(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 (also Self-Admitted) in mobile apps, their impact on change - and fault -proneness of classes as well as how mobile developers can deal with them.

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.
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 (also SATD) in mobile apps.
The open source projects selected are: Bee, Zero, Afwall, Kiwix, TapJoy, PageTurner, Alfresco, Flutter, ChatSecure and NotifyReddit. All apps are mostly written in java. For each project have been selected 5 versions.

App	URL of Repository
Bee	https://github.com/APISENSE/bee-android
TapJoy	https://github.com/reime005/react-native-tapjoy
Zero	https://github.com/ekylibre/zero-android
Afwall	https://tinyurl.com/m722ouo
Kiwix	https://tinyurl.com/ycvufxwn
PageTurner	https://tinyurl.com/yc26yklh
Alfresco	https://tinyurl.com/ya533yya
ChatSecure	https://tinyurl.com/pxcupkk
NotifyReddit	https://tinyurl.com/ydce46ge
Flutter	https://tinyurl.com/yd25noy3

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.

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. METHODOLOGY AND STEPS

I. TECHNICAL DEBT ESTIMATION

The first step of my work consists of extracting technical debt over the change history of the system. To accomplish this objective, I used 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 cloning the app from GitHub, I ran SonarQube launching it from the console, reporting the location of the app folder with this command: *./sonar.sh start*
After that, I put the app's folder in the *bin* folder of SonarScanner, I modified the file "sonar-project.properties" in this way:

sonar.projectKey=projectName
sonar.projectName=projectName
sonar.sources=/Users/federico/Desktop/SonarQube/sonar-scanner-4.2.0.1873-macosx/bin
sonar.java.binaries=.
sonar.scm.disabled=true

and I started Sonar Scanner with this command: *./sonar-scanner*

II. SELF-ADMITTED TECHNICAL DEBT (SATD) ESTIMATION

To analyze SATDs (Self-Admitted Technical Debts), I used 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 used 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").

III. CHANGE PRONENESS ESTIMATION

Change proneness is a quality characteristic of software artifacts that represents their probability to change in the future due to: *(a) evolving requirements*, *(b) bug fixing*, or *(c) ripple effects*. In the literature, change proneness has been associated with many negative consequences along software evolution. I would tell that change is something that is inevitable during the evolution of software, but in the context of the project I dealt with the negative aspect of a change, so for instance, due to a design problem (code smell) or presence of vulnerabilities, etc...
The third step of this analysis consisted of analyzing the change proneness of the app.
To accomplish this, I used 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).
The tool has a simple interface and is very easy to use.

IV. FAULT PRONENESS ESTIMATION

Fault proneness is sometimes defined as the probability that an artifact contains a fault.
To measure the fault prone, I followed these steps:

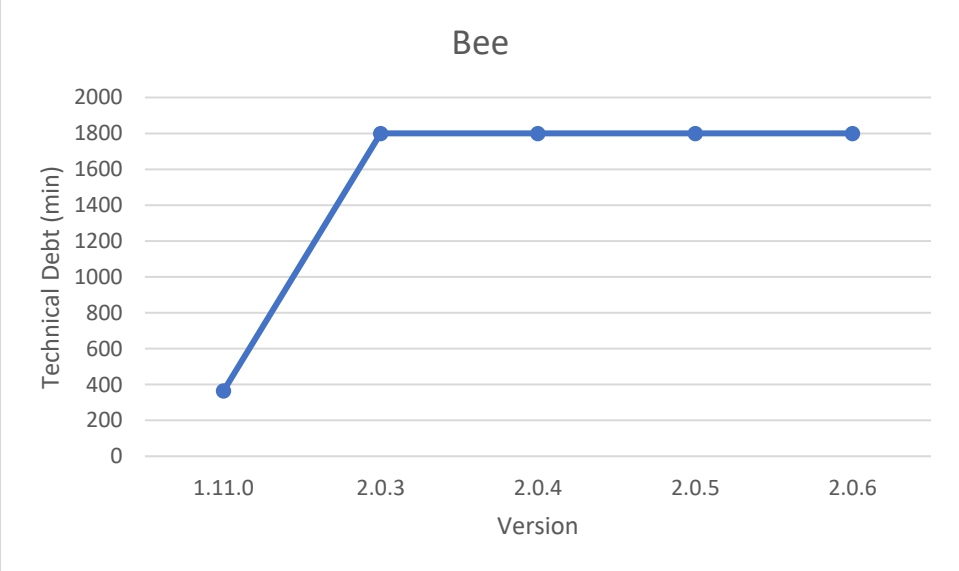
1) I analyzed the commit messages of the app using *PyDriller*, that is a Python framework that helps developers in analyzing Git repositories. So, it is relatively easy extract information such as commits, developers, modifications, diffs, and source codes.
2) I counted the number of *fixing operations* for each version of the project.

5. RESULTS AND FINDINGS

I. BEE

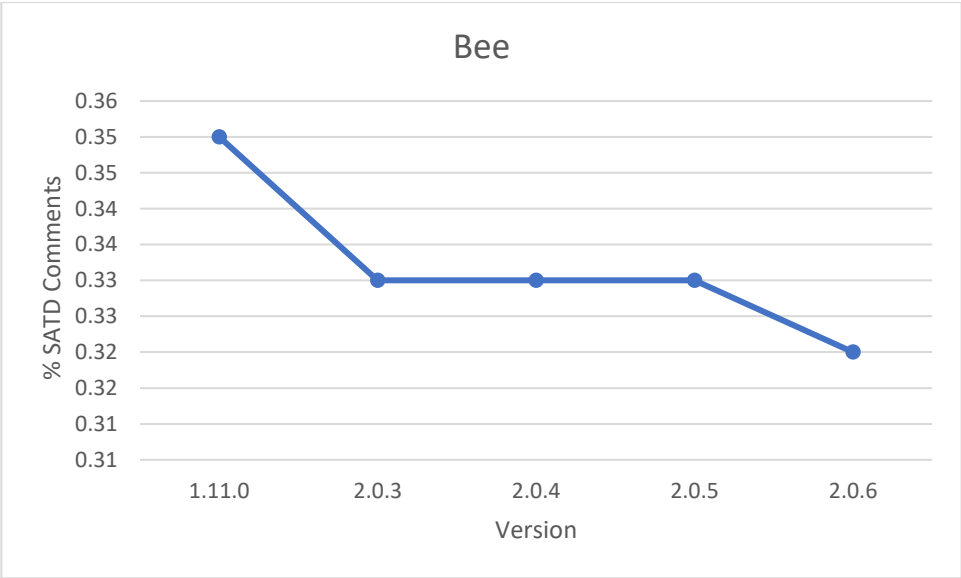
- *TECHNICAL DEBT*

Version	Technical Debt (min)
1.11.0	364
2.0.3	1800
2.0.4	1800
2.0.5	1800
2.0.6	1800



