Technical Debt Project Report

Ishita Kohli
Master of Computer Science
University of Ottawa
Ottawa, Canada

PART I. INTRODUCTION

Technical debt is a metaphor coined by Ward Cunningham. Just like financial debt, Technical debt comes in the form of extra effort in future development. It is the concept of software development where sometimes choosing an easy and dirty method instead of the better approach can lead to extra cost and effort for future. Causes of technical debt can be of two types i.e. intentional and unintentional. Intentional causes may include time constraints, source code complexity and Unintentional causes include lack of planning or coding standards. Presence of technical debt can lead to many software quality issues, maintenance problems or increased cost and effort. In order to get rid of such issues, various tools have been developed to measure, identify and prevent the accumulation of technical debt in the software development. In this project, I will be taking 4 open source projects and will use two tools i.e. Teamscale and Sonarqube to analyze the problems that lead to the accumulation of technical debt in the projects. The results of the analysis will be used to compare the projects with each other and to determine the usability of the tools.

PART II. MANAGING TECHNICAL DEBT USING TOOLS

1. Teamscale

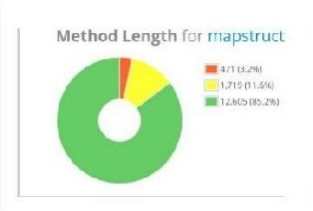
Team scale analyses the quality of code. With variety of static code analysis, it points you to quality defects that are easily missed. It analyses your code to identify specific maintainability constraints and avoid unexpected maintenance costs in the future. Teamscale helps to monitor the quality of the code over time and helps to prevent failures by continuously improving the software quality[1]. The various attributes which teamscale detects are : clone coverage, documentation, code quality, test gaps, coding conventions etc. The analysis of teamscale is shown as below:

LOC represents the total lines of code for the whole project.

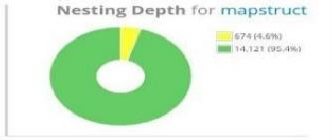
Clone Coverage represents the percentage of code found by duplication especially from cope and paste. The particular project shows 14.7% of duplication.

Findings count represents the number of anomalies found in the project. Here the anomalies are almost 1.6k which shows a huge amount of technical debt.

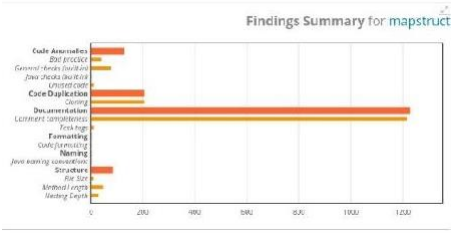
Method length describes if the methods are in optimal length range. Green describes normal, yellow describes not normal and red describes critical.



Nesting Depth checks for the nesting depth in project whether it is normal or not. Green represents normal, yellow represents not normal and red represents critical.

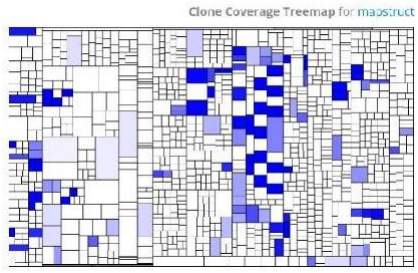


Findings Summary bar chart represents various findings and their quantity in the form of a bar chart.



LOC vs. Findings Trend helps to represent the history in which the lines of code evolved with respect to number of findings.

Tree Map tells about the clone coverage distributed in the code base. Every rectangles is the source code file and higher the blue the colour of rectangle leads to high clone.

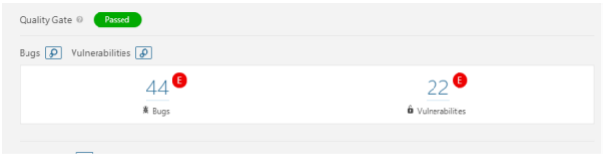


2. SonarQube

SonarQube is an open-source platform developed by SonarSource for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities on 20+ programming languages. It has an embedded feature called Quality Gate which can help you to fix the leak and hence results in improving the quality of the code[2]. The various of attributes analysed by sonarqube are code smells, duplications, coverage , reliability and security.

The analysis of sonarqube is shown as below:

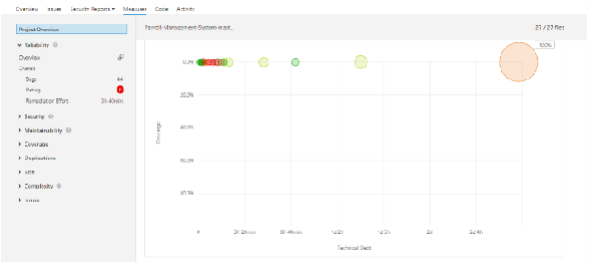
This represents the overvall quality of the project representing number of bugs and vulnerabilities.



The particular section tells us about the code smells i.e. the smells raised due to poor coding practices.

The issues page in sonarqube tells you about the overall issues in the project. It tells us who introduced the issues and give us the full power to correct them.

