(Self-admitted) Technical debt in mobile application

Federico De Roma
Università degli Studi di Salerno
f.deroma@studenti.unisa.it

ABSTRACT

Technical debt is a metaphor introduced by Cunningham to indicate "not quite right code which we postpone making it right". Examples of technical debt are code smells and bug hazards. Several techniques have been proposed to detect different types of technical debt.
In this document, I present an idea on how to study the diffusion/evolution of technical debt in mobile apps (also Self-Admitted), their impact on change - and fault -proneness of classes as well as how mobile developers deal with them.

CSS CONCEPT

• Software and its engineering → Software evolution; Maintaining software

KEYWORDS

Mining Software Repositories, Technical Debt, Empirical Software Engineering

1. INTRODUCTION

Ward Cunningham coined the technical debt metaphor back in 1993 [1] to explain the unavoidable interests (i.e., maintenance and evolution costs) developers pay while work-ing on not-quite-right code, possibly written in a rush to meet a deadline or to deliver the software to the market in the shortest time possible. In the last years researchers have studied the technical debt phenomenon from different perspectives. Several authors developed techniques and tools aimed at detecting specific types of technical debt, like code smells coding style violations.

Conference'17, July 2017, Washington, DC, USA

 ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00 https://doi.org/10.1145/nnnnnnn.nnnnnnn

Also, researchers have studied the impact of different types of technical debt on maintainability attributes of software systems and when and why technical debt instances are introduced in software systems.
Recently, Potdar and Shihab [2] pioneered the study of self-admitted technical debt (SATD), referring to technical debt instances intentionally introduced by developers (e.g., temporary patches to fix bugs) and explicitly documented in code comments. They showed how it is possible to detect instances of technical debt by simply mining code comments looking for patterns likely indicating (i.e., self-admitting) the presence of technical debt (e.g., fixme, todo, etc.). [3]
My work focuses on the evaluation of technical debt in mobile apps. The open source projects selected are: Wordpress, Bee, Resizer, Trojan, CryptoBuddy, Voice, Passman, BitBump, Meritbook and RIBs. All apps are mostly written in java.
The first step of my work consists of extracting technical debt over the change history of the system. To accomplish this objective, I use SonarQube. This is an open source platform developed by SonarSource for continuous code quality inspection, to perform automatic reviews with static code analysis. Thanks to the tools made available by SonarQube it's possible to estimate the technical debt's value expressed in temporal instant (class by class). After that, I measure the SATD present. I use the approach describe by S. Maldonado, Emad Shihab and Nikolaos Tsantalis. This approach consists of the following steps:

1. Extract the source code comments from the projects.
2. Apply five filtering heuristics to remove comments that result irrelevant for the identification of SATD (e.g., license comments, commented source code).

In order to do the first step, I opened the project from Eclipse and I extracted all the source code comments from the app.

To remove the comments that result irrelevant for the estimation of SATD, I use a tool able to filter only the comments that were useful for the estimation of them, implemented by E. Shibab et al..
This tool is an Eclipse plug-in downloadable from here: https://goo.gl/ZzjBzp
This plug-in is able to identify the comments that contain one of task-reserved words (e.g., “todo”, “fixme”, or “xxx”).
Once this, I measure change - and fault - proneness of classes, measuring when are technical debt items removed and how.
For the change proneness I use a tool made by N. Tsantalis et al..
This tool is able to calculate the probability of a class to change (from 0 to 1).
On the other side, for the fault proneness, I use Pydriller to extract the commit message in order to find when the technical debt has been introduced. After that, I count the number of the bug fixing operation in order to see if the technical debt has been threated and how.

2. MOTIVATIONS

Many studies on (Self-Admitted) technical debt in traditional system are carried out, but little is known in the context of mobile application. The evaluation of SATD can be very useful also in mobile apps.
This is why this idea was born.

3. AIMS

The main goals of the project are:

- To define a correct way to estimate the technical debt (also SATD) in the selected open source project.
- To know how the technical debt has been threated – if it has been removed, how and when.
- To explain the differences between the handle of technical debt in traditional application and the handle of technical debt in mobile application.

4. REFERENCES

[1] W. Cunningham. The WyCash portfolio management system. OOPS Messenger, 4(2):29–30, 1993.

[2] A. Potdar and E. Shihab. An exploratory study on self-admitted technical debt. In Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution, ICSME ’14, pages 91–100, 2014.

[3] 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories. A Large-Scale Empirical Study on Self-Admitted Technical Debt (Gabriele Bavota, Barbara Russo)

[4] IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. XX, NO. X, MONTH 2016 1.
Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt.
Everton da S. Maldonado, Emad Shihab, Member, IEEE, and Nikolaos Tsantalis, Member, IEEE

Federico De Roma