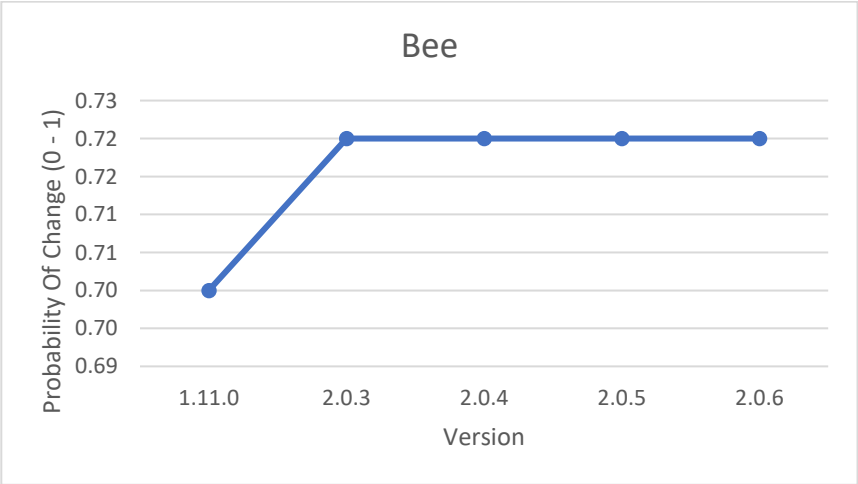
- *% SATD COMMENTS*

Version	% SATD Comments
1.11.0	0,35
2.0.3	0,33
2.0.4	0,33
2.0.5	0,33
2.0.6	0,32



- *PROBABILITY OF CHANGE*

Version	Probability of Change (0 - 1)
1.11.0	0,70
2.0.3	0,72
2.0.4	0,72
2.0.5	0,72
2.0.6	0,72



- *N° FIXING OPERATIONS*

Version	N° Fixing Operations
1.11.0	4
2.0.3	2
2.0.4	1
2.0.5	1
2.0.6	0

- *CONSIDERATIONS*

The TD has increased from version 1.11.0 to 2.0.3 and then remained constant until the last version (2.0.6).
There are few SATD comments in all versions of the app (about *0.35%*), and all the comments regarding *requirement debt*.
The probability for each version to be changed it's quite high because it's *higher than 0.5*.
There weren't many fixing operations between the versions.

- *POSSIBLE CAUSES AND IMPLICATIONS*

Let's consider the version *2.0.3* where the technical debt has undergone its increase.
Let's consider two classes:

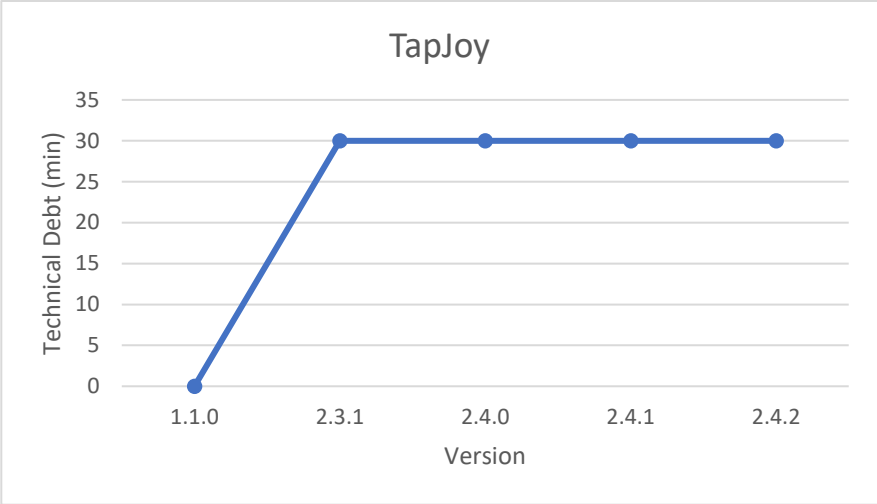
- The class with the highest TD, that is *EventObserver.java* that has *6h 5min* of TD.
- A class with a low TD, *CommonDetailsFragment.java* that has *2min* of TD.

EventObserver.java was not present in the previous version, so this class is a possible cause of the increase of TD.
The SATD tool didn't find any SATD comments in the classes.
The class with the highest TD has a probability of change of 0.88, instead the class with a low TD has a probability of change of 0.75.
This probably means that classes involved in TD have a higher % in terms of change prone.
In this version of the app there have been 2 fixing operations. These 2 operations didn't affect the TD that remains constant until the last version of the app.

II. TAPJOY

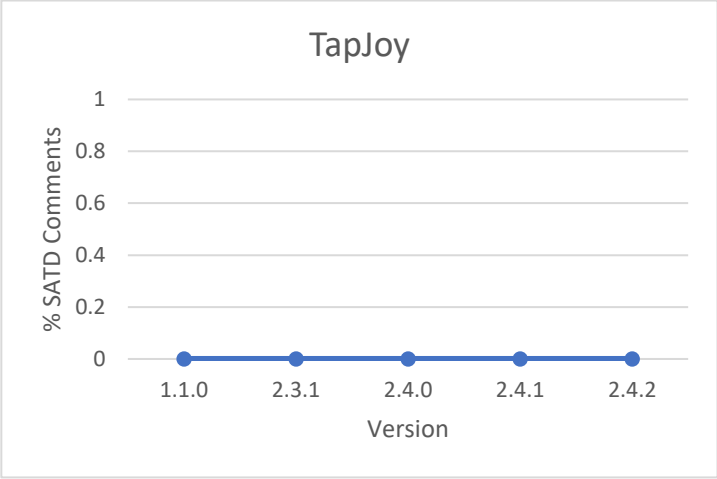
- *TECHNICAL DEBT*

Version	Technical Debt (min)
1.1.0	0
2.3.1	30
2.4.0	30
2.4.1	30
2.4.2	30



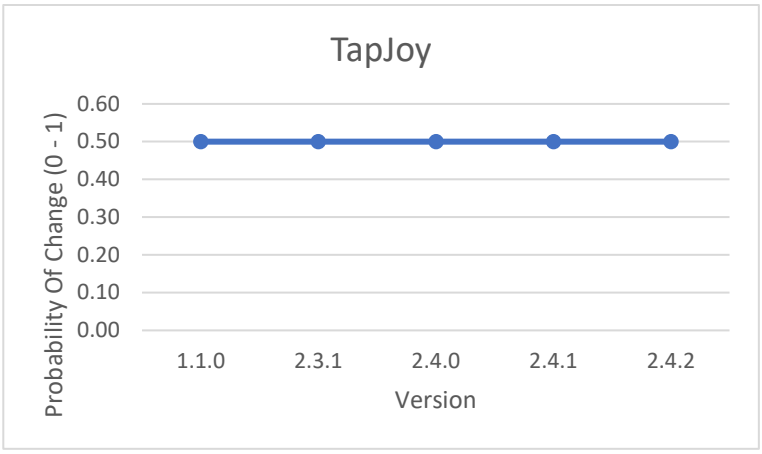
- *% SATD COMMENTS*

Version	% SATD Comments
1.1.0	0
2.3.1	0
2.4.0	0
2.4.1	0
2.4.2	0



- *PROBABILITY OF CHANGE*

Version	Probability Of Change (0 - 1)
1.1.0	0,50
2.3.1	0,50
2.4.0	0,50
2.4.1	0,50
2.4.2	0,50



- *N° FIXING OPERATIONS*

Version	N° Fixing Operations
1.1.0	1
2.3.1	4
2.4.0	4
2.4.1	4
2.4.2	4

- *CONSIDERATIONS*

The TD has increased from version 1.1.0 to 2.3.1 and then remained constant until the last version (2.4.2).
There isn't any type of SATD comments in the source code.
The probability of change it's the same for all versions and it is *0.5*.
There weren't many fixing operations between the versions.

