Technical Debt Analysis

Ashish Verma
School of Electrical Engineering and Computer Science
University of Ottawa

Abstract-**

Keywords—Technical debt, Repackaging, LOC, Refactoring

I. Introduction

Software development is prone to failure. Most often these failures are due to tight schedules or ill programming practices which makes the architecture of project rigid and very resistant to changes for future feature completeness. This is called as Technical debt. This technical debt metaphor was coined by Ward Cunningham in the 1992 OOPSLA experience report to describe how long-term code quality is traded for short-term gains such as increased productivity [1]. This additional effort of feature completeness in future can be compared to monetary debt [2]. Technical debt which is not taken care of, increases software entropy. Technical debt is not always bad, it is sometimes required to make progress during development and evaluate if project is moving on right path. E.g. POC or Proof of concept to develop prototype during requirement gathering phase. In Section 2 and Section 3 various dimensions of technical debt are represented and discussed for there severity and preventions. Also, a cost prediction model is used to calculate the debt in monetary terms.

II. Managing technical debt using tools

This section aims to categorize different debts as found in four projects. The projects analyzed for technical debt are pretty well known by everyone, some are even used during software development and one project belongs to the entertainment category and is a famous game Super Mario Bros. Below is the list of projects analyzed:

1. Fastjson: It is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. [3]
2. DrJava: It is a lightweight development environment for writing Java programs. It is designed primarily for students, providing an intuitive interface and the ability to interactively evaluate Java code. [4]
3. JFreeChart: It is a comprehensive free chart library for the Java platform that can be used on the client-side and server-side. [5]
4. Super Mario Bros.: It is a cult classic game played on Nintendo NES but this game is converted to java project and exists as an open source. [6]

All the above projects are widely used and have a strong open source community contributing to it. It will be worthwhile to analyze technical debt for these projects and deduce preventive measures for it.

To analyze projects for various technical debt and violations below tools were used:

- SonarQube [7]
- Teamscale [8]
- JaCoCo [9]
- PMD [10]
- Checkstyle [11]

The description and usage of these tools is discussed in further section of this paper.

A. *Quality assessment*

Quality assessment of project deals with quantifying quality attributes of project. These are non-functional requirements which are used to evaluate performance of the software system. Some of the quality attributes are:

- Reliability: It refers to failure free operation performed by software for a specified period of time in a specified environment. [12]
- Security: It is set of precautionary measures taken and implemented in the software system to protect it from malicious attacks or hacking related activities.
- Maintainability: It is a degree to which software system can be enhanced and repaired in case of failure.
- Testability: It is a measure with which we can easily create test cases and check for code coverage for software system. If testability is high it means it is easy to find bugs in software system by performing testing.

The tools used to analyze project for quality attributes are SonarQube, and Teamscale. SonarQube identifies reliability and security issue within software system efficiently. Teamscale can assess maintainability. To measure code coverage JaCoCo can be used but most of the project were not having Junit tests so it wasn't used to assess code coverage of software system i.e. testability.

Analysis using SonarQube

Project 1: Fastjson
Results of reliability can be drawn using below criteria:
Bugs: 67
Having rating: C (Moderate)
Remediation effort required: 1 day

Refer Fig 1, for graphical based reliability analysis. Color identifies severity of bugs. Red is the most severe. Size of the circle signifies lines of code or size of the bug. Horizontal axis or x-axis signifies lines of code where vertical axis or y-axis signifies reliability remediation effort.

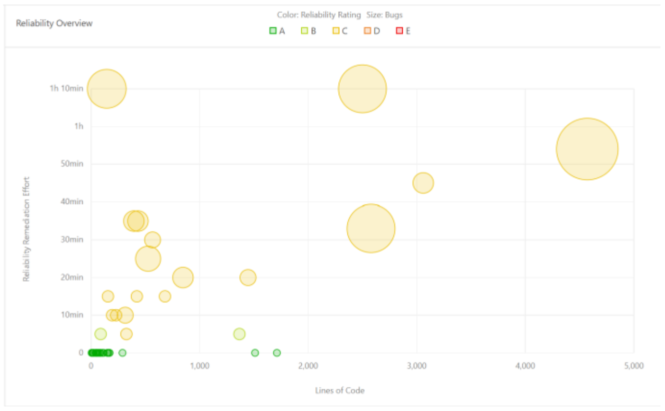


Fig 1. Reliability analysis of Fastjson

Similarly, analysis for Project 2, 3 and 4 were done. Below are the results.

Project 2: DrJava
Bugs: 6
Having rating: E (Severe)
Remediation effort required: 50 minutes

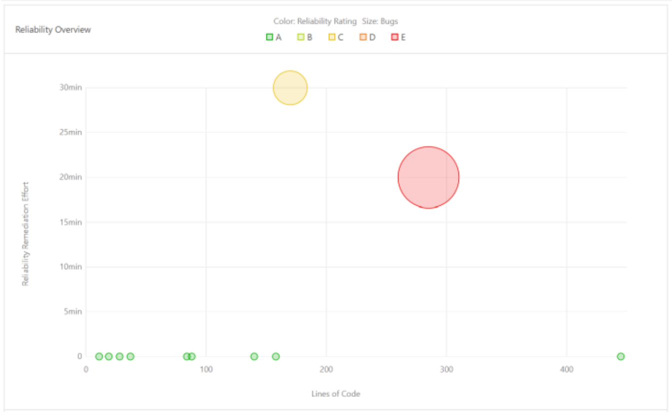


Fig. 2. Reliability analysis of DrJava

Project 3: JFreeChart
Bugs: 289
Having rating: D (High)
Remediation effort required: 7 days 1 Hour

