and tasks. It can therefore be said that Prediction Viewer presents information that was proven to aid software devel-opers in envisioning the future of the software and planning accordingly.

Prediction Viewer is a semi-interactive prototype that was tested with the stakeholders by presenting different scenarios covering the main functionalities of the tool in an open discussion session. However, they were not able to test the functionalities of the tool, and no usability tests were performed. Thus, based on evaluation session with the user groups, we can only prove the users' need of this tool in order to understand future defects and defects properties in managing software delivery time and efficiently allocating resources. Further exploration is needed to evaluate the usability of the tool and how the tool has improved daily work for software engineers.

The tool offers different views that present information regarding future software defects and defects properties. The causality view shows coefficient to the machine learning model rather than the causes of defects. This view might create confusion of the data clearly without condensing the screen, leading the users to false conclusions and frustration. Based on the observation that software developers do not use analysis tools frequently, there is a need for a deep exploration of the visualization of the tool with them to enhance their tool usage and better define how such a tool could be better integrated into their daily work. Event overlay is another feature of the tool that should improve the awareness of the causes of defects; however, it depends on a number of the events presented on the top of a graph. A high number of events might crowd the view, overload the users' cognition and increase the time

necessary to perceive the information.

In this study, three factors affected the choice of graph: the ability to display data over time, the uncertainty level, and user choices and ability to compare the data immediately without the need for complex interaction options. During this study, a total of four encounters with the stakeholders and frequent en-counters with the industrial supervisor took place. Each of the three data collection encounters was concentrated on a user group working at a particular organizational level (system, sub-system, or file level). Data was collected and the needs of each user group were identified. However, engaging the different user groups in a work-shop where each user group works together and then discussing the requirements with the other group would be beneficial in identifying the needs of each user group. I would therefore include more sessions for co-designing and evaluation. While conducting feedback sessions in addition to obtaining user feed-back, more requirements and needs would be gathered. Therefore, when there are difficulties in encountering users, I suggest having fewer data collection sessions and more feedback sessions with the users.

8.1 Reflections on the methods

8.1.1 HCD

This research work took place at Ericsson AB Company, Gothenburg, Sweden. The final result was influenced by the working environment at the company. Ericsson has three organizational structures, while other start-up companies might have different organizational structures. Furthermore, other projects at Ericsson might have different needs or data sets that they would like to predict regarding the type of the product.

HCD's primary concern is to design systems that are perceptually and cognitively intuitive (Giacomin, 2012) by obtaining an understanding of the end user's needs and desires prior to the implementation of an interactive system though communication and interaction with the end user during the design process. HCD design activities focus on understanding users, tasks, and ergonomics. Three focus group sessions were conducted to in order to gain a profound understanding of the development team's daily duties and needs.

IDEO (2015) proposes a field guide approach to HCD, which guided this study. It has three primary phases: inspiration, ideation, and implementation that deeply rein-forces relationships with the end users through immersion in their tasks and environment, engaging the users in the design creation, and obtaining feedback. This approach has guided this thesis on a clear path regarding how to perform the work.

HCD encourages user involvement in all the design phases in order to understand users, tasks, and environments. HCD is described as user-centered evaluation, which implies frequent user involvement with the purpose of constant feedback on the pro-

posed design to efficiently address users' needs. This approach might be difficult to apply in some situations where scheduling frequent encounters with the user is challenging. Large enterprises work in iteration to frequently deliver high-quality products to the end customer, so scheduling meetings takes time.

8.1.2 Semi-structured group interview

Discussion regarding each participant's role during the session takes time, especially when there are more than four participants engaging in the discussion. Additionally, when a participant spends a long time describing in detail his or her role, it increases the chance of others feeling left out. However, individually interviewing 15 participants is time-consuming and not easy to plan since participants have tight schedules and deadlines. To avoid participant boredom, direct questions during the introduction to the participants were asked to give each person gets the chance to speak and encourage them to take part in the discussion.

8.1.3 Prediction model presentation

Presenting the prediction model at the beginning of the session helped in limiting the topic and keeping participants focused on the defect area and the information needed for analyzing defects. However, during the focus group session participants could not stop thinking about the implementation possibilities, and they had many questions regarding the availability of the data needed for training the prediction model. This might keep the participants from mentioning their ideas. I recognized this from the first session with the developers, so I continually reminded the participants to ignore the possibilities of implementation and imagine that the model works perfectly, data is available, and any information can be predicted.