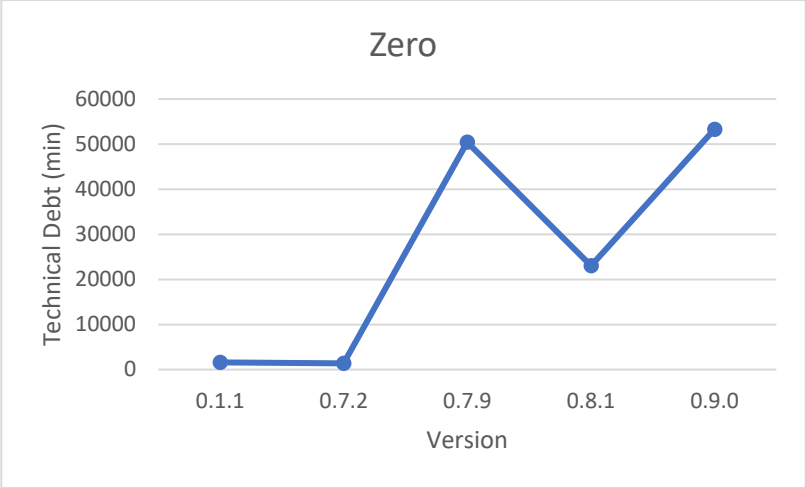
- *POSSIBLE CAUSES AND IMPLICATIONS*

The first version of the app (1.1.0) has not TD. The TD was introduced in the version *2.3.1*.
The increment of *classes* and *LOC* between the two versions contributed to the introduction of technical debt in the app. In fact, the first version of the app had *8 classes* and *392 LOC*, while the version 2.3.1 had *13 classes* and *443 LOC*.
The class with the highest TD in version 2.3.1 is *MyTJPlacementListener.java* that has *13min* of TD.
The SATD tool didn't find any SATD comments.
The most interesting thing is that the probability of change is the same for all versions. This probably means that, in the case, the TD didn't affect the change proneness.
Among the versions considered, were made *17 fixing operations*. These operations didn't affect the TD that remains constant until the last version of the app.

III. ZERO

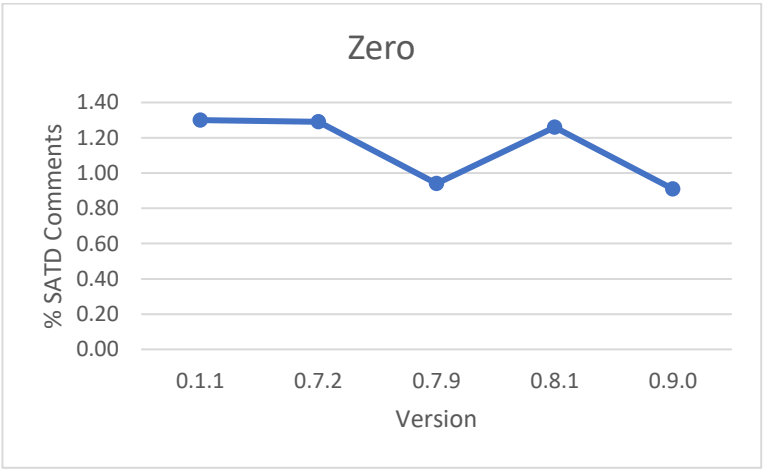
- *TECHNICAL DEBT*

Version	Technical Debt (min)
0.1.1	1620
0.7.2	1380
0.7.9	50400
0.8.1	23040
0.9.0	53280



- *% SATD COMMENTS*

Version	% SATD Comments
0.1.1	1,30
0.7.2	1,29
0.7.9	0,94
0.8.1	1,26
0.9.0	0,91



- *PROBABILITY OF CHANGE*

Version	Probability Of Change (0 - 1)
0.1.1	0,57
0.7.2	0,62
0.7.9	0,66
0.8.1	0,63
0.9.0	0,66

- *N° FIXING OPERATIONS*

Version	N° Fixing Operations
0.1.1	0
0.7.2	1
0.7.9	0
0.8.1	0
0.9.0	0

- *CONSIDERATIONS*

The TD was not subject to large changes until version *0.7.9* where increased by *49020min*.
Later, it decreased (in version *0.8.1*), then increased again in version *0.9.0*.
The % of SATD comments it's quite high, especially in version 0.1.1 where it's about *1.3%*.
The probability for each version to be changed it's quite high because it's *higher than 0.5*.
There weren't any kind of fixing operations between the considered version, except for the version 0.7.2.

- *POSSIBLE CAUSES AND IMPLICATIONS*

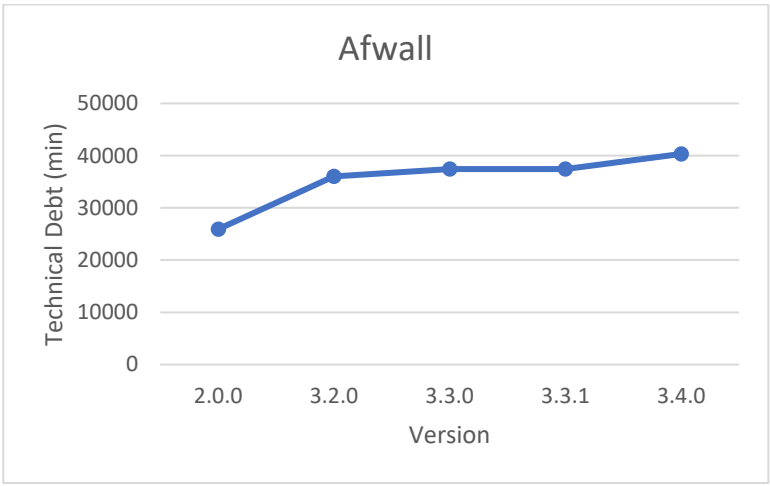
Let's consider the version where the TD has undergone its increase, version *0.7.9*.
In this version the TD where increased by *49020min*.
The class with the higher TD is *Grammar.java*, that has a TD of *17280min*.
This class was not present in the previous version, so it is a possible cause of the increase of TD.

In this class there isn't any kind of SATD comments.
Grammar.java has a probability of change of 0.89. This explains the increase of the change prone from *0.62* to *0.66*.
In this version there weren't any kind of fixing operations. The absence of fixing operations also influences the TD.

IV. AFWALL

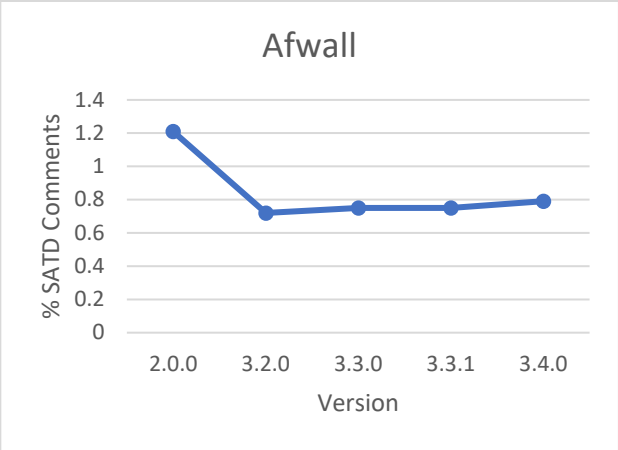
- *TECHNICAL DEBT*

Version	Technical Debt (min)
2.0.0	25920
3.2.0	36000
3.3.0	37440
3.3.1	37440
3.4.0	40320