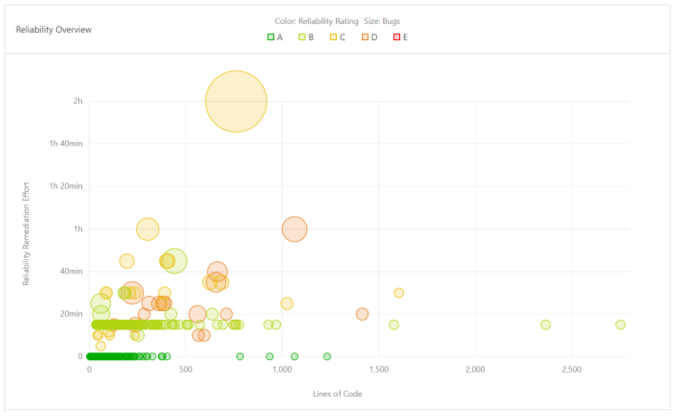


Fig. 3. Reliability analysis of JFreeChart

Project 4: Super Mario Bros
Bugs: 180
Having rating: E (Severe)
Remediation effort required: 2 days 1 Hour

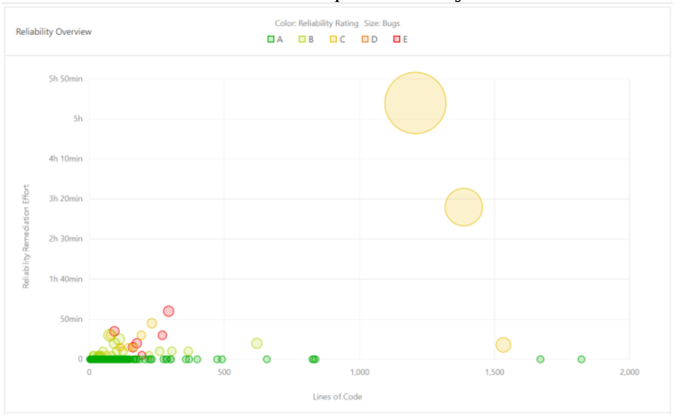


Fig. 4. Reliability analysis of Super Mario Bros

Analysis using Teamscale

Teamscale quantifies quality based on four parameters namely code anomalies, documentation, naming, and structure. This together correspond to findings summary. Below are some of the findings for each project.

Project 1: Fastjson
Total findings: 48.9K
Number of severe findings: 468
Severe finding type: Code anomalies

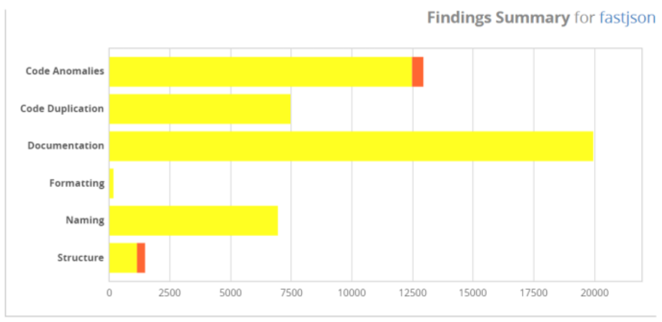


Fig. 5. Findings of Fastjson

Similarly, findings for Project 2, 3 and 4 is documented below:

Project 2: DrJava
Total findings: 213
Number of severe findings: 3
Severe finding type: Code anomalies
DrJava has very few structural/architectural quality defects.

Project 3: JFreeChart
Total findings: 8.6K
Number of severe findings: 320
Severe finding type: Code anomalies and Structure
JFreeChart has almost equal amounts of code anomalies and structural defect.

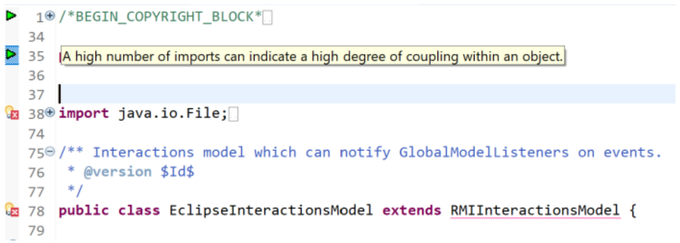
Project 4: Super Mario Bros
Total findings: 6.9K
Number of severe findings: 163
Severe finding type: Structure
It has severe code anomalies but they were very few, and doesn't require much effort to be fixed.

B. Technical debt identification

To identify technical debt, two tools, PMD and Checkstyle, capable of performing static code analysis which can be integrated into eclipse for free were used. These tools can identify technical debt from the readability point of view. These tools evaluate project on the basis of indentation, comments and violations.

It was analyzed that Fastjson has very high number of urgent violations. This project has more than 30 packages and more than 150 classes. Most of the violation were regarding comments being too large and cyclomatic complexity. Highest number of violations recorded were 4372. Average violation per KLOC were 1100. Similarly, DrJava and JFreeChart have 750 and 700 violations per KLOC. Super Mario Bros is very unstructured project for which there were many suggestions on indentation and naming of variable generated using check code feature of PMD. Fig. 5. depicts suggestion generated by PMD for DrJava.

Teamscale calculated high number of naming convention violation which corresponds to readability of the program. This was found in Super Mario Bros with count of 1865.



```
/*BEGIN_COPYRIGHT_BLOCK*

A high number of imports can indicate a high degree of coupling within an object.


import java.io.File;

/** Interactions model which can notify GlobalModelListeners on events.
 * @version $Id$
 */
public class EclipseInteractionsModel extends RMIInteractionsModel {

```

Fig. 5. PMD notifications for DrJava

C. Technical debt representation

Various types of technical debt identified in 4 projects are presented in tabular form. Teamscale and SonarQube was used to logically present this data, spread across 3 dimensions of technical debt categorized as:
Dimension 1: Code debt
Dimension 2: Documentation debt
Dimension 3: Architectural debt

Each technical debt is assigned a unique ID. This unique ID can be interpreted as:

ID: 1.2.X where,
First digit or 1 is the project number
Second digit or 2 is the dimension as mentioned above, i.e. documentation debt
X is the technical debt number in serial order.

