Evaluation of information visualization as a tool to improve technical debt management and communication

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

ABSTRACT

faucibus scelerisque eleifend donec pretium vulputate sapien nec sagittis aliquam malesuada bibendum arcu vitae elementum curabitur vitae nunc sed velit dignissim sodales ut eu sem integer vitae justo eget magna fermentum iaculis eu non diam phasellus vestibulum lorem sed risus ultricies tristique nulla aliquet enim tortor at auctor urna nunc id cursus metus aliquam eleifend mi in nulla posuere sollicitudin aliquam ultrices sagittis orci a scelerisque purus semper eget duis at tellus at urna condimentum mattis pellentesque id nibh tortor id aliquet lectus proin nibh nisl condimentum id venenatis a condimentum vitae sapien pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas sed tempus urna et pharetra pharetra massa massa ultricies mi quis hendrerit dolor magna eget est lorem ipsum dolor sit amet consectetur adipiscing elit pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas integer eget aliquet nibh praesent tristique magna sit amet purus gravida quis blandit turpis cursus in hac habitasse platea dictumst quisque sagittis purus sit amet volutpat consequat mauris nunc congue nisi vitae suscipit tellus mauris a diam maecenas sed enim ut sem viverra aliquet eget sit amet tellus cras adipiscing enim eu turpis egestas pretium aenean pharetra.

KEYWORDS

information visualization, technical debt

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. //SOURCE A common method to handle these challenges is for software development teams to work according to an agile methodology [7] 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.

In order to handle these iterative and fast paced change to the source code in a project, development teams are using version control systems to keep track of changes. This allows developers to work on different versions of the software in parallel and *commit* their changes to the project in batches when new features are completed or bugs are fixed. All these small incremental changes are recorded and make up a rich source of information into the evolution of the code. However, this information 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 VCS system. //SOURCE This 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 into the evolution of the source code and a shared understanding of the current state of the code can be valuable. [1]

Information visualization

One method to gain knowledge from a large set of complex information is to visualize it, or in other words, form a mental model of the information in order to understand it better. [10] 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. [6] However, the trend in recent years of collecting 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 to understand the data and form a mental model of the information. [2] Digital visualization tools can also enable rich interaction and allow

a user to explore the data step by step rather than getting overwhelmed by too much information at once. //SOURCE

Technical debt

When exploring source code evolution, and problems that commonly occur in long term software projects, a useful concept to utilize is *technical debt*. First described by Cunningham in 1992 [5], technical debt (TB) 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. [12] In some cases, taking on TB 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. [9]

Research question

TODO: Update this section with the new focus on technical debt management rather than visualization.

The goal of this paper is to design a visualization tool to explore the source code evolution in a software project in order to provide an overview and understanding of technical debt and how it is changing over time with the evolution of the software and source code.

In doing that, his thesis aims to investigate whether information visualization can be used to empower software development teams to make better informed decisions about software architecture and source code maintenance on order to manage technical debt. Based on this goal, a prototype of such an information visualization tool will be created to answer the following research question.

Can a visualization tool help software development teams manage technical debt by informing decision regarding maintenance and refactoring?

In order to answer the research question, the prototype will be evaluated with the following sub-questions:

- Does the prototype present enough information for developers to make decisions about whether to refactor and/or break up specific parts of the source code in order to pay of technical debt?
- Does the prototype present the information required in order to identify technical debt by highlighting “hotspots patterns” [8] in the source code?

Delimitations

TODO: This has to be updated based on the actual prototype and its limitations.

The main objective of this study is to evaluate a design concept, not 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. Further, the visualizations included in the design concept will be based on the data available from commits to a git repository with source code. In order to keep the scope focused, the source code itself will not be analyzed in this project, even though this would be an interesting topic for a complimentary study.