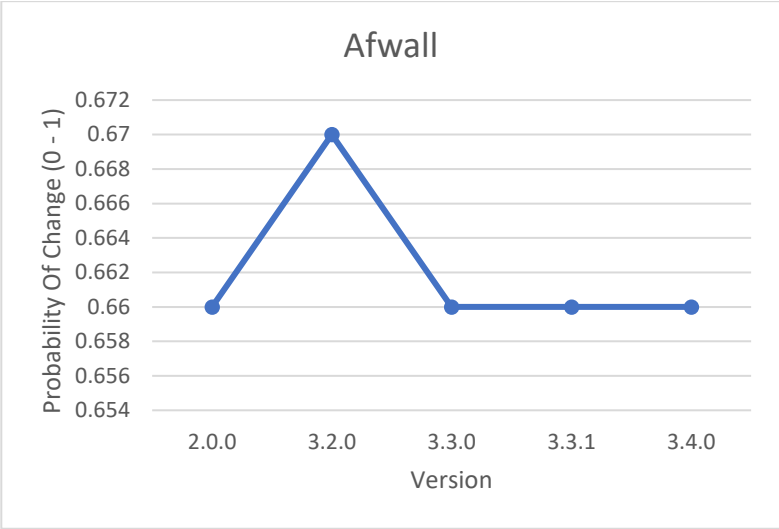
- *% SATD COMMENTS*

Version	% SATD Comments
2.0.0	1,21
3.2.0	0,72
3.3.0	0,75
3.3.1	0,75
3.4.0	0,79



- *PROBABILITY OF CHANGE*

Version	Probability Of Change (0 - 1)
2.0.0	0,66
3.2.0	0,67
3.3.0	0,66
3.3.1	0,66
3.4.0	0,66



- *N° FIXING OPERATIONS*

Version	N° Fixing Operations
2.0.0	3
3.2.0	1
3.3.0	1
3.3.1	2
3.4.0	2

- *CONSIDERATIONS*

The TD was not subject to large changes between versions. The most important variation was that between the version *2.0.0* and the version *3.2.*0. The TD between this two version increased by *10080min*.
The % of SATD comments it's quite high, especially in version *2.0.0* where it's about *1.21%*.
The probability for each version to be changed it's quite high because it's *higher than 0.5*.
There weren't many fixing operations between the versions.

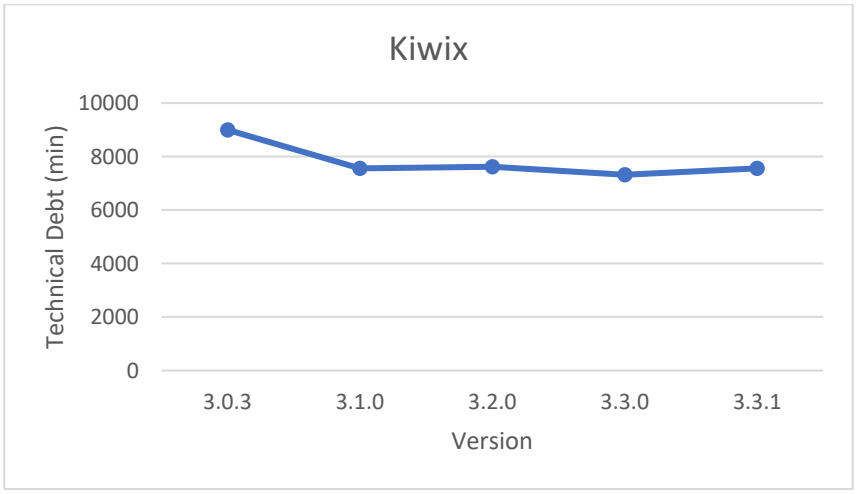
- *POSSIBLE CAUSES AND IMPLICATIONS*

Let's consider the version where the TD has undergone its increase, version *3.2.0*.
In this version the TD where increased by *10080min*.
The class with the higher TD is *Api.java*, that has a TD of *4740min*. This class, in version *2.0.0* had a TD of *3300min*. This means that, in version *3.2.0*, the developers have added new elements to the class, consequently increasing the td as well.
In this class there are some SATD comments, regarding design debt (for example a method that can be written in a better way).
Api.java has a probability of change of *0.5* in both versions, so, in this case, the increment of TD didn't affect the change prone.
Among the versions considered, were made *9 fixing operations*.

V. KIWIX

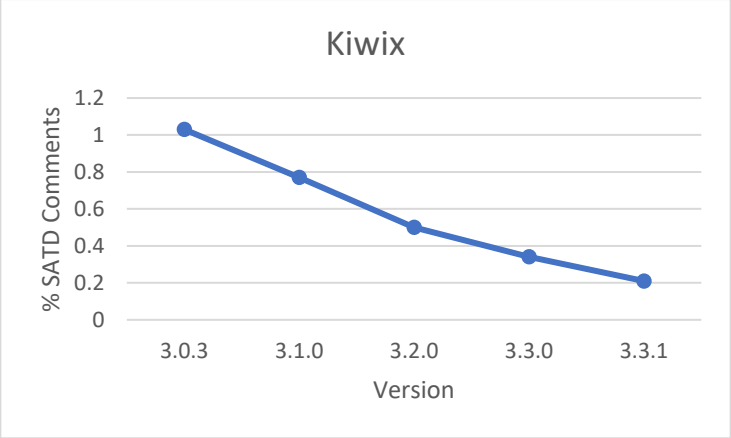
- *TECHNICAL DEBT*

Version	Technical Debt (min)
3.0.3	9000
3.1.0	7560
3.2.0	7620
3.3.0	7320
3.3.1	7560



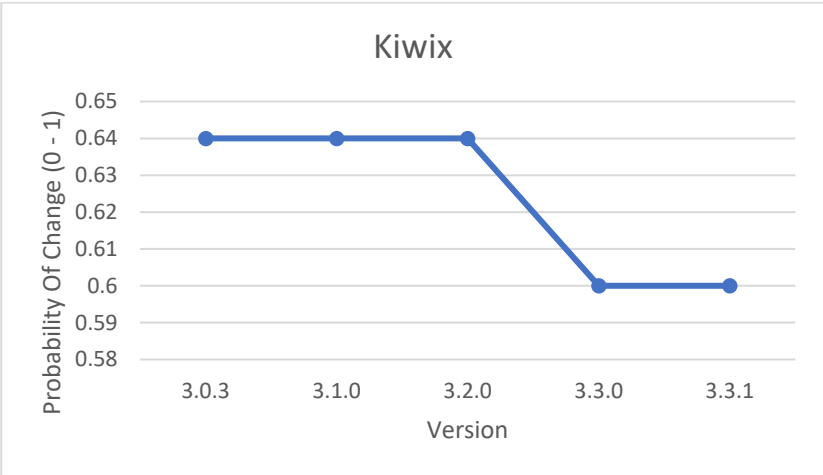
- *% SATD COMMENTS*

Version	% SATD Comments
3.0.3	1,03
3.1.0	0,77
3.2.0	0,5
3.3.0	0,34
3.3.1	0,21



- *PROBABILITY OF CHANGE*

Version	Probability Of Change (0 - 1)
3.0.3	0,64
3.1.0	0,64
3.2.0	0,64
3.3.0	0,6
3.3.1	0,6



- *N° FIXING OPERATIONS*

Version	N° Fixing Operations
3.0.3	1
3.1.0	6
3.2.0	0
3.3.0	14
3.3.1	20

- *CONSIDERATIONS*

The TD was not subject to large changes between versions. The most important variation was that between the version *3.0.3* and the version *3.1.*0. The TD between this two version decreased by *1440min*.
The % of SATD comments it's quite high in version *3.0.3* (about *1.03%*) then decrease in later versions.
The probability for each version to be changed remains more or less constant in all versions (about *0.65*).
In later versions of the app, there have been many fixing operations.