A total of 24 tables, with 6 tables belonging to each project were documented. These tables are presented in separate section of this report namely 'Appendix'. Different types of debts can be referred and inferred from these tables.

D. Technical debt estimation

To calculate debt estimation SonarQube was used. SonarQube estimates debt based on bugs, vulnerabilities and code smells found in the software system. Below in table 4 are the overall estimates for all the software system:

TABLE I. TECHNICAL DEBT ESTIMATION

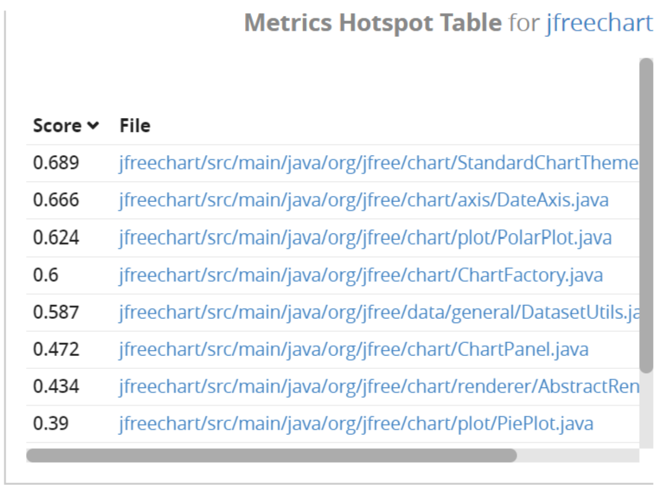
Project	Estimate
Fastjson	Debt Estimate: 72 days Debt contribution ratio: Major: Code smells, 72 days Least: Bugs, 1 day Grade: A (maintainability is less than 5%)
DrJava	Debt Estimate: 2 days Debt contribution ratio: Major: Code smells, 1 day 4 hours Least: Vulnerabilities, 0 Grade: A (maintainability is less than 5%)
JFreeChart	Debt Estimate: 78 days Debt contribution ratio: Major: Code smells, 77 days Least: Vulnerabilities, 6 hour 55 minutes Grade: A (maintainability is less than 5%)
Super Mario Bros	Debt Estimate: 61 days Debt contribution ratio: Major: Code smells, 60 days Least: Bugs, 2 days 1 hour Grade: A (maintainability is less than 5%)

The remediation effort was very high for JFreeChart around 7days and 1 hour, whereas, it was low for DrJava around 50 minutes.

It was also interesting to observe that DrJava has 0% code duplications whereas, all other projects have duplicate code snippets in some respect.

E. Technical debt monitoring

Monitoring of technical debt is an iterative process wherein technical debt is observed for any changes with a goal to contain it. If technical debt is introduced in a system our primary objective should be to reduce it. If reduction of technical debt is not possible than next best approach is to contain it and not let it spread. This can be achieved by observing certain section of code which have tendency to spread technical debt in the system. To monitor technical debt teamscale is used. Teamscale offers wide variety of options to customize dashboard which can suit need of any developer. In dashboard there is an analysis for metrics hotspot table. This table can be used to identify hotspots within the code. This widget can list potential files which have a capability of introducing technical debt in the system. It lists files based on score and lines of code.


Metrics Hotspot Table for jfreechart

Score	File
0.689	jfreechart/src/main/java/org/jfree/chart/StandardChartTheme
0.666	jfreechart/src/main/java/org/jfree/chart/axis/DateAxis.java
0.624	jfreechart/src/main/java/org/jfree/chart/plot/PolarPlot.java
0.6	jfreechart/src/main/java/org/jfree/chart/ChartFactory.java
0.587	jfreechart/src/main/java/org/jfree/data/general/DatasetUtils.ja
0.472	jfreechart/src/main/java/org/jfree/chart/ChartPanel.java
0.434	jfreechart/src/main/java/org/jfree/chart/renderer/AbstractRen
0.39	jfreechart/src/main/java/org/jfree/chart/plot/PiePlot.java

Fig. 6. Metrics Hotspot Table for JFreeChart

Further if any changes to code are done then it keeps on updating the hotspot list. Interestingly, predicted highest score is assigned to one file from DrJava project which is referred in all the classes. Its score is 1.

Another great widget to monitor technical debt is LOC vs. Findings trend. This line graph can be used to monitor if technical debt spreads across system. As the code is edited this graph will be updated automatically. We can configure this widget with additional metrics value to see overall trend at once. As project wasn't modified, so it is showing a linear trend in Fig. 7.

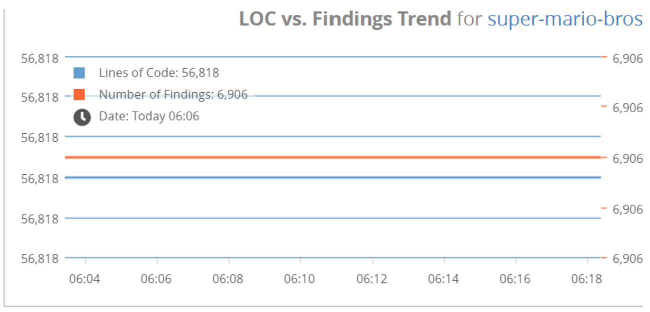


Fig. 7. LOC vs. Findings trend for Super Mario Bros

Another great widget to view 3D projection of size of individual source file is Code City. This can used to check LOC of individual classes depicted like a building in a map. As seen in Fig. 8. The shorter cases have small block size whereas larger classes have larger block size.

