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

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 collboration becomes harder. //SOURCE A common method to handle these challanges 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 allowes 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 nessecarily 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 desicion in order to reach a critical deadline in time. However, if not properly managed the debt can be costly by decreasing developemnt velocity and increasing maintenance costs. [9]

Research question

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.

Does a visualization of the evolution of software source code 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

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.

2 RELATED RESEARCH

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 proccess 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

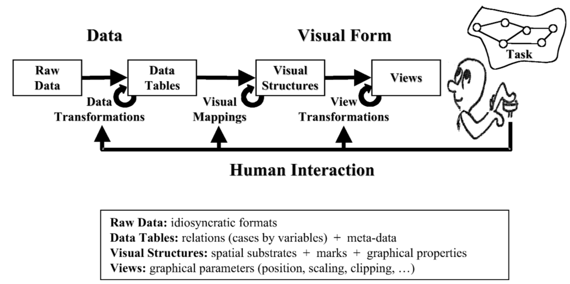

Figure 1: Reference model for visualization by Card et al. 1999. [3]

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. code-swarm 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 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.

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.

Based on the gathered feedback in the first critique session, an interactive high-fi prototype was developed as a second iteration on the concept idea and design. When the defined concept and features were implemented in the prototype, a second critique session was conducted. 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.

The data collected during the user tests invludes both qualitative data gathered during the interviews and quantitative data from screen recordings in the form of time measurements from tasks and quantitative results from structured parts of the interview.

4 RESULTS

In this section the results the study is presented. The result will be divided into subsections for the result from the survey conducted to establish requirements and the results from the thematic analysis of the interviews conducted to evaluate a prototype of a proposed technical debt management tool.

Survey results

The survey was conducted among proffessionals 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 provive 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.

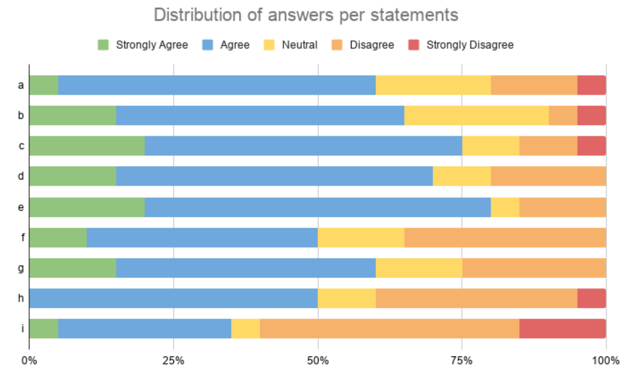

Figure 2: Chart representing the distribution of answers per statement.

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.
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 halv 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 tecnical 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 activly 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 thechnical debt in many software development projects.

Interviews

Based on the data collected and analyzed from the survey, a prototype was designed with the goal to be a common source of information about a software development teams strategy and priorities regarding technical debt.

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 localise 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.

6 CONCLUSION

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