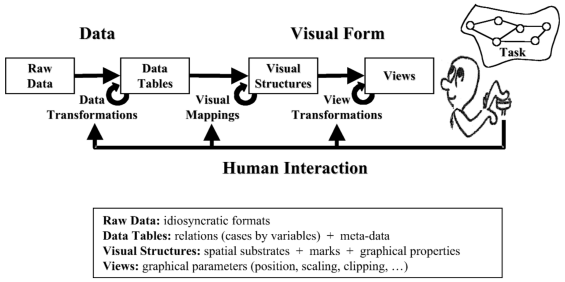

Figure 1: Reference model for visualization by Card et al. 1999. [3]

2 RELATED RESEARCH

This section presents the research upon which this study builds.

TODO: Update this section with more focus on technical debt rather than 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. [3] This common reference model is helpful in order to analyze and compare the different techniques as seen in the taxonomy proposed by Chi based on model by Card et al. [4]

The raw data can take many forms which can be classified intro 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. [2]

Previous research has been published investigating software evolution through different visualization techniques.

Bayer and Hassan developed "Evolution Storyboards"

, examples include code-swarm, CVSscan and Software evolution storylines. However, during the last decade tools and workflows used by software development teams have evolved and more information is recorded with every change, allowing for rich analyses of the history and evolution of software source code.

The previous work mentioned contributed with valuable knowledge and in doing that also created new questions

to be answered. In comparison to CVSscan which focuses on single files, this thesis will investigate a whole repository of source code to give a better overview and richer understanding about how the whole projects evolves. codeswarm and Software evolution storylines presented intuitive visual mappings for data points but targeted casual users and did not allow for interaction and deeper exploration of the data. In contrast, this study targets advanced users with domain knowledge and information about the context of the software project, allowing for a more advanced interactive visualization tool that presents complex information.

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.

TODO: Update this section to better reflect the actual method after changes because of covid-19.

In order to investigate the research question, a methodology informed by Stoltermans Concept-Driven Design Research was used. [11] Two design iterations was conducted in 6 steps listed below over a period of 10 weeks.

First iteration:

(1) Generate concept.
(2) Explore design space and generate low-fi sketches.
(3) First critique session, focus group.

Second iteration:

(1) Develop design artefact, an interactive prototype of the concept.
(2) Second critique session, user study with prepared tasks and interview.
(3) Concept contextualization and design refinement.

Survey

TODO: Rewrite this part

The critique sessions was conducted with participants form the target group; developers, designers and project managers at Mentimeter AB. The critique session during the first iteration will be a focus group workshop where the participants will be presented with the concept idea and low-fi prototypes and sketches created during the first two steps. This session will be structured like group interview and workshop where the goal is to get early feedback on the overall concept design and help in choosing a direction for the second integration.

Prototype

Based on the gathered feedback from the survey, an interactive prototype of a tool to manage technical debt was developed.

TODO: Describe the prototype and the decisions made during its development. Also the shortcomings of the prototype and what mas left out.

Interviews

When the defined concept and features were implemented in the prototype, interviews were conducted in order to evaluate the concept and high level functionalities presented with prototype. This session aimed to gather more specific critique and was structured as task- based user tests in a one-on-one setting where participants were asked to complete a set of tasks with the interactive prototype, followed by a semi-structured interview.

TODO: Describe this more thoroughly.

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.

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. The results are presented in figure 2, 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.
d) I have a clear overview of the amount of technical debt in the project.

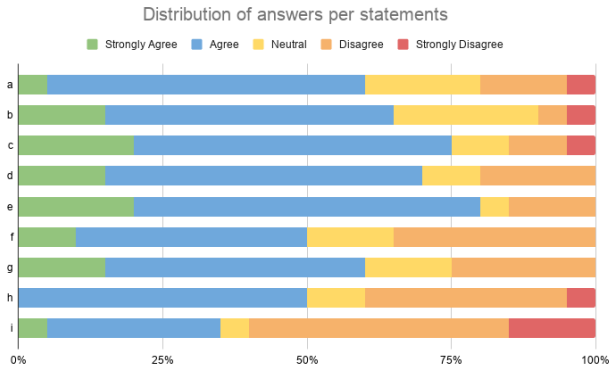

Figure 2: Chart representing the distribution of answers per statement.