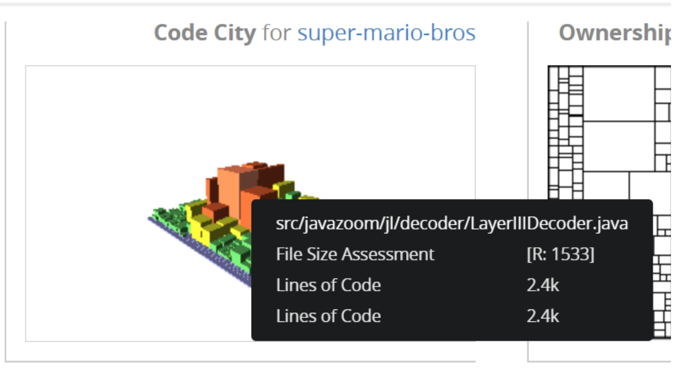

Code City for super-mario-bros
Ownership
src/javazoom/jl/decoder/LayerIIIDecoder.java
File Size Assessment [R: 1533]
Lines of Code 2.4k
Lines of Code 2.4k

Fig. 8. Code city for Super Mario Bros

At last, to check the clone percentage in the class we can use clone coverage treemap. The blue color changes its shade depending on the amount of duplicate code found in the software system. For e.g., the dark blue box shown in the Fig. 9. refers to clone percentage of more than 90% whereas other boxes have less than 50% clone percentage. Using this widget, we can easily monitor the debt and prevent it from increasing as we can get an overview of all the classes in the project in one single widget.

Using combination of above widgets in one dashboard for each project, it was observed that:

- DrJava has 0% clone coverage, it is a very well-structured source code.
- Fastjson suffers from formatting issues. Like, multiple declarations in statement.
- JFreeChart has very high nesting depth.
- Super Mario Bros have very long methods, which can act like god class if added with some more functionality.

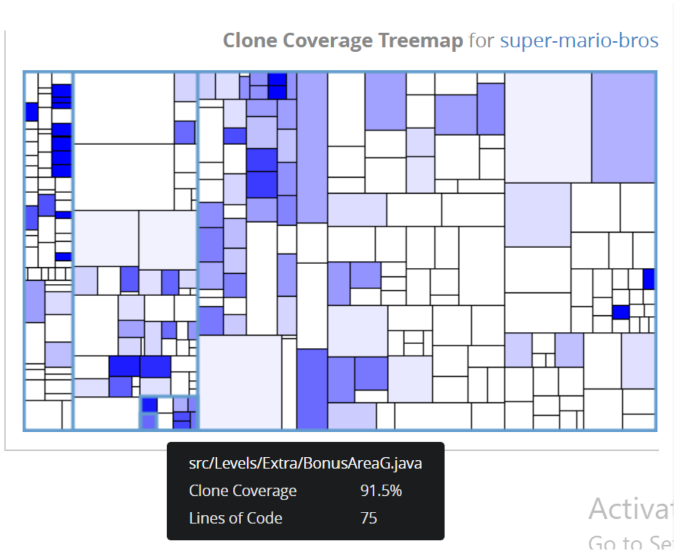

Clone Coverage Treemap for super-mario-bros
src/Levels/Extra/BonusAreaG.java
Clone Coverage 91.5%
Lines of Code 75
Activat
Go to Se

Fig. 9. Clone coverage treemap for Super Mario Bros

F. *Technical debt repayment*

Repaying the debt is a careful estimation as it comes with its tradeoffs. Repaying technical debt has several advantages and disadvantages, because by repaying debt, we are completely modifying the software system. If debt is not carefully paid off it can introduce many more anomalies into the system, thereby, increasing technical debt. Repaying technical debt also includes extra development cost, due to which it is most often not considered as an option when project budget is too steep. There are several approaches to repay technical debt. Some of them are: refactoring, reengineering, Bug fixing, fault tolerance, automation testing etc.

From our 4 projects we have found 3 dimensions of technical debt i.e. code debt, test debt, documentation debt. In Fastjson we can use rewriting to fix the code debt as there is a lot of duplicate code. This can increase technical debt exponentially if not eliminated properly.

DrJava is a well-structured software system, it doesn't require much changes except having proper comments in place to enhance readability of the source code. There is also almost no duplicity found in DrJava code.

JFreeChart and Super Mario Bros are two projects having many faults from code smells to vulnerability issues. We can use repacking and refactoring technique as it contains most of the classes being referred across all source map. Different methods like extract method, extract class, extract interface, move method and move field can be used. The dominant classes or parent classes can be broken down into sub-modules using move method refactoring. Another way to fix code smells is by bug fixing as there were many bugs in these two projects. Also, JaCoCo results confirmed that code coverage wasn't sufficient for these two projects. Proper test cases and suites can be designed and implemented using automation testing tools like Selenium [13] and ZapTest [14] to pay off test debt.

G. *Technical debt prevention*

To prevent technical debt from getting introduced into system is a continuous effort of developer both on timely and efficient basis. There are various tools to help developer monitor and identify technical debt in the system. Some of these tools we have used in this paper can be effectively put in to use.

SonarQube is the best tool as it identifies security vulnerabilities along with documentation debt in the software system. It also gives the fair idea on how reliable the system is and how maintainable it is. The duplication parameter in SonarQube also helps in identifying any duplicate code in the system. But before using tools we can also use some common agile best practices to avoid the entry of technical debt in the system. Some of them are:

1. Pair programming
2. Test-driven Development
3. Refactoring
4. Continuous Integration

Above written best practices if put in to use, can effectively help in preventing the technical debt from entering into the system.