8.1.4 Cluster technique

Cluster technique aims to organize ideas/ and requirements into logical groups and to avoid ambiguity and the possibility of analyzing an overwhelming amount of information. It was not possible to conduct this methods in the three focus group sessions as there was not enough time. If there was a chance to prioritize the clustered requirements during the session, then a comparison of the importance of requirements between each focus group and could be used as the basis of the design of the visualization. There are two ways to solve this issue. The first is to prioritize the requirements by the most common ones, for instance, by asking which are the most mentioned requirements by each user group and which are the common requirements shared among all the users groups.

8.1.5 Break during the sessions

Focus group organizer should make sure to plan for a break in the middle of the session, so the participants can get fresh air and stretch their legs. During the break participants also can have the chance to freely exchange ideas with each other

freely and come up with more requirements. Additionally, break time could be used to compensate for the lost time during the session after securing the participants' permission.

8.2 Future improvements

The main goal of this thesis is to explore how to visualize software-predicted information extracted from the machine learning algorithm. However, this was expanded since the end users requested more such features. The result is a complex tool in terms of implementation which might require a significant budget and time to implement. Therefore, existing open-source visualization tools could be explored to visualize Prediction Viewer as a next step. There is a need to study existing visualization tools and propose an optimal tool that has the needed functionality, interaction mechanisms, and types of graphs matching those used in the design of the Prediction Viewer. Many visualization tools exist could be explored, such as D3 (Chabot et al., 2003 and Tableau (Bostock et al., 2011).

Another possible area of improvement is to explore the excluded requirements from Prediction Viewer (see Section 6.2). This information is believed to add more value and aid in decision-making even though it is not related to data prediction. These features could be further explored.

Many visualization tools are available for the developers and project managers. They were not satisfied with some of these tools as they lack usability. They referred to these visualizations as not easy to use and learn. Usability principles were taken into consideration in the Prediction Viewer tool. However, usability was not the focus of this thesis work. A usability test could be performed on the Prediction Viewer prototype where users try out some scenarios to evaluate its usability and point out the issues regarding usability. However, during the evaluation session for the tool, we received positive feedback regarding its usability.

Many types of graphs were explored and tested against the data sets and the data properties collected in the three focus group sessions. However, there are many other types of graphs that could be explored, such as funnel plots and 3-D graphs. However, I believe they require more interaction and are better suited to other devices such as tablets, table boards, and touch screens. Development teams use desktop systems, and they usually present the tools on projectors during the meetings to facilitate discussions. Three-dimensional graphs would be an obstacle with projectors and require careful consideration. Several papers discuss the benefits of the flame graph in visualizing software code performance, so it requires an in-depth exploration. Conducting a co-designed session would be suitable to envision stakeholders' way of using the graph with predicted data. It might be good for visualizing performance data but not predicted data. There is a need to explore the utilization of the flame graph in predicted data since the end users were interested in this type of graph.

It might be frustrating to the end users that many requirement categories were postponed for future work. However, this was necessary to keep the focus on the functionalists that can be implemented and thus design a realistic tool. Additionally, the postponed requirements help for focusing and creating primary functionalities; future studies may then focus on other information that is not related to data prediction but rather helps to analyze future data and facilitate decision making.

As mentioned, the types of graphs used were affected by the obtained data sets and data properties. Additionally, for the sake of consistency in the presentation of information, types of graphs were chosen based on their ability to present information regarding the three organizational levels (system, sub-system, and file level). Due to time constraints the functional level was not explored. I believe that requirements at the function level might have different data types or data properties that could be explored as well. Additionally, visualizing this information might not fit the chosen types of graphs in the Prediction Viewer.

Finally, project managers were interested in comparing legacy, present, and future defect levels with other departments. The comparison might indicate weakness or strength in the development process or shortages in resources within a department. Sharing experiences between departments might lead to a reduction in defect levels in other departments.

