Improving technical debt management and communication by visualizing source code

A concept driven design study to explore the concept of using a visualization tool to improve technical debt management and communication

Axel Ekwall
axelekw@kth.se
KTH Royal Institute of Technology
Stockholm, Sweden

Abstract

With the increasing reliance on digital technology in our lives we, more than ever before, depend on software products and services. The development of such products and services is thereby becoming more complex and sophisticated. In these large software projects, developers often have to make compromises or settle for less than optimal solutions in order to reach deadlines or deliver a product to market in time. This can cause what is known in the industry as *Technical debt*, the accumulation of cost created by "shortcuts" taken during development.

The goal of this paper is to design a concept of a visualization tool to aggregate and present an overview of technical debt in a software development project. Based on this goal, the study tries to answer the following question: *Can a visualization tool help software development teams manage technical debt by improving awareness and communication about technical debt strategy and priorities?*.

A literature study on the topics of technical debt management and information visualization techniques was conducted in order to create a foundation for the work in this thesis.

By leveraging *concept driven design research* [16], a survey was conducted and based on the results, an interactive prototype was developed and evaluated, in order to answer the research question. The prototype was evaluated through critique sessions with semi-structured interviews with expert users working in software development projects. The main findings are that a visualization tool, like the one proposed in this study, could be used to help developers and other stakeholders in software projects raise awareness about technical debt strategies and priorities. However, more research are required in order to refine the tool and visualizations included in the prototype.

Keywords information visualization, technical debt management, concept driven design research

1 Introduction

In recent years society has undergone a digital transformation and our everyday life depends on digital technology more than ever before. This development relies on an ever-growing collection of software systems and services that span our society and connect our lives. With the increasing size and scope of these software projects, and the growing numbers of software developers working in the same project simultaneously, development planning and collaboration becomes harder. A common method to handle these challenges is for software development teams to work according to an agile methodology [10] where requirements and solutions in a project are evolving over time in collaboration between developers, project managers and users. This process is conducted in an iterative cycle with continuous reflections and improvements on previous work.

While this method has improved the ability to quickly develop and release software, the fast and dynamic development process can lead to sub-optimal solutions when designing systems and compromises in the quality of the code. These "shortcuts" accumulate over time, and makes it hard for stakeholders to get an overview of how the project in general, and more specifically the source code, evolves over time. In this context, where rapid continuous decision making and reflection is important, insights and a shared understanding of the current state of the code can be valuable [2].

A useful concept to utilize in order to understand this common issue is *technical debt*. First described by Cunningham in 1992 [7], technical debt is a metaphor to financial debt and describes the common situation with increased development costs over time in software projects caused by poor software engineering practices [17].

However, technical debt is cumbersome to grasp and oversee and therefore often not used in an effective way, partly because it is not aggregated and presented as relevant metrics, but also because it is hidden from non-technical stakeholders who might not be able to retrieve the information from the source code.

One method to gain insights from a large set of information is to visualize it, or in other words, form a mental model of the information in order to understand it better [15]. This practice has been conducted since long before modern technology was available to help and it is thus not necessarily required to use technology to aid in this process [8]. However, the recent trend to collect increasingly larger amounts of data creates new challenges in visualization and manual processing of data might not be an option. In this scenario, the capability of modern technology is a useful tool to help process, filter and map large data sets to visual representations in order to help the user understand the data and form a mental model of the information [4].

1.1 Research question

The goal of this paper is to design a concept of a visualization tool to aggregate and present an overview of technical debt in a software development project. In doing that, his thesis aims to investigate whether an information visualization tool can be used to empower software development teams to make better informed decisions about software architecture and source code maintenance in order to manage technical debt. Based on this goal, this paper will try to answer the following question.

Can a visualization tool help software development teams manage technical debt by improving awareness and communication about technical debt strategy and priorities?

In order to answer the research question, a prototype of a visualization tool will be developed and evaluated based on the following sub-questions:

- Does the prototype present enough information for developers to make decisions about if and when to pay off technical debt?
- Can the prototype help improve awareness about technical debt strategy and priorities within a software development team?

1.2 Delimitations

The main objective of this study is to evaluate a design concept, rather than developing a fully functioning visualization tool. Only the features and interactions necessary to properly evaluate the concept and generate design guidelines will be implemented in the prototype.

2 Related research

To be able to design and develop the prototype of a visualization tool to help manage technical debt, previous work in relevant fields of research was reviewed. This section presents and reviews the research from two main areas, technical debt management and information visualization, upon which this study builds.

2.1 Technical Debt

First described by Cunningham in 1992 [7], technical debt is a metaphor to financial debt and describes the common situation with increased development costs over time in software projects caused by poor software engineering practices [17].

Over time, the concept of technical debt gained in popularity and with that the scope of the metaphor was expanded to include any imperfection in the software lifecycle. With this broader definition of technical debt, the need for strategies to monitor and manage debt in a project was created. Seaman and Guo, in the paper *Measuring and monitoring technical debt*, proposes a "technical debt list" with "debt items" in order to monitor and keep track of instances of debt present in the project [13].

The notion of a "debt item" will be used in this study to represent the debt in a project and present an overview to developers and other users of the proposed tool. The "debt item" described by Seaman and Guo includes a description of where in the system the debt is present and why the task needs to be payed off [13]. In this study the concept is expanded to also include the type, or dimension of debt which the item represents.

The idea of *debt dimensions* was introduced by Tom et. al. in their exploration of the technical debt definition [17]. Through a multivocal literature review of the current technical debt literature as well as interviews with participants from industry, Tom et. al. proposed five distinct dimensions of technical debt; *Code debt*, *Design and architectural debt*, *Environmental debt*, *Knowledge distribution and documentation debt* and *testing debt* [17]. Further, Tom et. al. also explores the factors that contributes to technical debt in projects and among them they identified the following main factors; *Pragmatism*, *Prioritization*, *Process*, *Attitudes* and *Ignorance* [17].