H. *Discussion*

While analyzing projects for there quality metrics various challenges were encountered. Some of them were:

1. SonarQube and Teamscale doesn't support some of the well-known programming languages like AngularJS.
2. One advantage of Teamscale over SonarQube is that it doesn't require project to be converted to Maven. To use SonarQube some of the projects were converted to Maven.
3. Both these tools don't support auditing and code review functionalities like Squale. So, it is not able to compute social dimension of code based on audit.

Doing analysis for the project was very knowledgeable. It helped in relating theoretical knowledge with practical usage of tools. It also helped in getting aware with different metrics, which tools propose, and locating hotspots for specific files in the project. While analyzing the bugs and code smells it helped in identifying good programming practices and architectural pattern to improve code quality.

III. Technical Debt Principal Calculation Model and Further Analysis

In this section firstly, a calculation model is proposed. Using this model, we can quantify technical debt in monetary terms. Secondly, some tools are proposed which can quantify social dimension and can monitor audit and review process of software system.

A. *Calculation Model*

There are various ways to calculate principal technical debt. Two most commonly used methods are:

1. Method supported by SonarQube

2. Method supported by CAST (by Curtis, Sappidi and Szynkarski)

In this paper, as we have used SonarQube we will be using method supported by SonarQube. This will in turn help us verify results predicted by tool and make monetary judgement about them. As technical debt is usually described in monetary terms, so it will be quantified in terms on money using any desired currency. This monetary value is actually the interest which developer will repay in order to eliminate technical debt from the system.

After careful analysis of all the projects, the default values of calculation model were modified based on project complexity. These values gave near to approximate results when used in estimating principal model.

The formula to estimate the debt value is [15]

$$Debt = duplication + violation + comments + coverage + complexity + design \tag{1}$$

Here the individual components of debt are calculated as below:

$$duplication = cost\ to\ fix\ one\ block * duplicated\ blocks \tag{2}$$

$$violations = cost\ to\ fix\ one\ violation * mandatory\ violations \tag{3}$$

$$comments = cost\ to\ comment\ one\ API * public\ undocumented\ API \tag{4}$$

$$coverage = cost\ to\ cover\ one\ of\ complexity * uncovered\ complexity\ by\ tests \tag{5}$$

$$design = cost\ to\ cut\ an\ edge\ between\ two\ files * package\ edge\ weights \tag{6}$$

$$complexity = cost\ to\ split\ a\ method * (function\ complexity\ distribution\ \geq 7) + cost\ to\ split\ a\ class * (class\ complexity\ distribution\ \geq 80) \tag{7}$$

The default values as proposed by SonarQube were modified based on project complexity, is represented in Table 2.

TABLE II. DEFAULT VALUES FOR CALCULATING EFFORT

Cost	Default value (in person hours)
cost_to_fix_one_block	1.9
cost_to_fix_one_violation	0.192
cost_to_comment_one_API	0.13
cost_to_cover_one_of_complexity	0.32
cost_to_cut_an_edge_between_two_files	4.42
cost_to_split_a_method	0.52
cost_to_split_a_class	8.45

The value calculated from effort estimation is in person hours. These hours are then converted to days, where on an average a person works 8 hours per day. Post calculating days it is then converted to monetary value where $400 is cost per person-day.

For Fastjson cost calculation is as follows:

$$Debt = 1.9 * 65 + 0.192 * 87 + 0.13 * 72 + 0.32 * 59 + 0.52 * 71 + 4.42 * 84$$

$$Debt = 576.644\ person\ hours$$

$$Debt = \frac{576.644}{8}\ person\ days = 72.08\ person\ days$$

$$Cost = 72.08 * 400 = \$28{,}832$$

Similarly, the estimation for each project is represented in below Table 3

TABLE III. COST TO REPAY TECHNICAL DEBT

Project	Cost (in US Dollars)
Fastjson	$28,832
DrJava	$824
JFreeChart	$31,240
Super Mario Bros	$24,142

It can be seen that DrJava has the least cost when it comes to repayment of technical debt. It is a well-maintained code with very small number of smells.

B. More tools to manage Technical Debt

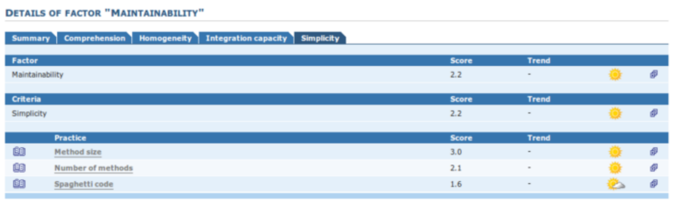
Managing technical debt efficiently and effectively, is now becoming a concern amongst people working in IT. As more and more people are becoming aware of technical debt the demand for tools to manage technical debt are now on a surge. New software companies have now started to contribute to the development to technical debt tools.

One such tool to analyze and maintain technical debt is Squale (Software Quality Enhancement) [15]. Squale is an open-source platform that monitors source-code for software quality attributes. This application can support multi-language applications. Squale has very strong developer community and has frequent updates which enhances its prediction value on continuous basis. It is distributed under LGPL v3 license.

Setting up Squale is very simple. Go to www.squale.org and download the installer. The installer is GUI based and easy to install by following steps displayed on screen.