8.3 Threat to validity

This study was conducted at Ericsson AB, where the information gathered from end users for the company use and the final design was influenced by their needs, process, tasks, and organizational structure. Therefore, the final design and the information presented in the Prediction Viewer might require some changes to suit and be used by other software companies with different strategies and processes. The design of Prediction Viewer was conducted by the author of this study and reviewed by the industrial supervisor before presenting it to the stakeholders and users for their feedback. It was difficult to engage the users during the designing phase due to their tight schedule. However, we conducted a feedback session to present several use cases with the semi-interactive tool (Prediction Viewer) to consider their opinion and concerns. In the future, I would engage the stakeholders more through the data analysis and design of the tool by presenting several prototypes and asking them to evaluate and suggest improvements. Thus, instead, I would have data collection sessions with the stakeholders at different organizational levels and two feedback sessions.

8.4 Ethical considerations

All the end users' encounters were recorded on video and all the participants were informed and their permission was acquired before we started the activities. The records were stored in a secure location during the study and will be destroyed

directly after the study ends. I did not ask for names or any personal information that might identify a participant in the thesis report. This study was reviewed by the industrial supervisor at the company to ensure that all the information related to the company is not sensitive. The final prototype was revised and reviewed by the industrial supervisor to ensure that it does not present sensitive information of the company. Prediction Viewer supports the development teams to early predict and expect the amount of work required, thus, base their decisions and allocate needed resources accordingly. Project managers can predict defects early, thereby allocate the needed efforts, resources and avoid risks using prediction viewer. Thus, on the one hand,it can reduce stress of failures and inability to deliver on time. Reducing stress would improve project managers decisions, performance and an increase in efficiency when dealing with clients. However, on the other hand, if an experience user is utilizing the tool it can increase the stress levels of a user due to lack of working with visualizations. The tool is designed to present large amounts of data of the user groups needs, thus to be able to address the needs it requires users interaction to show several aspects of the data and find patterns. However, since the end users asked for limited interaction options, the visualization might also hide other aspects of the meta data that helps in supporting their decision. The data reliability might be an issue that misleads the end users, for example, the data presented could be unreliable due to missing data or the prediction algorithm might generate wrong data. The tool presents massive amount of data, therefore it might be slow in rendering the views, which might result in user frustration.

9
Conclusion

This projects has aimed to visualize software-predicted data to enhance awareness of future defects in software and better asses resources, thereby supporting development teams in everyday work activities and decision-making. The final design (Prediction Viewer) is a semi-interactive web-based prototype that allows development teams to analyze predicted defects.

The research question of this thesis is "How to visualize predicted defects of software source code over time?" has been addressed in this thesis. The main supportive research question, “if we can predict defects, what would the development team use this ability for?”, has been addressed as well. That is, three focus group sessions were conducted to understand user needs and desires in order to answer the main research question. Nineteen groups of needs (use cases) were found through three focus group sessions with 15 software developers and managers at Ericsson AB. The most relevant use case that the three user groups defined was causality. They want to understand the potential causes of defects peaks in the future and prevent. Moreover, it was important for both project managers and architects to analyze the testing phase when a defect will be found (phase found) and the area where most defects are (granularity), and prioritize test cases fixing and improve test coverage (test case selection). Software developers would like to visualize the categories of error prone and code smell to create more test coverage and perform defect fixing planing. Architects were concerned about more drill-down information, such as defect severity and answer code on all system levels to indicate technical debt in the software.

Given the second supportive research question of "how should we visualize the use cases gathered from the development teams?", we should note the development teams use several tools during their daily work activities. Each tool varies in its type of interaction mechanism used and complexity. It might therefore prove difficult for end users to remember the functions provided by the tool and the different ways to manipulate the data. The design of the Prediction Viewer tool should help the users recall the functionality provided by the tool and quickly find the information. In Prediction Viewer, the main functionalities are presented at the top of the page in the form of buttons tabs. After exploring a visualization tool at the company and collecting different scenarios regarding the complexity of the tools. Prediction viewer visualizes the most common information without interaction on the part of the user. Traditional 2-D graphs were used because they are easy to interpret. These graph types also enable defect comparison, because they show uncertainty level of

future defects in software over time as well as comparing legacy defects and predicting defects in software files. Detailed information that is necessary for a specific user group, such as answer code, can be accessed by interacting with the tool and manipulating the data. Since this tool is intended to be used by several user groups, each user selection and data manipulation is saved and retrieved the next time the user uses the tool. The main reason that the Prediction Viewer is a web-based tool is because it is intended to be integrated with another web-based analysis tool.