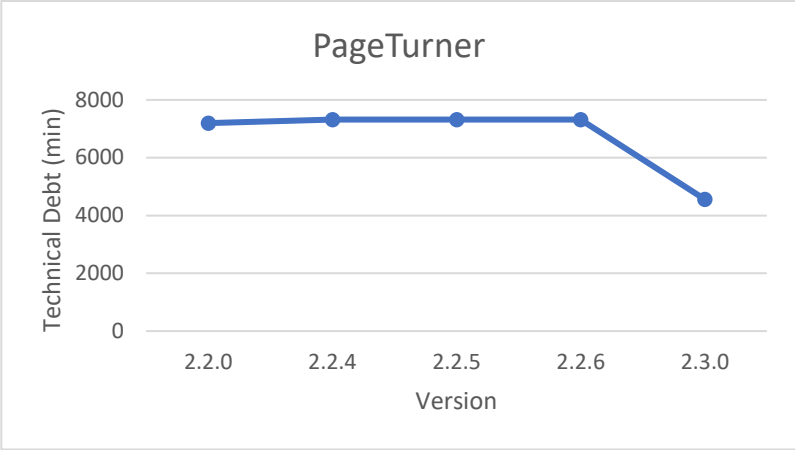
- *POSSIBLE CAUSES AND IMPLICATIONS*

Let's consider the version where the TD has undergone its decrease, version *3.1.0*.
In this version the TD where decreased by *1440min*.
In version *3.0.3*, the class with the higher TD is *MainActivity.java*, that has a TD of *402min*.
This class, in version *3.1.0* was not present. This means that the removal of the class from the source code, or the extraction of the class in other classes, contributed to the decrease of TD.
The decrease of TD can also be due to the decrease of SATD comments.
The probability of change remains constant in all versions (about *0.65*), so in this case the TD didn't affect the change prone.
Among the versions considered, were made *41 fixing operations*, which seem not to have affected TD.

VI. PAGETURNER

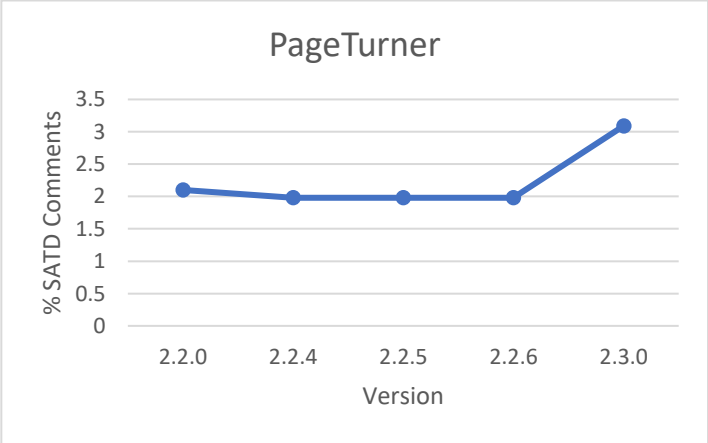
- *TECHNICAL DEBT*

Version	Technical Debt (min)
2.2.0	7200
2.2.4	7320
2.2.5	7320
2.2.6	7320
2.3.0	4560



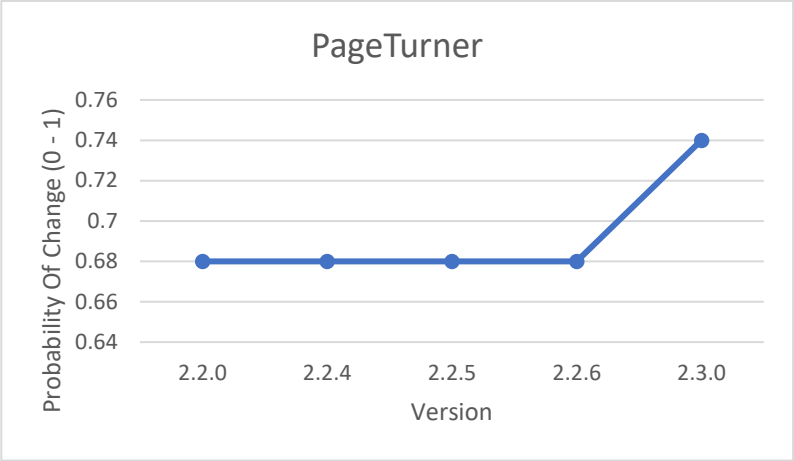
- *% SATD COMMENTS*

Version	% SATD Comments
2.2.0	2,1
2.2.4	1,98
2.2.5	1,98
2.2.6	1,98
2.3.0	3,09



- *PROBABILITY OF CHANGE*

Version	Probability Of Change (0 - 1)
2.2.0	0,68
2.2.4	0,68
2.2.5	0,68
2.2.6	0,68
2.3.0	0,74



- *N° FIXING OPERATIONS*

Version	N° Fixing Operations
2.2.0	10
2.2.4	4
2.2.5	11
2.2.6	3
2.3.0	1

- *CONSIDERATIONS*

TD remains fairly constant from version *2.2.0* to version *2.2.6*, after that, in version *2.3.0*, it underwent a variation of *2760min*.
The % of SATD comments it's quite high (*>1%*). It remains fairly constant until version *2.3.0*, where the % increase about *3.09%*.
The probability for each version to be changed remains more or less constant in all versions (about *0.7*).
There have been many fixing operations, especially in version *2.2.0* and in version *2.2.5*.

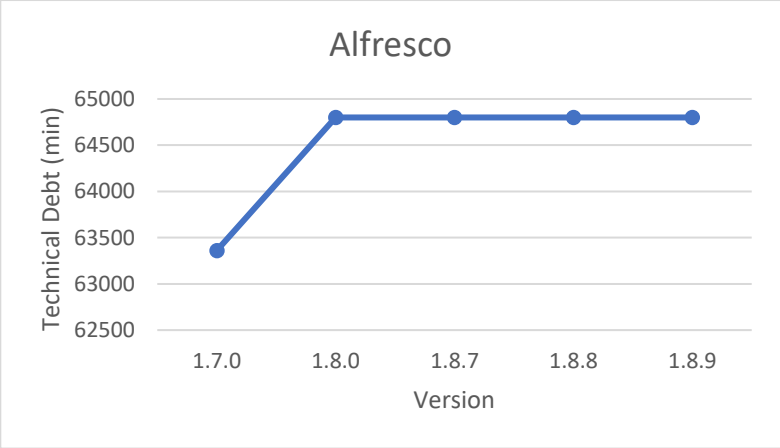
- *POSSIBLE CAUSES AND IMPLICATIONS*

Let's consider the version where the TD has undergone its decrease, version *2.3.0*.
In this version the TD where decreased by *2760min*.
In version *2.2.4*, the class with the higher TD is *ReadingFragment.java*, that has a TD of *327min*. This class, in version *2.3.0* has a TD of *198min*. This means that the TD has been treated by the developers that modified some elements of the class.
The interesting thing is that despite the TD has decreased, the SATD has increased to about *3.09%*. The majority of the SATD introduced, is *design debt*.
The probability of change remains constant in all versions (about *0.7*). In this case, a decrease in TD led to an increase in change prone (from *0.68* to *0.74*).
Among the versions considered, were made *29 fixing operations*. Despite, in version *2.2.0*, *10 fixing operations* were carried out, this does not seem to have influenced the TD which has increased by *120min*.