Squale has multiple features, some of them include:

1. Assisting developer in improving their code. Fig. 10.



DETAILS OF FACTOR "MAINTAINABILITY"

Factor	Score	Trend
Maintainability	2.2	-

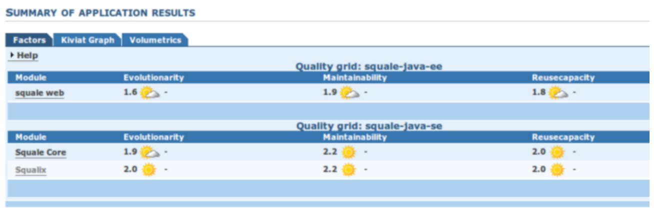
Criteria	Score	Trend
Simplicity	2.2	-

Practice	Score	Trend
Method size	3.0	-
Number of methods	2.1	-
Spaghetti code	1.6	-

Fig. 10. Quality metrics from Squale

2. Creating users in the system with multiple administrators.

3. Project managers can customize the view of dashboard.
4. It allows top-managers to monitor overall health of software system via dashboard. Fig. 11.



Quality grid: squale-java-ee

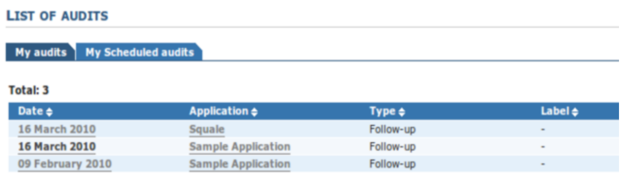
Module	Evolutionarity	Maintainability	Reusecapacity
squale web	1.6 ⛅ -	1.9 ⛅ -	1.8 ⛅ -

Quality grid: squale-java-se

Module	Evolutionarity	Maintainability	Reusecapacity
Squale Core	1.9 ⛅ -	2.2 ☀ -	2.0 ☀ -
Squalix	2.0 ☀ -	2.2 ☀ -	2.0 ☀ -

Fig. 11. Dashboard from Squale

5. It allows audits/reviews to be performed for the software system. Fig. 12.



Total: 3

Date	Application	Type	Label
16 March 2010	Squale	Follow-up	-
16 March 2010	Sample Application	Follow-up	-
09 February 2010	Sample Application	Follow-up	-

Fig. 12. Audit view from Squale

IV. CONCLUSION

This paper analyses technical debt for four projects which is presented in separate section of this paper. Each section deals with different aspects of technical debt from identification, management, repayment and prevention. We observed that some of open-source projects which have very strong developer community had the least debt for e.g. DrJava. Whereas, Fastjson from Alibaba has very high technical debt even though it is backed up by big corporate. The debt depends on the number of decisions taken by developer intentionally and unintentionally. Intentional decisions can be taken under schedule pressure where unintentional are taken due to lack of knowledge.

We analyzed projects for there quality attributes and its metrics. Based on these attributes they were categorized under one of three dimensions of technical debt. This information is presented in the tabular form in section C and Appendix. At the end a cost model based on SonarQube, with modified values to suit the project complexity, is proposed. This model is used to estimate technical debt repayment as an interest monetary value in US dollars. It was observed that JFreeChart has very high technical debt with cost of $31,240. Whereas Fastjson is not far behind even though it is a product from Alibaba, a known corporate. After careful analysis of the project monitoring and repayment technique of technical debt is discussed. At the end an open-source tool Squale is recommended to be used for technical debt measurement and performing audits in the system.

This paper helped in gaining practical knowledge by the use of tools. The measurement calculated by tools were then compared with theoretical concepts to carefully evaluate the importance of each technical debt. The easiest to use tools were SonarQube and Teamscale, which has very intuitive GUI. Few shortcomings of using these two tools are:

1. It doesn't support wide variety of programming language
2. There is no way of analyzing local project on the cloud by uploading zip file
3. Pricing of licenses scales with lines of code for most advanced editions.

APPENDIX – IDENTIFIED DIMENSIONS FOR TECHNICAL DEBT

TABLE IV. FASTJSON : TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	1.1.1 Code Anomalies
Name	Null pointer dereference
Location	src/main/java/com/alibaba/fastjson/JSONPath.java:139
Responsible/author	Developer
Dimension	Code debt
Date/time	Apr 14, 2019 06:01
Context	The variable 'context' may contain a null value at this point and is dereferenced
Propagation rules	This may lead to an exception being thrown at runtime.
Intentionality	Intentionally

TABLE V. FASTJSON : TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	1.1.2 Code Anomalies
Name	Catch of generic exception
Location	src/test/java/com/alibaba/json/bvt/issue_1000/Issue1066.java:54-56
Responsible/author	Developer
Dimension	Code debt
Date/time	Apr 14, 2019 06:01
Context	Catch clause catches generic exception Exception
Propagation rules	Generic exceptions should not be handled by a catch block. For example, in Java, the exceptions Throwable, Error, RuntimeException, and Exception of the java.lang package count as generic.
Intentionality	Intentionally

TABLE VI. FASTJSON : TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	1.2.1 Documentation
Name	Task tags
Location	src/test/java/com/alibaba/json/bvt/parser/stream/JSONReaderTest_error2.java:48
Responsible/ author	Developer
Dimension	Documentation debt
Date/time	Apr 14, 2019 06:01
Context	TODO Auto-generated catch block
Propagation rules	Task tags (TODO, FIXME, etc.) can be harmful if there is no process to manage and get rid of them. If they accumulate over time, they are better placed into an issue tracker.
Intentionality	Unintentionally

TABLE VII. FASTJSON : TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	1.2.2 Documentation
Name	Comment completeness
Location	src/main/java/com/alibaba/fastjson/JSON.java:603
Responsible/author	Developer
Dimension	Documentation debt
Date/time	Apr 14, 2019 06:01
Context	Interface comment missing
Propagation rules	Missing interface documentation makes it hard to get a quick overview of the interface provided by a class or API.
Intentionality	Unintentionally