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 respondent (60%) find technical debt to be a problem in their project, a majority (65%) agree that technical debt is necessary in order to deliver. 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, 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 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 as technical dept. A platform that 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)

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. The goal of the tool is to raise awareness 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 development as a web application with modern web technologies. 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.

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.

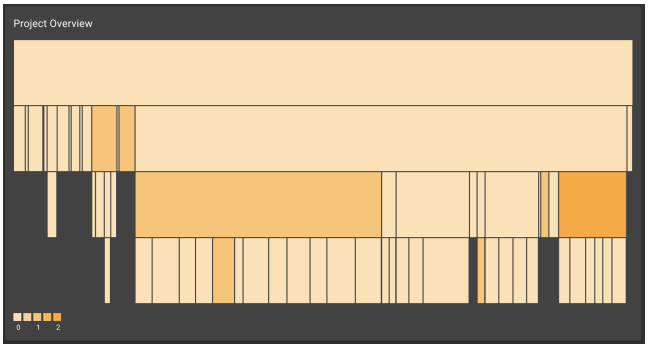

Figure 3: The overview visualization in the prototype.

The overview widget (figure 3) is a visualization of the source code represented as a partition layout. 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.

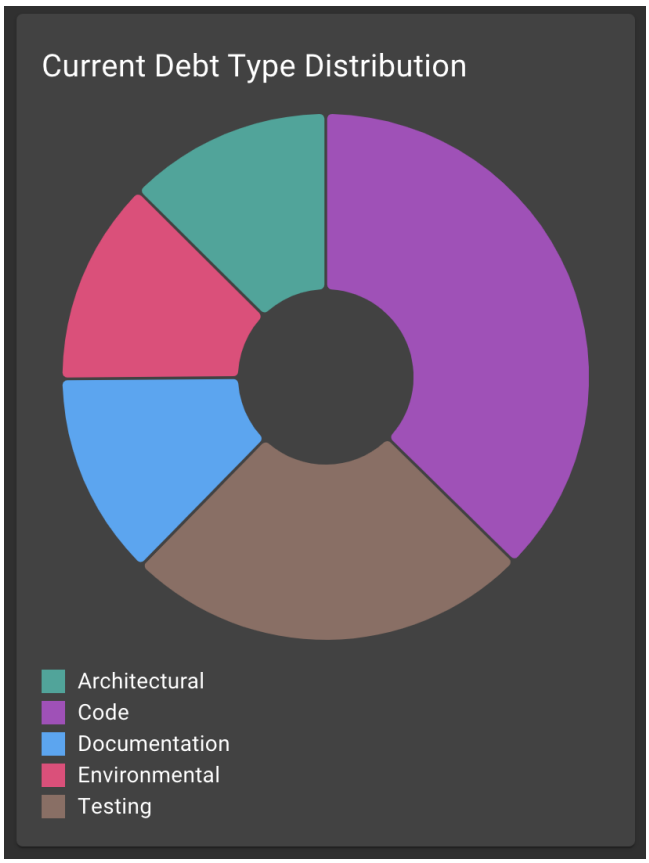

Figure 4: The debt type visualization in the prototype.

The debt type widget (figure 4) 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 5) is included in the prototype. The timeline widget 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.

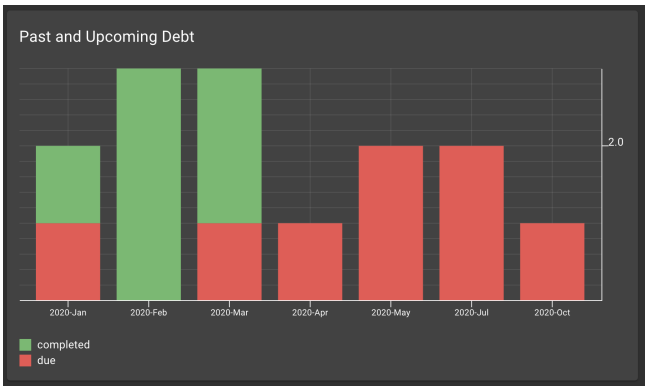

Figure 5: The history visualization in the prototype.