VII. ALFRESCO

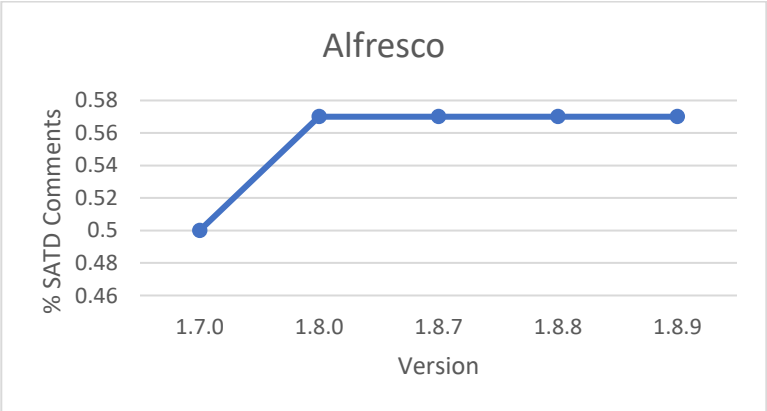
- *TECHNICAL DEBT*

Version	Technical Debt (min)
1.7.0	63360
1.8.0	64800
1.8.7	64800
1.8.8	64800
1.8.9	64800



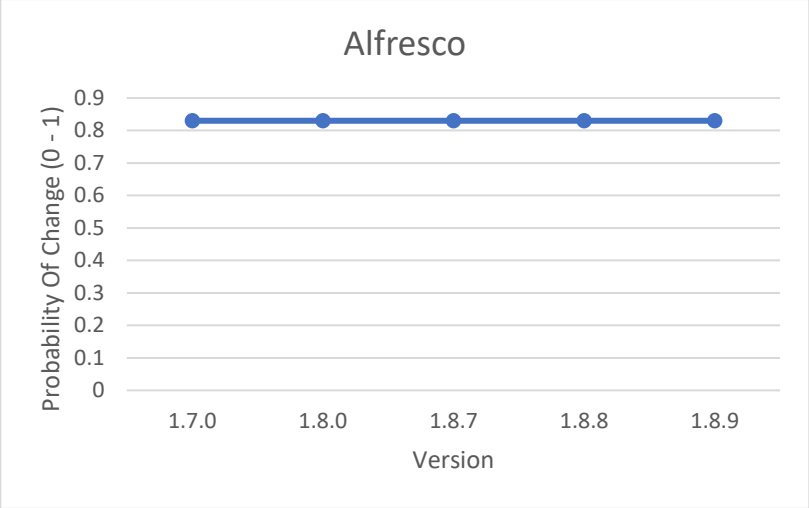
- *% SATD COMMENTS*

Version	% SATD Comments
1.7.0	0,5
1.8.0	0,57
1.8.7	0,57
1.8.8	0,57
1.8.9	0,57



- *PROBABILITY OF CHANGE*

Version	Probability Of Change (0 - 1)
1.7.0	0,83
1.8.0	0,83
1.8.7	0,83
1.8.8	0,83
1.8.9	0,83



- *FIXING OPERATIONS*

Version	N° Fixing Operations
1.7.0	9
1.8.0	10
1.8.7	3
1.8.8	3
1.8.9	1

- *CONSIDERATIONS*

TD remains fairly constant in all versions considered. The only small variation, there was between version *1.7.0* and version *1.8.0* where the TD increased by *1440min*. The % of SATD comments it's not so high (*<1%*). It remains fairly constant in all versions considered (about *0.5%*).
The probability for each version to be changed remains constant in all versions (about *0.83*). There have been many fixing operations, especially in version *1.8.0* and in version *1.7.0*.

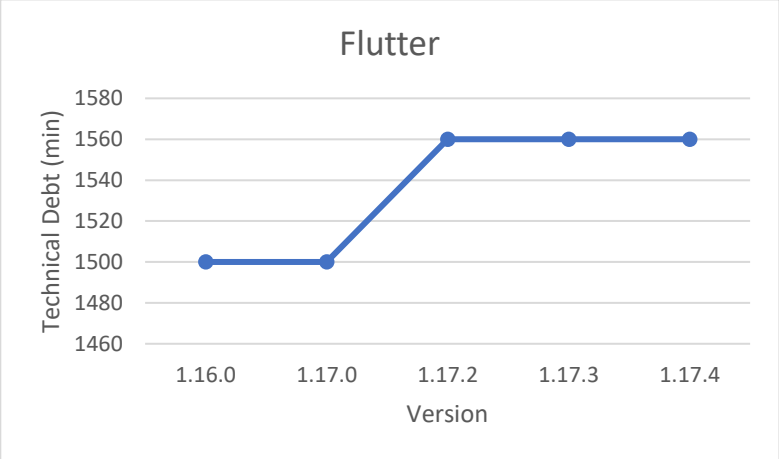
- *POSSIBLE CAUSES AND IMPLICATIONS*

Let's consider the version where the TD has undergone its only small variation, version *1.8.0*. In this version the TD where increased by *1440min*.
The increment of *classes* and *LOC* between the first two versions considered contributed to the increment of technical debt. In fact, version *1.7.0* has *930* classes and *99,963* LOC, while version *1.8.0* has *1,100* classes and *101,551* LOC.
SATD comments remains constant in all versions of the app. The majority of the comments concern *design debt*.
The probability of change remains constant in all versions (about *0.83*), so in this case the TD didn't affect the change prone.
Among the versions considered, were made *25 fixing operations*, which seem not to have affected TD.

VIII. FLUTTER

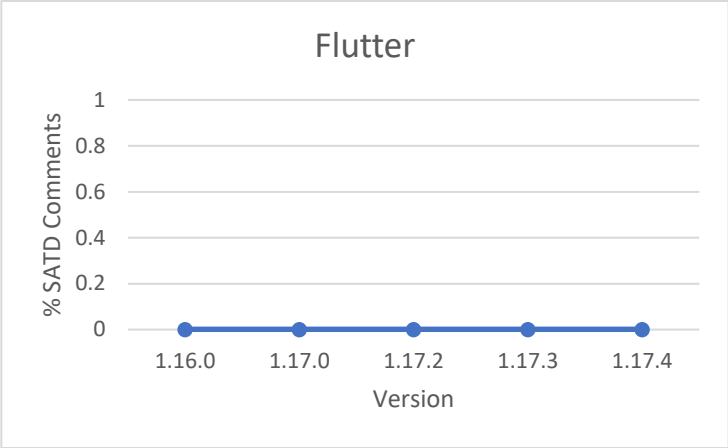
- *TECHNICAL DEBT*

Version	Technical Debt (min)
1.16.0	1500
1.17.0	1500
1.17.2	1560
1.17.3	1560
1.17.4	1560