TABLE VIII. FASTJSON : TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	1.3.1 Structure
Name	File size
Location	src/main/java/com/alibaba/fastjson/JSON.java
Responsible/author	Developer
Dimension	Architectural debt
Date/time	Apr 14, 2019 06:01
Context	Violation of file size threshold (source lines of code) of 750: 842
Propagation rules	Long files can complicate both locating a specific feature within a file and understanding the consequences of a change. Ideally, a file should contain a clearly defined set of features. Very long files often indicate that too many features are intermixed in the code.
Intentionality	Intentionally

TABLE IX. FASTJSON : TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	1.3.2 Structure
Name	Nesting Depth
Location	src/main/java/com/alibaba/fastjson/JSONPath.java:1149-1150
Responsible/author	Developer
Dimension	Architectural debt
Date/time	Apr 14, 2019 06:01
Context	Violation of nesting depth threshold of 5: 6
Propagation rules	Deep nesting of control structures, such as conditionals and loops, complicate the understanding of the context and conditions holding at the deeply nested statements. Often the end of each nested construct is not easy to spot.
Intentionality	Intentionally

TABLE X. DRJAVA : TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	2.1.1 Code Anomalies
Name	Exception Handling
Location	src/edu/rice/cs/drjava/plugins/eclipse/repl/EclipseInteractionsModel.java:359
Responsible/author	Developer
Dimension	Code debt
Date/time	Apr 14, 2019 06:02
Context	Throw of generic exception RuntimeException
Propagation rules	Generic exceptions should not be thrown by methods. Appropriate subclasses should be used instead.
Intentionality	Intentionally

TABLE XI. DRJAVA : TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	2.1.2 Code Anomalies
Name	Exception Handling
Location	src/edu/rice/cs/drjava/plugins/eclipse/views/InteractionsController.java:139-143
Responsible/author	Developer
Dimension	Code debt
Date/time	Apr 14, 2019 06:02
Context	Catch clause catches generic exception Throwable
Propagation rules	Generic exceptions should not be handled by a catch block. For example, in Java, the exceptions Throwable, Error, RuntimeException, and Exception of the java.lang package count as generic.
Intentionality	Intentionally

TABLE XII. DRJAVA: TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	2.2.1 Documentation
Name	Comment completeness
Location	src/edu/rice/cs/drjava/plugins/eclipse/views/InteractionsController.java:711
Responsible/author	Developer
Dimension	Documentation debt
Date/time	Apr 14, 2019 06:02
Context	Interface comment missing
Propagation rules	Missing interface documentation makes it hard to get a quick overview of the interface provided by a class or API.
Intentionality	Unintentionally

TABLE XIII. DRJAVA : TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	2.2.2 Documentation
Name	Task tags
Location	src/edu/rice/cs/drjava/plugins/eclipse/repl/EclipseInteractionsModel.java:80
Responsible/author	Developer
Dimension	Documentation debt
Date/time	
Context	TODO: Read input from System.in
Propagation rules	Task tags (TODO, FIXME, etc.) can be harmful if there is no process to manage and get rid of them. If they accumulate over time, they are better placed into an issue tracker.
Intentionality	Unintentionally

TABLE XIV. DRJAVA : TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	2.3.1 Structure
Name	Nesting Depth
Location	src/edu/rice/cs/drjava/plugins/eclipse/util/text/SWTDocumentAdapter.java:187-192
Responsible/author	Developer
Dimension	Architectural debt
Date/time	Apr 14, 2019 06:02
Context	Violation of nesting depth threshold of 3: 4
Propagation rules	Deep nesting of control structures, such as conditionals and loops, complicate the understanding of the context and conditions holding at the deeply nested statements.
Intentionality	Intentionally

TABLE XV. DRJAVA: TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	2.3.2 Structure
Name	Method Length
Location	src/edu/rice/cs/drjava/plugins/eclipse/repl/EclipseInteractionsModel.java:296-361
Responsible/author	Developer
Dimension	Architectural debt
Date/time	Apr 14, 2019 06:02
Context	Violation of method length threshold (source lines of code) of 30: 32
Propagation rules	Long lists of statements in functions, methods, static initializers, or as top-level code complicate both code understanding and modification, as often the entire statement sequence has to be understood before introducing changes.
Intentionality	Intentionally

TABLE XVI. JFREECHART : TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	3.1.1 Code Anomalies
Name	Bad practice
Location	src/test/java/org/jfree/chart/StrokeMapTest.java:105
Responsible/author	Developer
Dimension	Code debt
Date/time	Apr 14, 2019 06:02
Context	m1 is compared with itself.
Propagation rules	Equals() called on the same object will always return true.
Intentionality	Intentionally

TABLE XVII. JFREECHART: TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	3.1.2 Code Anomalies
Name	Bad practice
Location	src/main/java/org/jfree/data/xy/DefaultWindDataset.java:331
Responsible/author	Developer
Dimension	Code debt
Date/time	Apr 14, 2019 06:02
Context	More than one top-level class in file
Propagation rules	Multiple top-level classes in a single file are possible in Java but make them hard to find for developers. Better use separate files, inner classes or anonymous classes.
Intentionality	Intentionally

TABLE XVIII. JFREECHART: TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	3.2.1 Documentation
Name	Comment completeness
Location	src/main/java/org/jfree/chart/editor/DefaultValueAxisEditor.java:363
Responsible/author	Developer
Dimension	Documentation debt
Date/time	Apr 14 2019 06:02
Context	Interface comment missing
Propagation rules	Missing interface documentation makes it hard to get a quick overview of the interface provided by a class or API.
Intentionality	Unintentionally