Of special interest to this study are *Prioritization*, *Process* and *Attitudes* since these are factors that are addressable through a visualization tool. Tom et. al. writes, "The visibility of all forms of technical debt decreases when poor communication and collaboration processes are in place, making it easier for debt to accumulate without being noticed." [17] suggesting that visibility and communication are key to establishing a successful technical debt management process.

Worth noting when when dealing with technical debt is that, even though the term in many ways are associated with negative effects, in some cases, taking on technical debt can be a strategic short term decision in order to reach a critical deadline in time. However, if not properly managed the debt can be costly by decreasing development velocity and increasing maintenance costs [14].

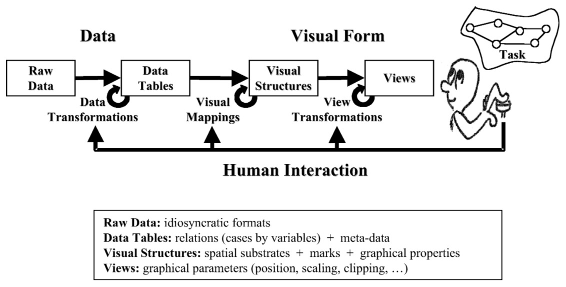

Data
Visual Form
Task
Raw Data
Data Tables
Visual Structures
Views
Data Transformations
Visual Mappings
View Transformations
Human Interaction
Raw Data: idiosyncratic formats
Data Tables: relations (cases by variables) + meta-data
Visual Structures: spatial substrates + marks + graphical properties
Views: graphical parameters (position, scaling, clipping, ...)

Figure 1. Reference model for visualization by Card et al. 1999 [5].

2.2 Information Visualization

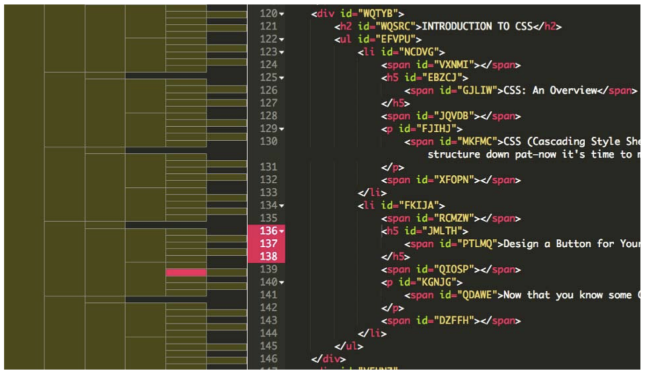
The core of information visualizations are the mapping of data and intent to visual representations. Although there are many different techniques used in the field, this central process of visualizations can be described by the *Reference model for visualization* by Card et al. presented in figure 1 [5]. This common reference model is helpful when analyzing and comparing the different techniques as seen in the taxonomy proposed by Chi based on model by Card et al. [6].

The raw data can take many forms which can be classified into three main categories based on its properties; *nominal data* is simple data with some sort of label but no quantitative value, *ordinal data* can be ranked according to some attribute and *quantitative data* which support arithmetic operations [4].

In this study, the source code of a software project is visualized. The source code is a collection of files and folders that translates into a hierarchical data structure with nodes where each node can contain, or link to, a set of child nodes generating a tree-structure. Hierarchies are closely related to network data structures, with the main difference being that hierarchies does not contain any loops within the connections [15]. Techniques to visualize hierarchies has been a heavily researched topic going far back, and there are multiple approaches and techniques to consider which can be divided into two separate groups, according to a survey of the design space by Schultz et. al. [12]. *Explicit tree visualizations* are visual representations where the relations between nodes are shown with explicit lines or arcs, whereas *Implicit tree visualizations* uses properties such as adjacency and encapsulation to represent the relationship between parent and child nodes in more space efficient way [12].

Further, within the implicit tree visualization category there are two main visualization techniques to consider, *Treemaps* and *Icicle plots*, which both allow for many different layouts in both 2D and 3D. Originating from Venn diagrams, *treemaps* are the most commonly used technique and are constructed by nesting the child nodes inside the parent node using *inclusion* to represent the relationship between nodes. *Icicle plots* improves of the depth aspect by leveraging adjacency to represent relationship between nodes [12].

In a more recent study by Woodburn et al. in 2019, comparing *icicle plots* with *treemaps*, it was concluded that *icicle plots* are preferable by users as they are more intuitive and uses the screen space more efficiently [19]. *Treemaps* was found to result in slower performance and accuracy regarding hierarchy navigation and understanding [19].



```
<div id="WQTYB">
    <h2 id="WQSRC">INTRODUCTION TO CSS</h2>
    <ul id="EFVPU">
        <li id="NCDVG">
            <span id="VXNMI"></span>
            <h5 id="EBZCJ">
                <span id="GJLIW">CSS: An Overview</span>
            </h5>
            <span id="JQVDB"></span>
            <p id="FJIHJ">
                <span id="MKFMC">CSS (Cascading Style She
                    structure down pat-now it's time to
            </p>
            <span id="XFOPN"></span>
        </li>
        <li id="FKIJA">
            <span id="RCMZW"></span>
            <h5 id="JMLTH">
                <span id="PTLMQ">Design a Button for Your
            </h5>
            <span id="QIOSP"></span>
            <p id="KGNJG">
                <span id="QDAWE">Now that you know some
            </p>
            <span id="DZFFH"></span>
        </li>
    </ul>
</div>
```

Figure 2. Tree visualization prototype from Bacher et al. [1].

There are multiple examples of tree visualizations to draw inspiration from. Bacher et al. used both an *isicle plot* and an *circular treemap* to visualize source code in order to support comprehension, see figure 2 [1]. Further, based on the survey of the design space by Schulz et al. [12], Schulz have gathered a collection of tree visualization examples categorized by dimensions of the design space available online at treevis.net [11]. Lastly, there are examples with visualizations of multiple trees in the work by Graham and Kennedy where one possible application area is to compare multiple versions of source code [9].