To be able to visualize these use cases, a third supportive research question was defined, namely "What information is needed to build the visualization?" Firstly, the presentation of information over time should be influenced by the release cycle of the development team. Since three user groups that have participated in the present study, three different release cycles were defined (weekly, quarterly, and yearly). However, predicting information on a sprint release cycle (2 weeks) or a longer release cycle (12 months) might not be reliable. In this study the time scale was chosen based on the main release cycle at the company, which is three months (quarterly). Second, understanding the properties of the data set, such as error bar to represent the uncertain data sets and inform analysts, is crucial to selecting the graph type. Finally, the software file structure must be considered when presenting information. There are three levels of file structures; system level consists of hundred of sub-systems and each sub-system consists of thousands of files.

Prediction Viewer is a concept for a web page-based representation of software predicted data to inform software teams about present and future defects and gain insight on how to improve their processes and better assess resources. The final design was prototyped using the Axure prototyping tool in order to easily communicate the design and the information presented to the end users.

At the beginning of the present study, the aim was to visualize the standard machine learning model used by the company to predict software data. However, it was realized that no empirical research has been published on the defects-related data needed and how this information could positively influence the development team's decisions during the software development process. Additionally, the standard machine learning model produces limited information. Visualizing the information without the end users' (i.e., the development team's) collaboration might not be insightful as they might need certain data combinations and correlations to gain insight. I therefore took a step back to understand the user's needs and desires.

The most important lessons learned when visualizing software predicted defects is to choose a graph type that is able to warn analysts that the data is uncertain and the numerical uncertainty value should be clearly presented. Additionally, they should be able to compare the certainty value between the software source code files. In order for the development teams to prioritize their tasks they should be able to trace defects values on software source code files over time.

Prediction Viewer was evaluated with the end users in the form of a semi-interactive

prototype. Several scenarios were created and presented to evaluate the validity of the information presented and the tool functionalities. The end users were satisfied with the tool and thought that presenting the functions at the top of the pages helped in understanding and recalling what can be done with the tool.

Bibliography

Abuzaid, D. & Titang, S. (2014). The visualization of software quality metrics-systematic literature review.

Andrews, K. (2006). Evaluating information visualisations. In *Proceedings of the 2006 avi workshop on beyond time and errors: Novel evaluation methods for information visualization* (pp. 1–5). ACM.

Axure Software Solutions, I. (2002–2016). Axure rp. Retrieved from https://www.axure.com/

Bostock, M., Ogievetsky, V., & Heer, J. (2011). *D^3 data-driven documents*. IEEE. Retrieved from https://d3js.org/

Carr, D. (1999). Guidelines for designing information visualization applications. In *Ecue'99: 01/12/1999-03/12/1999*.

Chabot, C., Stolte, C., & Hanrahan, P. (2003). *Tableau software*. Retrieved from https://www.tableau.com

D'Ambros, M. & Lanza, M. (2006). Software bugs and evolution: A visual approach to uncover their relationship. In *Software maintenance and reengineering, 2006. csmr 2006. proceedings of the 10th european conference on* (10–pp). IEEE.

Derehag, J. (2016). Sw-analytics toolkit: Software analytics and exploratory visualization, a tool for scraping, visualizing and drawing conclusions out of software data. Retrieved from (Retrieved%20March%2026,%202017%20from%20https://github.com/jderehag/swat)

Derehag, J., Weyuker, E., Ostrand, T., & Sundmark, D. (2016). Transitioning fault prediction models to a new environment. In *Dependable computing conference (edcc), 2016 12th european* (pp. 241–248). IEEE.

Egger, F. N. (2000). Lo-fi vs. hi-fi prototyping: How real does the real thing have to be? In *Teaching hci workshop, ozchi*.

Erdemir, U., Tekin, U., & Buzluca, F. (2011). E-quality: A graph based object oriented software quality visualization tool. In *Visualizing software for understanding and analysis (vissoft), 2011 6th ieee international workshop on* (pp. 1–8). IEEE.

Fenton, N. E. & Neil, M. (1999). A critique of software defect prediction models. *IEEE Transactions on software engineering*, *25*(5), 675–689.

Few, S. (2009). *Now you see it: Simple visualization techniques for quantitative analysis*. Analytics Press.

Forsell, C. (2012). Evaluation in information visualization: Heuristic evaluation. In *Information visualisation (iv), 2012 16th international conference on* (pp. 136–142). IEEE.