TABLE XIX. JFREECHART: TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	3.2.2 Documentation
Name	Task tags
Location	src/main/java/org/jfree/chart/plot/CombinedDomainCategoryPlot.java:614
Responsible/author	Developer
Dimension	Documentation debt
Date/time	Apr 14, 2019 06:02
Context	FIXME: this code means that it is not possible to use more than
Propagation rules	Task tags (TODO, FIXME, etc.) can be harmful if there is no process to manage and get rid of them. If they accumulate over time, they are better placed into an issue tracker.
Intentionality	Unintentionally

TABLE XX. JFREECHART: TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	3.3.1 Structure
Name	File Size
Location	src/main/java/org/jfree/chart/axis/ValueAxis.java
Responsible/author	Developer
Dimension	Architectural debt
Date/time	Apr 14, 2019 06:02
Context	Violation of file size threshold (source lines of code) of 750: 777
Propagation rules	Long files can complicate both locating a specific feature within a file and understanding the consequences of a change. Ideally, a file should contain a clearly defined set of features. Very long files often indicate that too many features are intermixed in the code.
Intentionality	Intentionally

TABLE XXI. JFREECHART: TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	3.3.2 Structure
Name	Method Length
Location	src/main/java/org/jfree/chart/axis/CategoryAxis.java: 1018-1112
Responsible/author	Developer
Dimension	Architectural debt
Date/time	Apr 14, 2019 06:02
Context	Violation of method length threshold (source lines of code) of 75: 92
Propagation rules	Long lists of statements in functions, methods, static initializers, or as top-level code complicate both code understanding and modification, as often the entire statement sequence has to be understood before introducing changes.
Intentionality	Intentionally

TABLE XXII. SUPER MARIO BROS: TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	4.1.1 Code Anomalies
Name	Exception Handling
Location	src/com/golden/gamedev/GameLoader.java:505-506
Responsible/author	Developer
Dimension	Code debt
Date/time	Apr 14, 2019 06:03
Context	Catch clause catches generic exception Exception
Propagation rules	Generic exceptions should not be handled by a catch block. For example, in Java, the

	exceptions Throwable, Error, RuntimeException, and Exception of the java.lang package count as generic.
Intentionality	Intentionally

TABLE XXIII. SUPER MARIO BROS: TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	4.1.2 Code Anomalies
Name	Null pointer dereference
Location	src/com/golden/gamedev/funbox/GameSettings.java: 413
Responsible/author	Developer
Dimension	Code debt
Date/time	Apr 14, 2019 06:03
Context	The variable 'fout' may contain a null value at this point and is dereferenced
Propagation rules	The variable that is de-referenced in this statement may contain a null value. This may lead to an exception being thrown at runtime.
Intentionality	Intentionally

TABLE XXIV. SUPER MARIO BROS: TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	4.2.1 Documentation
Name	Comment completeness
Location	src/Animations/Black.java:8
Responsible/author	Developer
Dimension	Documentation debt
Date/time	Apr 14, 2019 06:03
Context	Interface comment missing
Propagation rules	Missing interface documentation makes it hard to get a quick overview of the interface provided by a class or API.
Intentionality	Unintentionally

TABLE XXV. SUPER MARIO BROS: TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	4.2.2 Documentation
Name	Task tags
Location	src/javazoom/jl/decoder/Header.java:337
Responsible/author	Developer
Dimension	Documentation debt
Date/time	Apr 14, 2019 06:03
Context	TODO
Propagation rules	Task tags (TODO, FIXME, etc.) can be harmful if there is no process to manage and get rid of them. If they accumulate over time, they are better placed into an issue tracker.
Intentionality	Unintentionally

TABLE XXVI. SUPER MARIO BROS: TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	4.3.1 Structure
Name	File Size
Location	src/Levels/Five/Level_54.java
Responsible/author	Developer
Dimension	Architectural debt
Date/time	Apr 14, 2019 06:03
Context	Violation of file size threshold (source lines of code) of 750: 832
Propagation rules	Long files can complicate both locating a specific feature within a file and understanding the consequences of a change. Ideally, a file should contain a clearly defined set of features.
Intentionality	Intentionally

TABLE XXVII. SUPER MARIO BROS: TECHNICAL DEBT TYPE DIMENSIONM

Parameter	Value
ID	4.3.2 Structure
Name	Nesting Depth
Location	src/SandBox/Mario.java:1445
Responsible/author	Developer
Dimension	Architectural debt
Date/time	Apr 14, 2019 06:03
Context	Violation of nesting depth threshold of 5: 7
Propagation rules	Deep nesting of control structures, such as conditionals and loops, complicate the understanding of the context and conditions holding at the deeply nested statements.
Intentionality	Intentionally

REFERENCES

[1] Zadia Codabux, Byron J. Williams, Nan Niu, "A Quality Assurance Approach to Technical Debt"

[2] I. Allman, Eric (May 2012). "Managing Technical Debt". Communications of the ACM. 55 (5): 50–55. doi:10.1145/2160718.2160733

[3] https://github.com/alibaba/fastjson.git

[4] https://github.com/DrJavaAtRice/drjava

[5] https://github.com/jfree/jfreechart

[6] https://sourceforge.net/projects/super-mario-bros-java/files/

[7] https://www.sonarqube.org/

[8] https://www.cqse.eu/en/products/teamscale/landing/

[9] https://www.eclemma.org/jacoco/

[10] https://pmd.github.io/

[11] http://checkstyle.sourceforge.net/

[12] https://users.ece.cmu.edu/~koopman/des_s99/sw_reliability/

[13] https://www.seleniumhq.org/

[14] https://www.zaptest.com/

[15] http://www.squale.org/