3 Method

In this section the overall methodology used in this paper is presented. Each step is described in more detail in the next section "Study and results" together with the results.

In order to investigate the research question, a methodology informed by Stoltermans Concept-Driven Design Research [16] was used with four methodological activities listed below.

1. **Concept generation**, related literature study and survey gathering requirements.
2. **Concept exploration** and **design of artifact**, an interactive prototype manifesting the concept.
3. **External critique session**, user study with prepared tasks and interview of target users.
4. **Concept contextualization**, a discussion of the concept and results in this study positioning it against related research.

In contrast to other design driven research methods, where the goal is to gain deep insight into the user behavior in a very specific situation and context, concept driven design research aims to develop more conceptual and theoretical contributions to the general understanding of a use case or a situation [16]. In other design methodologies, a prototype is created to evaluate a specific design solution, whereas in this concept driven study the prototype is used to portray possible future design solutions and work as a probe to understand how users will react to the concept design.

As described by Stolterman, in order for a concept to be valuable it should be based on previous theoretical work [16]. In this study, the **concept generation** is based on the theoretical work on *technical debt* and *information visualization* presented in the previous related work section as well as requirements gathered through a survey with respondents among potential users. The survey was used in order to gather insight into the problems users face as well as their current processes and tools to manage and communicate about technical debt. The survey was conducted online and included an introduction of the topic, a section collecting background information about the respondents and two sections with questions. In the first part, the respondents were presented with a collection of statements about their technical debt in their current project and were asked to evaluate their attitude on a Likert scale [18], and in the second part the respondents were asked a set of open ended questions.

The next phase in this study was the combination of two activities, **Concept exploration** and **design of artifact**. During this phase, the form and functions of the concept was explored and an interactive prototype was designed and developed. The prototype was developed with the purpose of testing new ideas of how to present technical debt in a software development project rather than refining current solutions or evaluating details in a design. As Stolterman writes, "Concept design research does not strive to refine or test established ideas; instead, it explores new territories and design spaces" [16].

When the defined concept and features were implemented in the prototype, an **external critique session** was conducted in order to evaluate the concept manifested by the developed prototype. The sessions were conducted online, one-on-one, with a video conferencing service and all sessions were recorded with video and sound for later analysis. Each session followed a strict agenda starting with an introduction to the research project, concept and prototype read by the researcher followed by a section where the participant was exploring and interacting with the prototype by completing a set of predefined tasks. These steps were conducted mainly to allow the participant to understand and asses the prototype to be able to criticize the concept and the tasks were not designed to evaluate the usability or interaction with the application.

When the participant had completed the tasks and had gotten familiarized with the prototype, a semi-structured interview was conducted. In the first part of the interview, the participants were presented with four statements in order to spark a discussion about the prototype. After each statement they were asked to discus their position regarding the statement. The second part of the interview, the participants were asked a set of open ended questions about the concept in general. After the interview were conducted, the data collected were analyzed using the *thematic analysis* described by Braun and Clarke [3], and a collection of themes and sub-themes were discovered.

The last phase, **concept contextualization**, includes a discussion about concept evaluated in this paper and the knowledge gained, and tries to position it withing the landscape of previous research. This discussion is conducted in the fifth section of this report, and builds upon the findings from the study and results presented in the next section.

4 Study and Result

In this section the study will be described step by step in separate subsections in chronological order. For each step, the method and results will be presented together.

4.1 Survey

The survey was conducted among professionals with various roles in software development projects, with the most common role being "Developer" (90%) followed by "Tech Lead or Project Manager" (60%) and "DevOps" (20%). The number of respondents was 20 with mixed and somewhat evenly distributed level of experience between "less than 5 years" (40%), "between 6 and 10 years" (25%) and "more than 10 years" (35%). All but one (95%) of the respondents were familiar with the term "Technical Debt" indicating that the respondent have a understanding of the concept and thus the required prior knowledge to be able to provide relevant answers in the survey.

In order to understand how technical debt affects the daily work of the respondents and their experience of communicating about and manage technical debt in their current situation, the survey presented nine statements asking the respondents to indicate whether they agree or disagree according to a Likert scale [18]. The results are presented in figure 3, where each statement is assigned a letter and the distribution of the answers for each statement represented with a bar in the figure.

Statements:

a) I find that technical debt is a problem in my current project.
b) Technical debt is often necessary in order to deliver in time.
c) The amount of technical debt in my current project is acceptable.

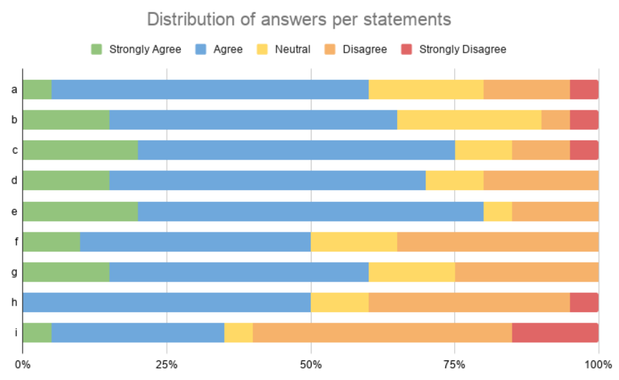


Figure 3. Chart representing the distribution of answers per statement.

d) I have a clear overview of the amount of technical debt in the project.
e) I have a clear overview of where in the project technical debt is present.
f) The technical debt in the project is actively managed.
g) The project team has clear communication about technical debt.
h) The project team has a clear strategy for how to manage technical debt.
i) Everyone in the project team are aware of the technical debt strategy.