Foundation, I. D. (2017). *Information overload, why it matters and how to combat it.* Interaction Design Foundation. Retrieved from (Retrieved%20March%2026, %202017%20from%20https://www.interaction-design.org/literature/article/information-overload-why-it-matters-and-how-to-combat-it)

German, D. M. & Hindle, A. (2006). Visualizing the evolution of software using softchange. *International Journal of Software Engineering and Knowledge Engineering*, *16*(01), 5–21.

Giacomin, J. (2012). What is human centered design? *P&D Design.*

Gregg, B. (2014). Memory leak (and growth) flame graphs. Retrieved from Retrieved%20November%2012,%202016%20from:%20http://www.brendangregg.com/FlameGraphs/memoryflamegraphs.html

Gregg, B. (2016). Cpu flame graphs. Retrieved from Retrieved%20November%2012,%202016%20from:%20http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html

Guidelines, T. (2011). Automatic alignment and analysis of linguistic change - transcription guidelines. (pp. 1–16).

Gulliksen, J., Lantz, A., & Boivie, I. (1999). User centered design in practice-problems and possibilities. *Sweden: Royal Institute of Technology*, *315*, 433.

Gyimothy, T., Ferenc, R., & Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software engineering*, *31*(10), 897–910.

Han, W. J., Jiang, L. X., Zhang, X. Y., & Sun, Y. (2013). A software defect prediction model during the test period. *Applied Mechanics and Materials*, *475*, 1186.

Harley, A. (2014). Icon usability. *Nielsen Norman Group.* Retrieved from https://www.nngroup.com/articles/icon-usability/

Harrower, M. (2003). Representing uncertainty: Does it help people make better decisions. In *Ucgis workshop: Geospatial visualization and knowledge discovery workshop* (pp. 1–7).

IDEO. (2015). The field guide to human-centered design. (1st ed.). IDEO.org.

Jabangwe, R., Petersen, K., & Mite, D. (2013). Visualization of defect inflow and resolution cycles: Before, during and after transfer. In *Software engineering conference (apsec), 2013 20th asia-pacific* (Vol. 1, pp. 289–298). IEEE.

Jones, C. & Bonsignour, O. (2011). *The economics of software quality* (1st ed.). Addison-Wesley Professional.

Kang, Y.-a., Gorg, C., & Stasko, J. (2011). How can visual analytics assist investigative analysis? design implications from an evaluation. *IEEE Transactions on Visualization and Computer Graphics*, *17*(5), 570–583.

Kim, S., Zimmermann, T., Whitehead Jr, E. J., & Zeller, A. (2007). Predicting faults from cached history. In *Proceedings of the 29th international conference on software engineering* (pp. 489–498). IEEE Computer Society.

Kontio, J., Lehtola, L., & Bragge, J. (2004). Using the focus group method in software engineering: Obtaining practitioner and user experiences. In *Empirical software engineering, 2004. isese'04. proceedings. 2004 international symposium on* (pp. 271–280). IEEE.

Li, P. L., Herbsleb, J., Shaw, M., & Robinson, B. (2006). Experiences and results from initiating field defect prediction and product test prioritization efforts

at abb inc. In *Proceedings of the 28th international conference on software engineering* (pp. 413–422). ACM.

LUMA, I. (2012). Innovating for people handbook of human-centered design methods. *LUMA Institute*.

Mace, R. (1997). The 7 principles. Retrieved from (Retrieved%20March%2026, %202017%20from%20http://universaldesign.ie/What-is-Universal-Design/The-7-Principles/)

Maguire, M. (2001). Methods to support human-centred design. *International journal of human-computer studies*, *55*(4), 587–634.

Nathan, Y. (2013). Visualizing uncertainty still unsolved problem. Retrieved from (Retrieved%20March%2026,%202017%20from%20http://flowingdata.com/2013/07/10/visualizing-uncertainty-still-unsolved-problem/)

Norman, D. A. (2006). Logic versus usage: The case for activity-centered design. *interactions*, *13*(6), 45–ff.

Pousman, Z., Stasko, J., & Mateas, M. (2007). Casual information visualization: Depictions of data in everyday life. *IEEE transactions on visualization and computer graphics*, *13*(6).

Rawat, M. S. & Dubey, S. K. (2012). Software defect prediction models for quality improvement: A literature study. *IJCSI International Journal of Computer Science Issues*, *9*(5), 288–296.