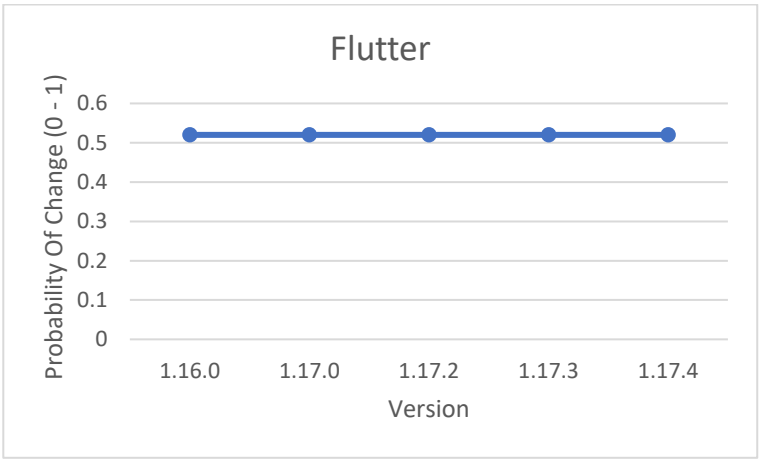
- *% SATD COMMENTS*

Version	% SATD Comments
1.16.0	0
1.17.0	0
1.17.2	0
1.17.3	0
1.17.4	0



- *PROBABILITY OF CHANGE*

Version	Probability Of Change (0 - 1)
1.16.0	0,52
1.17.0	0,52
1.17.2	0,52
1.17.3	0,52
1.17.4	0,52



- *N° FIXING OPERATIONS*

Version	N° Fixing Operations
1.16.0	10
1.17.0	6
1.17.2	13
1.17.3	8
1.17.4	3

- *CONSIDERATIONS*

The TD was not subject to large changes between versions.
The isn't any kind of SATD comments.
The probability for each version to be changed it's quite high because it's *higher than 0.5* and it remains constant in all versions considered.
There have been many fixing operations, especially in version *1.16.0* and in version *1.17.2*.

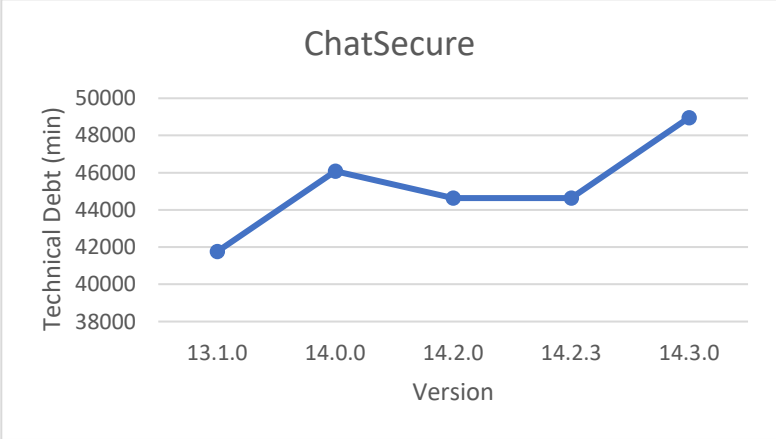
- *POSSIBLE CAUSES AND IMPLICATIONS*

TD remains fairly constant in all versions considered.
The class with the higher TD, in all versions, is *MainActivity.java*, that has a TD of *52min*.
There isn't any kind of SATD comments.
The probability of change remains constant in all versions (about *0.52*).
Despite the *10 fixing operations* carried out in version *1.16.0*, and the *13 fixing operations* carried out in version *1.17.2* this did not influence in any way the TD.

IX. CHATSECURE

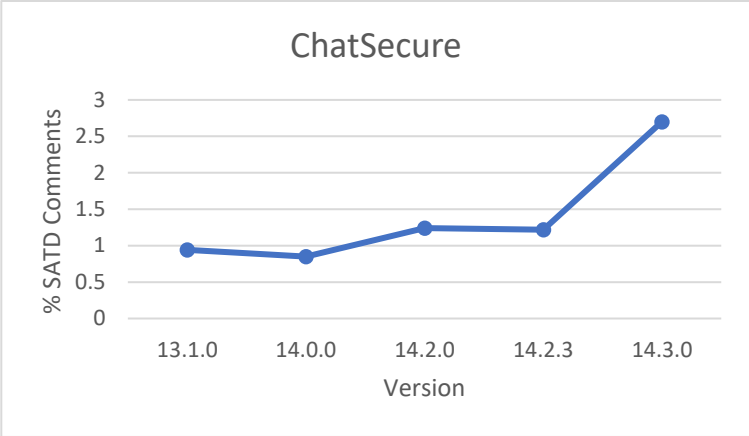
- *TECHNICAL DEBT*

Version	Technical Debt (min)
13.1.0	41760
14.0.0	46080
14.2.0	44640
14.2.3	44640
14.3.0	48960



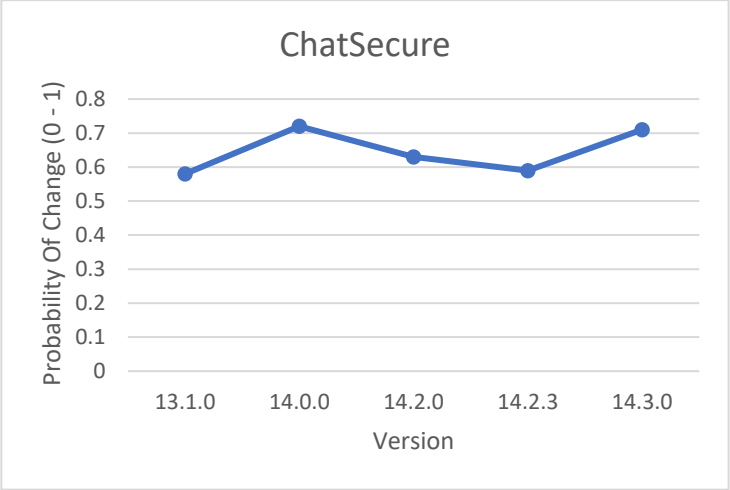
- *% SATD COMMENTS*

Version	% SATD Comments
13.1.0	0,94
14.0.0	0,85
14.2.0	1,24
14.2.3	1,22
14.3.0	2,7



- *PROBABILITY OF CHANGE*

Version	Probability Of Change (0 - 1)
13.1.0	0,58
14.0.0	0,72
14.2.0	0,63
14.2.3	0,59
14.3.0	0,71



- *N° FIXING OPERATIONS*

Version	N° Fixing Operations
13.1.0	2
14.0.0	4
14.2.0	4
14.2.3	1
14.3.0	0

- *CONSIDERATIONS*

There have been 2 significant variations between versions. The first variation is between version *13.1.0* and version *14.0.0*; the second variation is between version *14.2.3* and version *14.3.0*. In both cases, the TD increased by *4320min*.
The % of SATD comments it's quite high. The highest percentage was in version *14.3.0* (about *2.7%*).
The probability for each version to be changed it's quite high (*>0.5*).
There have not been many fixing operations.