The results show that although more than half of the respondents (60%) find technical debt to be a problem in their project, a majority (65%) agree that technical debt is necessary in order to deliver on time. Further, most respondents (75%) consider their current level of debt acceptable. This indicates that developers are facing the need to balance the problems caused by technical debt with the need to deliver in time and that this requires accepting a certain level of technical debt.

A greater part of the respondent indicate that they have a clear overview of both the amount of technical debt (70%) and where the debt is located in the project (80%). However, only half of the respondents work in a project where technical debt is actively managed (50%) or with a clear strategy for technical debt management (50%), and a majority (60%) does not experience that everyone in their team are aware of the technical debt strategy. This points to a problem with an absence of awareness and communication about technical debt in many software development projects.

The second part of the survey included a collection of open ended questions about the respondents experience with technical debt in their daily work. When asked how technical debt impacts their daily work the most common theme was that it leads to a slow down in the development process, which results in longer time to deliver and limits the ability to add new features. One respondent writes:

> "[It] takes longer time to deliver new features, onboard new team members, limits what solutions we can choose for problems." (respondent 17)

Next, the respondents were asked to describe how they currently manage technical debt. The most common answer was that they do it little by little as they stumbled upon it. The "boy scout principle" was mentioned multiple times by different respondents. One respondent explains:

> "Boy scout principle. Leave the code a little better than you found it." (respondent 11)

Another aspect mentioned by multiple respondents is the difficulty managing a balance between paying off technical debt and adding new features as one respondent describes in the following statement:

> "We write stories for it and add them to the backlog. Each sprint we include some such stories - but it's a bit of a tug of war with the business who wants more features instead." (respondent 16)

When asked about what tools or processes the respondents currently are using to manage technical debt most of the respondents did not use any special tool but rather relied on their existing workflow for bug and feature tracking, code reviews and sprint meetings.

Lastly, the respondents were asked what features they would want in a tool designed to help manage technical debt. Among the many different answers there were a few recurring suggestions for features or important aspects of such a tool. Multiple respondents requested a way of understanding the balance between the cost of fixing instances of debt and the potential benefit gained by paying off the debt. An example is the following answer:

> "Wow, good question. Not sure, but I guess it'd be important for me to somehow be able to see how 'expensive' a certain tech debt payoff project would be together with it's potential reward. This would make prioritization between different tech debt payoff projects better." (respondent 18)

Another respondent describes a platform where you can document areas in the code where technical debt is present and keep track of the technical debt in the project:

> "A tool that could help us document part of the code we think has technical dept. A platform where we could have conversation about things and keep track of ideas how to solve things." (respondent 19)

The idea of a tool to help a team schedule and prioritize technical debt also surface multiple times among the answers. One respondent writes:

> "Maybe a 'queue' of topics we have decided is tech debt and for how long they have not been addressed. Otherwise, a solution for the current standards and ways of working with notifications for 'now you have not reviewed this in X months'. Making a tool which proactively helps you schedule tech debt fixing which is open and transparent to both managers and developers." (respondent 15)

4.2 Prototype

Based on the data collected and analyzed from the survey described in the previous section, a prototype of a tool was designed and developed. One of the main takeaways from the survey is that a majority of respondents felt a lack of awareness among their team members. Based on that finding, the goal of the tool is to raise awareness about technical debt among the team members of a development team, about the strategy, decisions and priorities regarding technical debt in a project. Further, the tool also aims to be a common source of information about the technical debt in a project and to be a helpful tool in discussions and decision making.

The prototype was developed as a web application with modern web technologies, React and Redux, using the programming language Typescript. In order to speed up the development process, a component library, *Material-UI*, was used to create the overall structure of the interface. It consists of a main view with a dashboard containing multiple widgets containing information about the technical debt in the project. The widgets are interconnected and responsive to user interaction. In order to facilitate the evaluation, the prototype was developed with static data and no integration with live projects. However, the prototype is coded with real data in mind, and could be integrated with source code repositories with more development. The static data used for the source code visualization was exported from an open source code repository via the GitHub API. This data included the folder hierarchy with all source code files available in the source code repository, and the size and name of each file and folder. The static data set used in the prototype also included a collection of debt items created by the researcher as an example of what real data could look like in the tool, and the participants in the evaluation could add and remove these items in order to alter the data.

Based on the work by Seaman and Guo [13], review in the related work section, technical debt is represented in the prototype as "debt items" which are instances of debt that can be assigned to a location in the source code. Each debt item includes, in addition to the location in the project, also a title, description, deadline, type and priority editable by the user. The debt *types* used in this study are informed by the *dimensions od technical debt* defined by Tom et. al. [17].

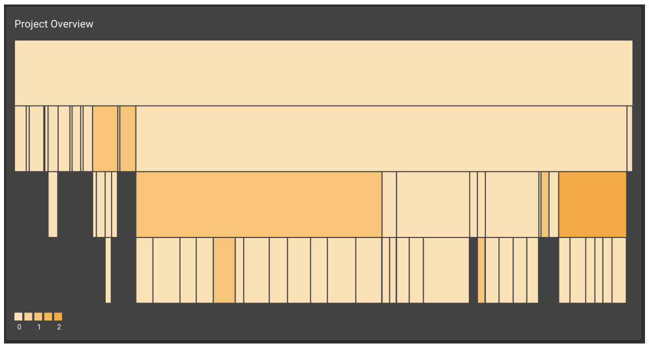


Figure 4. Prototype overview visualization.

The overview widget (figure 4) is a visualization of the source code hierarchical data. Based on the work by Schultz et. al. reviewed in the related work section, two main visualization techniques were considered, *treemaps* and *icicle plots* [12]. A *treemap* would have the advantage of the overall size of the visual representation being decided by the size of the parent node, and thus easy to predict without knowing the size of the data set. However it might decreases the perception of depth in the hierarchy and the parent nodes are often covered by its children nodes resulting in a less clear overview of the data. Since the overview of the project is the main purpose of this visualization, the *icicle plot* was decided to be a better choice since it uses adjacency to represent the relationship between nodes and gives all levels in the hierarchy the same amount of space and presents the whole data structure clearly. It was also concluded by Woodburn et al. to be more intuitive and allow for easier navigation and understanding of the data [19].