Rennekamp, R. A. & Nall, M. A. (2000). Using focus groups in program development and evaluation. *University of Kentucky, College of Agriculture, Lexington. Available at*.

Rohde, M. (2011). Sketching: The visual thinking power tool. *A List Apart*. Retrieved from (Retrieved%20January%2025,%202017%20from%20https://alistapart.com/article/sketching-the-visual-thinking-power-tool)

Sedlmair, M., Isenberg, P., Baur, D., & Butz, A. (2011). Information visualization evaluation in large companies: Challenges, experiences and recommendations. *Information Visualization*, *10*(3), 248–266.

Sefelin, R., Tscheligi, M., & Giller, V. (2003). Paper prototyping-what is it good for?: A comparison of paper-and computer-based low-fidelity prototyping. In *Chi'03 extended abstracts on human factors in computing systems* (pp. 778–779). ACM.

Storey, M., Fracchia, D., & Muller, H. (1999). *Cognitive design elements to support the construction of a mental model during software exploration*. Elsevier.

Storey, M. [Margaret], Čubranić, D., & German, D. M. (2005). On the use of visualization to support awareness of human activities in software development: A survey and a framework. In *Proceedings of the 2005 acm symposium on software visualization* (pp. 193–202). ACM.

Ware, C. (2004a). Information visualization. (2nd ed., Chap. Visual Objects and Data Objects, pp. 180–237).

Ware, C. (2004b). Information visualization. (2nd ed., Chap. color, pp. 97–144).

Whitenton, K. (2013). Minimize cognitive load to maximize usability. nielsen norman group. *Retrieved June*, *25*, 2014. Retrieved from https://www.nngroup.com/articles/minimize-cognitive-load/

Williams, A. (2009). User-centered design, activity-centered design, and goal-directed design: A review of three methods for designing web applications. In *Proceedings of the 27th acm international conference on design of communication* (pp. 1–8). ACM.

Yang, X., Tang, K., & Yao, X. (2015). A learning-to-rank approach to software defect prediction. *IEEE Transactions on Reliability*, *64*(1), 234–246.

Yi, J., ah Kang, Y., & Stasko, J. (2007). *Toward a deeper understanding of the role of interaction in information visualization.* IEEE.

A

Appendix

A.1 Focus groups-semi-structured interviews

A.1.1 Project managers at system level

As a project manager, you're responsible for planning, controlling, adjusting schedules in real time and ensuring software quality. An unexpected number of defects is one of the challenges which might affect the quality, budget and delivery.

- how you prevent these situations from happening?
- When defect discovery initiated during the development process?
- At which phase defects are usually found?
- How you mitigate a high number of defects?
- Do you seek to improve defect discovery process?
- Do you use any software analytical tools?
- What type of information do these tools present?
- How analytical tools affect your daily work and decision making?
- During a program development and especially testing periods problem an unexpected number of defects might rise, how you deal with such as situation?

A.1.2 Architect at sub-system level

You as an architect defines the basic structure of a software and keep maintaining the quality of the software structure during the development process.

- Do you use any software analytical tools?
- What kind of data it present? Was it useful?
- What indicates a technical debt in software artifacts?
- Do you use any analytical tool to perform an analysis during your daily work?
- How important to visualize information regarding software artifacts?
- What type of information is needed for better decision making?
- Mention some issues with that tool, why it used or not used.
- Defects affect product backlog, what is the process for tracing defects?
- How do you decrease defects?

A.1.3 Developer's at file level

Your role as developers is to write and maintain programs functionalities and quality.

- Describe the process of software debugging and testing at the company?
- How do you plan for testing?

- What is the process for repairing and maintaining a software program?
- What is the planning process? How do you prioritize bugs fixing?
- When you write programs, unexpected numbers of defects appears, how this affects the development process? How you mitigate this? Do you have any process to prevent them from happening?
- Have you used any software analytical tools before? What information does it visualize?

A.2 Paper prototype

In this section, the paper prototype used to evaluate the design is presented. This paper prototype was used to simulate the stakeholder needs and evaluate the information and the design with the them.