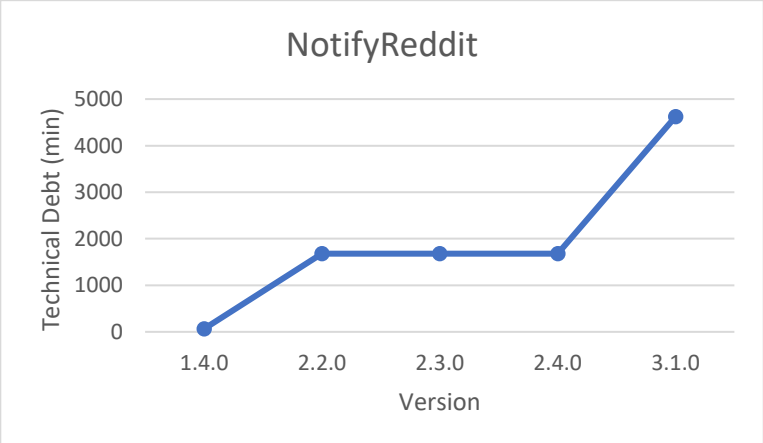
- *POSSIBLE CAUSES AND IMPLICATIONS*

Let's consider the two version where the TD has undergone its increase, version *14.0.0* and version *14.3.0*.
In those versions the TD where increased by *4320min*.
Both variations, are due to the increase of classes and LOCs between versions.
The majority of the SATD comments introduced, concern *design debt*.
In this case, an increase in TD led to an increase in change prone (from *0.58* to *0.72* and from *0.59* to *0.71*).
Among the versions considered, were made *11 fixing operations*, which seem not to have affected TD.

X. NOTIFYREDDIT

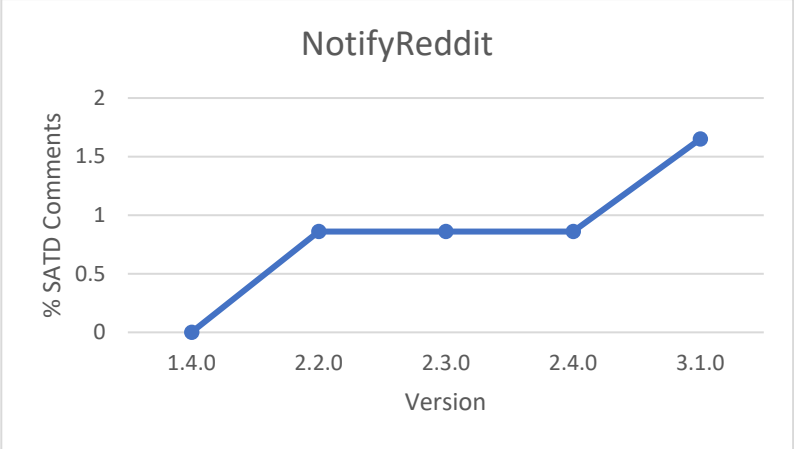
- *TECHNICAL DEBT*

Version	Technical Debt (min)
1.4.0	64
2.2.0	1680
2.3.0	1680
2.4.0	1680
3.1.0	4620



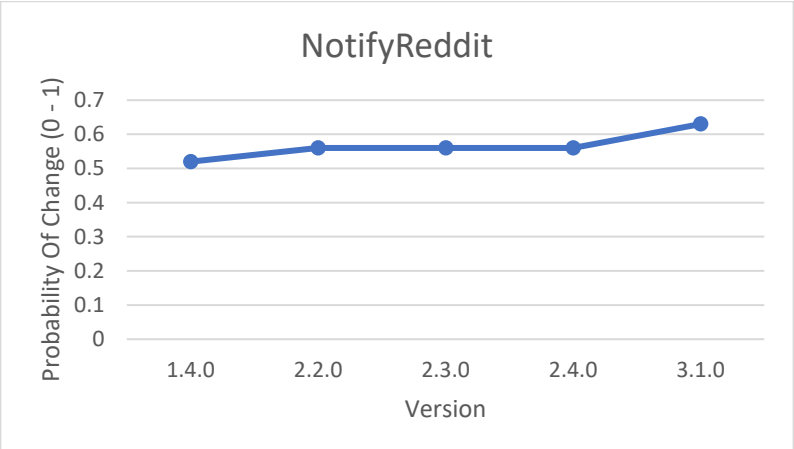
- *% SATD COMMENTS*

Version	% SATD Comments
1.4.0	0
2.2.0	0,86
2.3.0	0,86
2.4.0	0,86
3.1.0	1,65



- *PROBABILITY OF CHANGE*

Version	Probability Of Change (0 - 1)
1.4.0	0,52
2.2.0	0,56
2.3.0	0,56
2.4.0	0,56
3.1.0	0,63



- *N° FIXING OPERATIONS*

Version	N° Fixing Operations
1.4.0	4
2.2.0	1
2.3.0	1
2.4.0	0
3.1.0	2

- *CONSIDERATIONS*

There have been 2 significant variations between versions. The first variation is between version *1.4.0* and version *2.2.0*; the second variation is between version *2.4.0* and version *3.1.0*. In the first case, TD increased by *1,616min*; in the second case, TD increased by *2,940min*.
The % of SATD comments it's quite high. The lowest percentage was in version *1.4.0* (*0%*). The highest percentage was in version *3.1.0* (about *1.65%*).
The probability for each version to be changed it's quite high (*>0.5*).
There have not been many fixing operations.

- *POSSIBLE CAUSES AND IMPLICATIONS*

Let's consider the two version where the TD has undergone its increase, version *2.2.0* and version *3.1.0*.
In the first case, TD where increased by *1,616min*; in the second case, TD where increased by *2,940min*. Both variations, are due to the increase of classes and LOCs between versions.
The majority of the SATD comments introduced, concern *design debt*.
In this case, an increase in TD led to an increase in change prone (from *0.52* to *0.56* and from *0.56* to *0.63*).
Among the versions considered, were made *8 fixing operations*, which seem not to have affected TD.

6. CONCLUSIONS AND FUTURE WORK

The term technical debt is being used for practitioners and researchers in the software engineer community to express shortcuts and workarounds employed in software projects. These shortcuts will most often impact the maintainability of the project hindering the development if not addressed properly.
My work explores specifically technical debt (also self-admitted technical debt) in the context of mobile application.
In my study I analyzed 10 open source projects which are: Bee, Zero, Afwall, Kiwix, TapJoy, PageTurner, Alfresco, Flutter, ChatSecure and NotifyReddit. All apps are mostly written in java. For each project 5 versions have been selected.
I used them in order to:

- estimate the TD;
- identify and classify different types of SATD;
- measure the probability for each app to be changed (change proneness);
- count the number of fixing operations and their impact on TD and SATD (fault proneness)

I find that the TD can decrease between versions due to refactoring operations (for example due to an extract class refactoring) or, in some cases, it can increase due to the addition of functionalities to the app or due to the increment of classes and LOCs between versions.
I find that the majority of the self-admitted technical debt comments are design debt. The % of SATD comments it's quite high in all considered apps (>1%). Based on this result, we can say that the self-admitted technical debt types that developers admit to the most are related with the design of the project, potentially indicating that developers feel the need to admit and be forthcoming about such debt.
In all the apps considered the probability of change is quite high because it's higher than 0.5. This means that the apps have the 50% to be changed in the future. Moreover, the TD seems not to impact on change proneness the, in some cases, it is even increased when the TD decreased.
In the projects analyzed, there weren't many fixing operations but, when they were, they do not seem to have contributed neither to the decrease nor to the increase of TD.
I hope that this work will encourage future research in the area of technical debt and self-admitted technical debt in the context of mobile application. I also think that the information provided by this dataset can be a cornerstone for more advanced techniques as natural language processing. In a future work I plan to improve the current analysis adding more projects to it. With a richer dataset I expect that more patterns and characteristics of technical debt and self-admitted technical will be retrieved.

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

[4] Everton da S. Maldonado and Emad Shihab. Detecting and Quantifying Different Types of Self-Admitted Technical Debt

Federico De Roma