Each file and folder in the tree structure is represented as a node, in this case a rectangle. The layout is presented in a top down orientation implying that the top most node contains all nodes below it. A node encodes three pieces of data, the location in the source code represented by the location in relation to other nodes, the size of the file or folder represented by the size of the node and the number of debt items associated with the node represented by the shade of the node. The user can also interact with the visualization, more details are presented on demand when the user hovers over a node, and clicking on a node filters the backlog based on that location in the source code.

The debt type widget (figure 5) visualizes the distribution of debt types among the debt items in the project as a circle chart. Each type corresponding to a distinct color which is recurring in other places in the prototype in order to help the user to find the type of a debt item at a glance.

To present the user with information about the evolution of debt in the project over a span of time, a timeline widget (figure 6) is included in the prototype. The timeline widget

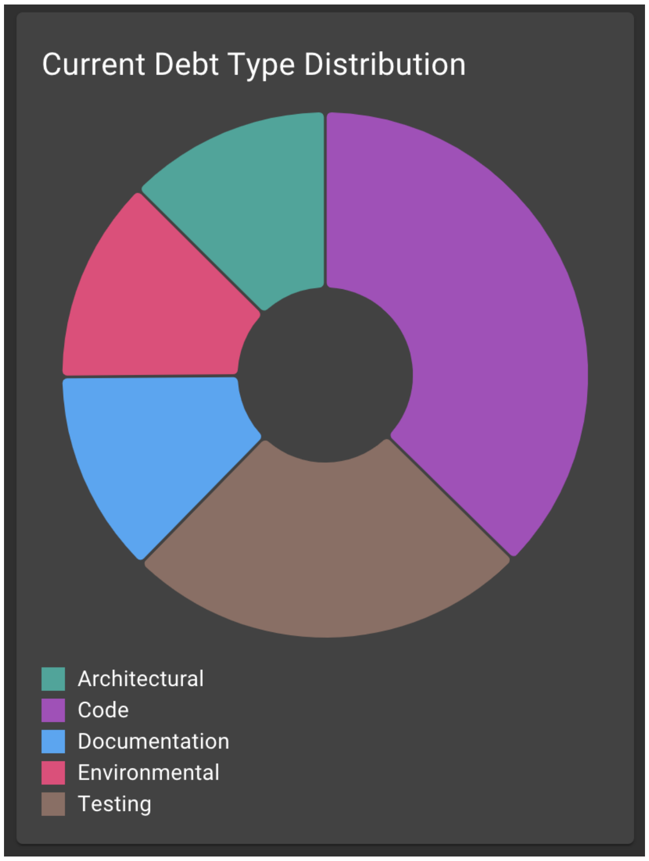


Figure 5. Prototype debt type visualization.

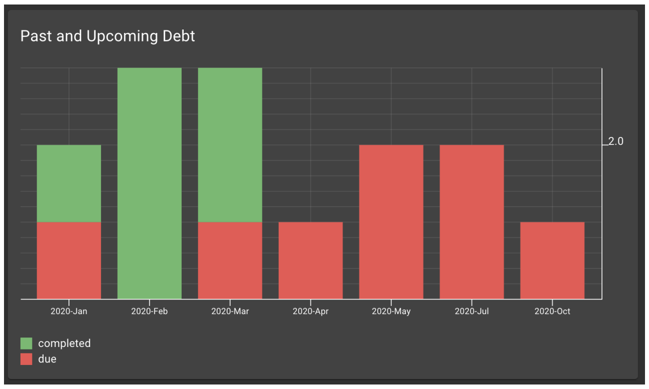


Figure 6. Prototype timeline visualization.

summarizes the number of debt items with deadlines for each month and visualizes this information as bar graph. The bars include both completed (green) and non-completed (red) items colored by state.

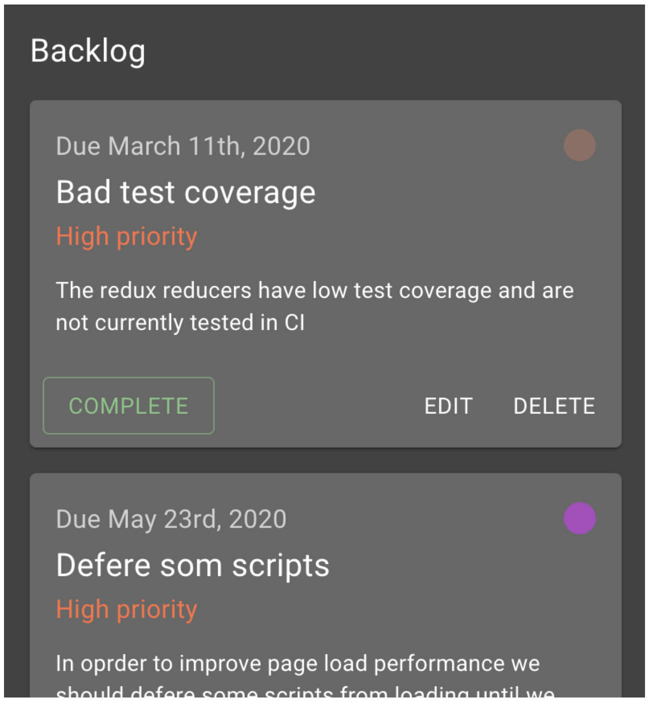
The prototype also includes a column with cards (figure 7) representing debt items, a "backlog" of items for the development team to work on. This is inspired by the approach proposed by Seaman and Guo where debt is presented as a "technical debt list" [13]. The cards are sorted by priority and deadline with the highest priority debt items on top. Each card presents all relevant information about a debt item and three action buttons: "complete", "edit" and "delete". The user can also create a new debt item by clicking the button at the end of the list of cards.