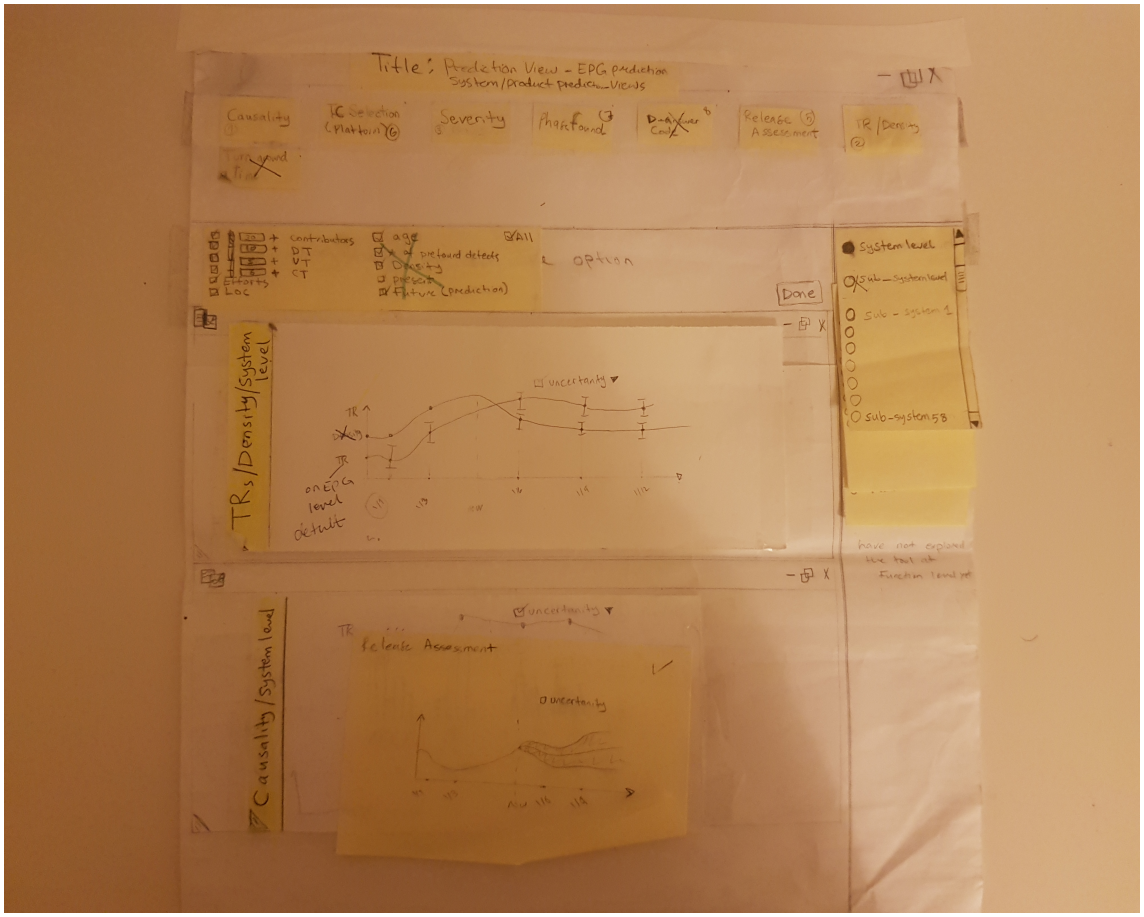

Figure A.1: Paper prototype used to simulate the users needs

A.3 Transcribed document

Can you explain more this point?

Jesper: I have something similar. I call it release assessment or quality assessment. Can you explain more this point?

Patrik: turnaround time. How long time does it take to solve a TR (defect found).

Patrik: more on that granularity—the actual fault, but when aggregating it. I t mean that, oky we have 14 faults within this system part. It will encourage 20 days to fix them. Because that says something about the complexity and that helps us in our planning.

Terese added: it's both, how much effort will be needed to fix a fault and how long lead time to fix a fault. There are not necessary the same. You need a lot of resource but you could fix it in one day or you need to send it to someone to fix something and then you can fix something so it takes time.

Jesper: overall TR planning. Sometimes we would need old way of having TR raises overtime or kind of stopping feature development or whatever.

What do you mean by old way?

Terese: For each fault, I would like to know if it is an old or a new fault. Does it exist in old version?. So how many are new and how many are already exist?

Andrea: Time from commit and fault is found. When each fault was introduced?

Figure A.2: A snap from the transcript document sent to the Project managers