The prototype also includes a column with cards (figure 6) representing debt items, a "backlog" of items for the development team to work on. 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.

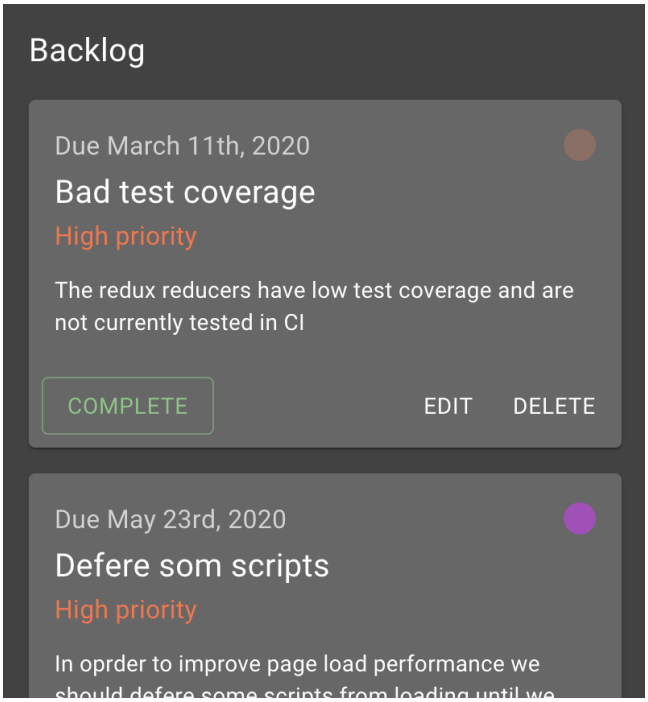

Figure 6: The backlog column in the prototype.

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.

Interviews

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

The interview participants in the study were recruited from two companies, Mentimeter AB and Prototyp Stockholm AB, one of which is a digital agency while the other is a product company providing a SaaS solution for interactive presentations. TODO: describe the participants in more detail?

In the first part of each interview, after exploring the prototype, the participants were presented with the following four statements and asked whether they agree or disagree with each statement. They were also asked to explain their position briefly.

(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 (TODO: x/10) 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 (TODO: x/10) agreed that visualizations in general do add value to the tool and were useful. 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 3) 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 4) or a month in the timeline bar chart (figure 5).

Although the majority of the participants agree with the third statement about the priority of debt items, many ask for a sorting option for the backlog column (figure 6) 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 more disagreement (TODO: x/10) 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 cant 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 of 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 four 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?

A couple of themes emerged from the answers and discussions resulting from the questions: (1) *A designated place for technical debt discussions and decisions*, (2) *Complexity of visual representations of software projects*, (3) *Not another tool* and (4) *Cost and benefit estimates*.

5 DISCUSSION

Topics (placeholders)

- The importance of having a place for structuring and communicating about technical debt in order to successfully manage it and with that also be able to take advantage of the positive aspects of debt.

- The difficulty in presenting an overview of the debt in a single view since debt lives on many levels in a project. Some dimensions of debt are present in very specific locations in the source code an requires a single file overview to be able to localize in the project, where as some other dimensions of debt like architecture or environmental debt can span multiple micro services or code repositories and requires a much broader view of the project to be able to show the location.

Method Critique

Future Work

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.

6 CONCLUSION

TODO: Summarize the findings and the discussion. This should be pretty similar to the abstract.

ACKNOWLEDGMENTS

I want to thank Björn and Tino...

REFERENCES

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

[2] 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

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

[4] 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

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

[6] 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

[7] 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

[8] Ran Mo, Yuanfang Cai, Rick Kazman, and Lu Xiao. 2015. Hotspot Patterns: The Formal Definition and Automatic Detection of Architecture Smells. (2015), 10.

[9] 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

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

[11] 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

[12] 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