The concept proposed in this study through this prototype aims to help development teams communicate about and manage technical debt by enabling everyone in the team to have an overview of the current technical debt status and priorities. It allows the user to manage "debt items" representing an instance of debt somewhere in the project. The user can create, edit and delete "debt items" which are presented as a backlog for the team to work on. The user can also assign each item a deadline and priority. The goal is for this concept to be a common source of information about the teams strategy and priorities regarding technical debt, and also a help when making decision about technical debt in the project. In other words, a unified interface for all kinds of debt in a project.

4.3 Interviews

The prototype was evaluated through semi-structured interviews with users where the users were able to interact with and explore the interactive prototype.

The participants were recruited from two companies working with software development. Six participants from Mentimeter AB, a product company providing a SaaS solution for interactive presentations and three participants from


Backlog

Due March 11th, 2020

Bad test coverage

High priority

The redux reducers have low test coverage and are not currently tested in CI

Due May 23rd, 2020

Defere som scripts

High priority

In oprder to improve page load performance we should defere some scripts from loading until we

Figure 7. Prototype backlog column.

Prototyp Stockholm AB, a digital agency developing digital products and services for clients. The participants all has a background working in software development projects of varying size, and have roles such as *developer* (9), *tech lead* (7), *designer* (1) and *project manager* (1). Experience among the participants are varying between *up to 5 years* (4), *6-10 years* (4) and *more than 10 years* (1).

In the first part of each interview, after exploring the prototype, the participants were presented with the following four statements in order to spark a discussion about the prototype. After each statement they were asked to discus their position regarding the statement.

1. The prototype provides a good overview of technical debt in the project.
2. The visualizations in the prototype provides useful information.
3. The prototype provides clear information about the priority of debt items.
4. This tool provides the information needed in order to make a decision about when to pay off debt.

Most of the participants (7/9) generally agreed with the first statement. However, multiple participants noted that some types of debt might not map well to the source code and thus not be well presented in the overview visualization.

The second statement received mixed opinions. Almost all participants (8/9) agreed that visualizations in general do add value to the tool and were useful. One participants said:

> "[the prototype] is definitely providing a more valuable interface than what you would get out of an arbitrary project management tool." (participant 6)

However, the specific visualizations in the prototype receive some criticism regarding both the interaction and the presented information. Multiple participants mentioned that the overview visualization (figure 4) was missing labels for the nodes in order to understand it better without having to interact and trigger the tooltips. As with the first statement some participants noted that only visualizing the source files as an overview might be insufficient in order to show all kinds of technical debt, especially for environmental and architectural debt. Regarding the other two visualizations, a majority of the participants wanted to be able to filter the backlog based on type and date in addition to the location by selecting a debt type in the circle chart (figure 5) or a month in the timeline bar chart (figure 6).

Although the majority of the participants (7/9) agree with the third statement about the priority of debt items, many ask for a sorting option for the backlog column (figure 7) in order to make it clear that the column is sorted by priority. In the current version of the prototype without a user selectable sorting option, many users took some time to realize that the backlog is sorted at all. Further, one user points out that it would be valuable to be able to filter out all low and normal priority debt items from both the backlog and visualizations to be able to focus on high priority items in some situations.

The fourth and last statement generated the least agreement (4/9) compared to the previous statements and many participants were reluctant to agree with the statement. The general response was that the decisions about when to pay off debt are complex and requires weighing many different business opportunities against each other and a tool like this simply can not capture all that. With that said, most participants agreed that this tool would be a helpful addition in meetings and discussions about such decisions. One participant explains:

> "Debt doesn't live in it's own little bubble where you can just pay off debt all day, you have to balance it against all the other items in a backlog." (participant 3)

A feature many participants suggested that would help in that regard would be to add an estimated cost of paying off each debt item and the potential benefit of doing so.

After the initial statements, each participants was asked a couple of open ended questions in order to further explore their experience with the prototype and the broader concept of a tool like it.

- Do you think a tool like this would improve communications about technical debt?
- Do you think a tool like this would make it easier to manage technical debt?
- What would be the most important feature of a tool like this?
- In what context would you use a tool like this?
- How would you improve this tool?

The recorded answers and discussions from the open ended questions were analyzed according the the guidelines of *thematic analysis* by Braun and Clarke [3], and the following four themes emerged: (1) *A designated place for technical debt discussions and decisions*, (2) *Flexible visual representation of software project with multiple levels of detail*, (3) *Not another tool* and (4) *Cost and benefit estimates*. The themes captures the recurring opinions voiced by the participants and are summarized below with examples from the transcribes interviews.

(1) Multiple participants (6/9) mentioned the need for *a designated place for technical debt discussions and decisions* and found that the prototype could fill that need. Discussions around technical debt are many times held casually among developers and never recorded or structured and thus hard to follow up or base decisions on. A platform or a tool that encourages these discussions and keep a record of the outcome of such discussions can be a valuable addition in a development teams workflow.

> "I think its good to have somewhere to work structured with technical debt, forcing you to think about it and to make it structured by categorizing and documenting it definitely increases communication about technical debt." (participant 5)

(2) A large group of the participants (7/9) noted that in order to support the large variety in debt types and debt items possible in many project, the tool would need to adapt to multiple levels of detail. Some debt items could require the ability to specify an exact line of code in a source file, while other debt items can span a whole repository of source code or multiple micro services. This requires *flexible visual representation of software project with multiple levels of detail*, as described by one participant:

> "If I've got three projects, maybe there could be a better way of getting an overview of all three projects so that I can manage all of our teams technical debt as a whole rather than just one code base?" (participant 6)

(3) A common opinion among the participants (6/9) was that a key factor to adoption of this tool in their team is how well it can integrate with their current workflow and tools. "*Not another tool*" to add to the already long list of services to keep up to date, unless it can bring enough value in relation to the cost of maintenance and learning. For some of the participants, the solution would be to make it well integrated with their workflow, for example allow them to add new debt items directly from their code editor or use this as a dashboard for an existing issue tracker etc. Other participants would rather have this tool be very simple to the point where it's so easy to use that it won't add much complexity to their process, and keep it separated from their other tools and services. In the words of one participant:

> "That's the thing, you have to keep it simple. It can't reacquire users to do a lot of work. It has to be either automated or really simple to add information." (participants 7)

(4) When asked whether this tool could make it easier to manage technical debt, a recurring theme among the participants (5/9) was the need for a *cost and benefit estimate* from each debt item. Many participants noted that this is a complicated topic and one way of estimating might not translate to every project. However, a way of assigning cost and benefit to items is necessary in order to prioritize and make decision about paying off debt.

5 Discussion

This study aims to improve the managements and communication of technical debt in software development projects by introducing a visualization tool. A concept of the visualization tool was developed using concept driven design research and an interactive prototype of the tool was designed and evaluated in order to answer the question: **Can a visualization tool help software development teams manage technical debt by improving awareness and communication about technical debt strategy and priorities?**

The findings from the study suggests that a visualization tool can improve the communication and management of technical debt, a majority of participants in the study thought a tool like the prototype would improve communication about technical debt and make management easier. However, the study also finds that since technical debt always exists in the broader context of a project and a business with many competing priorities, technical debt management can be complex and can not be isolated from the rest of the development process.

In order to answer the main research question stated above, the prototype was evaluated based on two sub-questions. The first question, *Does the prototype present enough information for developers to make decisions about if and when to pay off technical debt?*, focused on the information provided by the prototype. In general the participants did find the information provided in the prototype adequate, with one main exclusion being a metric of the cost and benefit of paying off each debt item. Some participants did note that this information probably would be hard to define since there is no obvious unit in which to measure this. A suggestion from one participant was to provide a basic scale from 1-5 to assign as a cost vs benefit metric to each debt item and let each development team decide for themselves what this scale implicates. Another similar approach proposed by Seaman and Guo would be to enable a cost-benefit analysis borrowed from financial debt by including *principal* and *interest* estimates for each debt item [14]. This would make it possible to compare and easily prioritize debt items as well as aid in planning, which multiple participants raised as a concern without proper estimates of time to pay off each item.

The second sub-question, *Can the prototype help improve awareness about technical debt strategy and priorities withing a software development team?*, aims to help answer whether the prototype can help teams align on priorities and decisions and improve communication. A theme that emerged from the interview was that without a designated forum and platform to record and structure discussions about technical debt, they tend to happen casually among developers in the day to day work. It is of course not a bad thing to have these daily discussions among developers, but if the communication and decisions about technical debt never gets recorded or formally agreed upon by the team as a whole, it is hard to follow up and keep track of the accumulated debt.

According the the results from this study, a tool like the one proposed by this paper could be both a place to record and structure the teams strategy regarding technical debt, but also a tool to inform the team and make sure that everyone involved in a project are aware of the progress and priorities regarding debt. Many participants said that this would be a very valuable tool to use in sprint planning meetings to update the team on the current status of debt in the project and agree on what to do next regarding technical debt. It was also suggested that the tool could be used as a means of communication between developers and management with regards to budget, new features and long term project planning.

Further, regarding the visualizations presented in the prototype, even though a large majority of the participants did think the visualizations provided valuable information, there are some notable critique of the interface. The debt type and timeline widgets were overall accepted without major objections. However, the overview icicle plot did receive mixed opinions from the participants in the study, most notably that it would not fully capture all aspects and parts of a large software project, since it only represents the source code of a single repository. This, in turn, does make it far less useful since it only provides an overview of a subsection of the possible technical debt in a project. The combination of the breadth of technical debt problems that might arise and the complexity of large software architectures, a single visualization simply will not be enough. One possible solution would be to remove the possibility to assign a debt item to a location in the project, and focus on providing visualization for the other properties of technical debt such as severity, cost and timeline. With that said, the results in this study does show that the idea of providing an overview of the location does add value to the tool, and a better solution would be to explore the possibility to expand the interaction of the overview visualization and add more views and possibly more layers. Since the different debt types operate on different "levels", where for example architectural debt might include several micro services and span multiple code bases and code debt often affects only a couple of lines of code in a specific file, a multi layer visualization could allow for the user to zoom into the visualization and be presented with details for the relevant type of debt on each level. Furthermore, another point noted by the participants were the ability to view multiple source code repositories at the same time to manage debt for multiple projects together. In order to provide this functionality, it would be of interest to investigate the possibility of implementing some of the techniques from the survey of multiple tree visualizations by Graham and Kennedy [9].

5.1 Method Critique

The choice of method for this study came with some compromises due to the limited time frame and scope of the project. Stolterman suggest to include an *internal critique session* among the methodological activities in his description of *concept driven design research* [16]. This activity was not included in this study due to time constraints, and thus it only included a single design iteration. With an additional critique session before the external evaluation, the prototype would most probably been a better representation of the concept and in that also result in a more extensive evaluation of the idea.

Further, a larger number of participants from a broader recruitment population would enable conclusions to be generalized with more confidence. Since the workflow and process can be different in different organizations, to just include participants from two organizations, as this study did, will not capture the whole picture of how the industry works with technical debt. However, since this study was conducted during a global pandemic and most companies, including the two involved in this study, moved to a distributed remote working model, the recruitment of participants were significantly harder. The pandemic also forced the method to include only remote evaluations, and no in person interviews or workshops as initially planned were possible.

5.2 Future Work

This study evaluated a concept of having a tool to manage technical debt by designing a prototype of such a tool and present it to users from a target group. The design of the tool and the interaction supported by it needs to be further developed and evaluated in future studies. Here follows some suggestions of areas of further research.

The visualizations used in this study to represent a project was, according to many of the users in the evaluation, not able to fully capture all the complexity of a large software project. Other visualization techniques do need to be evaluated in future studies to find a suitable representation, or collection of representations, that can fully capture a whole project and present a intuitive visual representation for users. This would be necessary in order for a technical debt tool, like the one proposed in this study, to be able to present the user with a good overview of the magnitude and location of technical debt in a software project.

The filtering and sorting functionality and interaction could be further explored in future work. There are many properties of debt items that can support filtering and sorting and the visualizations needs to support these data changes and visually reflect each state in an intuitive way for the user.

6 Conclusion

This study explores whether technical debt communication and management can be improved by introducing a visualization tool. A *concept driven design research* methodology was used, and an interactive prototype was designed and developed in order to evaluate the visualization tool concept. The prototype was evaluated among users and the findings suggests that a tool, like the one proposed through the concept in this study, would improve communication and awareness about technical debt in software development projects. The potential for improvements in the technical debt representation and overview visualizations are discussed and suggestions for further research are presented withing these areas. In conclusion, this study illustrates the opportunities available to improve technical debt management by leveraging visualization in order to raise awareness and encourage communication about technical debt.

Acknowledgments

I want to thank my supervisor Björn Thureson for all support and good advise he has provided during my work with this thesis. I would also like to thank Niklas Ingvar and Mentimeter for providing supervision and support throughout the project and helping me with ideas and inspiration. Finally, I would like to send a warm thank you to all the participants in the study from Mentimeter and Prototyp as well as everyone who took from their time to respond to the survey.

References

[1] Ivan Bacher, Brian Mac Namee, and John D. Kelleher. 2016. On Using Tree Visualisation Techniques to Support Source Code Comprehension. In *2016 IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, Raleigh, NC, USA, 91–95. https://doi.org/10.1109/VISSOFT.2016.8

[2] Thomas Ball, Jung-Min Kim, Adam A Porter, and Harvey P Siy. 1997. If your version control system could talk, Vol. 11.

[3] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (Jan. 2006), 77–101. https://doi.org/10.1191/1478088706qp063oa

[4] S.K. Card and J. Mackinlay. 1997. The structure of the information visualization design space. In *Proceedings of VIZ '97: Visualization Conference, Information Visualization Symposium and Parallel Rendering Symposium*. IEEE Comput. Soc, Phoenix, AZ, USA, 92–99,. https://doi.org/10.1109/INFVIS.1997.636792

[5] Stuart K Card. 1999. *Readings in information visualization : using vision to think.* San Francisco : Kaufmann, San Francisco.

[6] E.H. Chi. 2000. A taxonomy of visualization techniques using the data state reference model. In *IEEE Symposium on Information Visualization 2000. INFOVIS 2000. Proceedings*. IEEE Comput. Soc, Salt Lake City, UT, USA, 69–75. https://doi.org/10.1109/INFVIS.2000.885092

[7] Ward Cunningham. 1992. The WyCash portfolio management system. (1992), 2. https://doi.org/10.1145/157709.157715

[8] Michael Friendly. 2008. A Brief History of Data Visualization. In *Handbook of Data Visualization*, Chun-houh Chen, Wolfgang Härdle, and Antony Unwin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 15–56. https://doi.org/10.1007/978-3-540-33037-0_2

[9] Martin Graham and Jessie Kennedy. 2010. A Survey of Multiple Tree Visualisation. *Information Visualization* 9, 4 (Dec. 2010), 235–252. https://doi.org/10.1057/ivs.2009.29

[10] Orit Hazzan and Yael Dubinsky. 2014. The Agile Manifesto. In *Agile Anywhere*. Springer International Publishing, 9–14. https://doi.org/10.1007/978-3-319-10157-6_3

[11] Hans-Jorg Schulz. 2011. Treevis.net: A Tree Visualization Reference. *IEEE Computer Graphics and Applications* 31, 6 (Nov. 2011), 11–15. https://doi.org/10.1109/MCG.2011.103

[12] H-J Schulz, S Hadlak, and H Schumann. 2011. The Design Space of Implicit Hierarchy Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics* 17, 4 (April 2011), 393–411. https://doi.org/10.1109/TVCG.2010.79

[13] Carolyn Seaman and Yuepu Guo. 2011. Measuring and Monitoring Technical Debt. In *Advances in Computers*. Vol. 82. Elsevier, 25–46. https://doi.org/10.1016/B978-0-12-385512-1.00002-5

[14] Carolyn Seaman, Yuepu Guo, Nico Zazworka, Forrest Shull, Clemente Izurieta, Yuanfang Cai, and Antonio Vetro. 2012. Using technical debt data in decision making: Potential decision approaches. In *2012 Third International Workshop on Managing Technical Debt (MTD)*. IEEE, Zurich, Switzerland, 45–48. https://doi.org/10.1109/MTD.2012.6225999

[15] Robert Spence. 2014. *Information visualization.* Springer, New York.

[16] Erik Stolterman and Mikael Wiberg. 2010. Concept-Driven Interaction Design Research. *Human–Computer Interaction* 25, 2 (May 2010), 95–118. https://doi.org/10.1080/07370020903586696

[17] Edith Tom, Aybüke Aurum, and Richard Vidgen. 2013. An exploration of technical debt. *Journal of Systems and Software* 86, 6 (June 2013), 1498–1516. https://doi.org/10.1016/j.jss.2012.12.052

[18] Chauncey Wilson. 2013. Questionnaires and Surveys. In *Credible Checklists and Quality Questionnaires*. Elsevier, 29–79. https://doi.org/10.1016/B978-0-12-410392-4.00002-7

[19] Linda Woodburn, Yalong Yang, and Kim Marriott. 2019. Interactive Visualisation of Hierarchical Quantitative Data: An Evaluation. *2019 IEEE Visualization Conference (VIS)* (Oct. 2019), 96–100. https://doi.org/10.1109/VISUAL.2019.8933545 arXiv: 1908.01277.