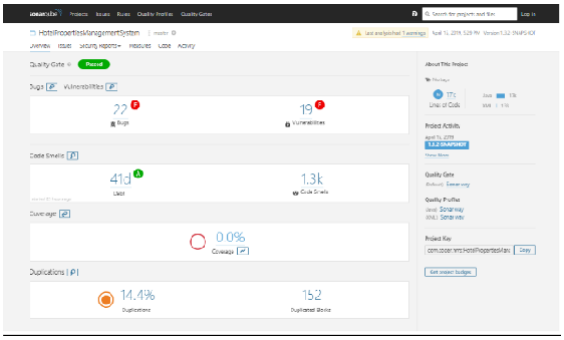
This feature of sonarqube tells us about the overall reliability, duplications in the project and how they evolved over time.



3. Quality Assessment

3.1 Project :Hotel Properties Management System

3.1.1 Analysis by SonarQube



Hotel properties managing system application is made for managing the hotel activities It operates independently of the operating system (Made by Java & Maven & Hibernate & Mysql & Jaspy Security). The project has 104 classes. The above analysis of the project shows overall bugs and vulnerabilities. The project has 17k lines of code. There are 22 bugs and 19 vulnerabilities in the code. The letter and the colour represents the the severity of the issue. Green represents for normal, yellow represents not normal and red represents critical. Here, the colour is red which means the issue is critical in this project. Code smells are introduced in 1.3k lines of code. Debt occurs for 41days of the life time of the project. This issue is represented with green colour which means that it is normal. There are 14.4% of duplications which means 152 block of code represents duplicated code.

SonarQube helps in accessing various quality attributes like reliability, maintainability and security. They are depicted below for the hotel management project:

Reliability Overview:

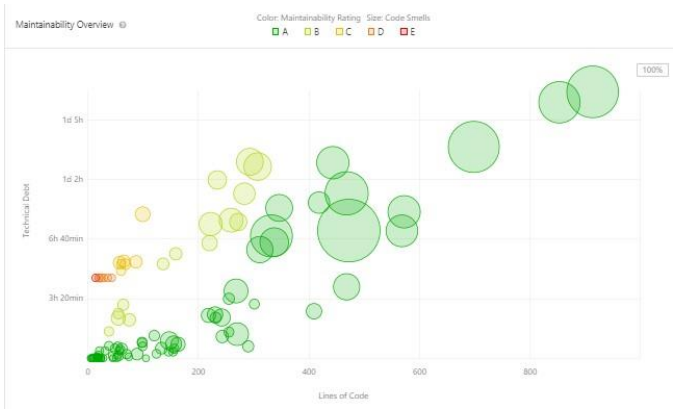
Here the green bubble represents that there is no issue whereas the red bubble represents the critical issues. However, there are 3 classes with 3 critical issues which prevail for first 5 min and then 15 minutes.

Security Overview:

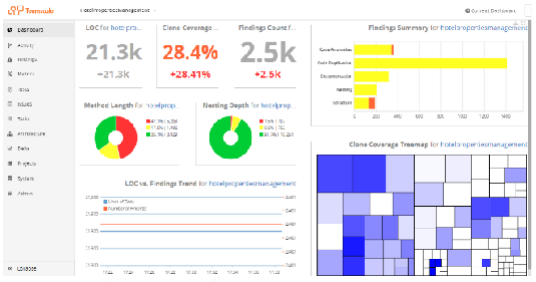
The overall effort required is for 3h 10min and the vulnerabilities are 19 in number which depicts the critical issue in the project.

Maintainability Overview:

There are 1304 code smells which require maintenance . However, the debt stays for 41days and the overall rating is good as most of the bubbles seen are green in color.



3.1.2 Analysis by Teamscale



The analysis of the project by teamscale shows 2.5k lines of code that can lead to the accumulation of technical debt. Out of which code duplication is one of the main reason for the accumulation of technical debt. About 56.8% i.e. 1414 finding represents the debt by duplication. 317 findings (12.7%) is caused by documentation and 13.8% due to code anomalies and 28.4% of debt is caused by clone coverage. This high number

of duplications and code anomalies makes it difficult to modify the code. Main cause of code duplication is redundant literals. Code anomalies is caused due to bad coding practices like if statement without braces, parameter number and type differs. Issue in documentation is caused mainly by interface comment missing. These issues causes the readability problems and thus decreases the modularity of the code.

3.1.3 Technical Debt Identification

In this project, Teamscale can analyse both code and documentation debt. Sonarqube is used to analyse code smells and its history for the evolution of code smell. Various item types that were identified are as follows :

- **Code Debt:** Code duplication like redundant string literals, redundancy clones, code anomalies like code must not only differ in casing, unnecessary parameter less super constructor call, missing braces for block statements.

- **Documentation Debt:** Most of the documentation debt is due to comments/missing interface comments and comments/task tags.

3.1.4 Technical Debt Representation

Table column name	Table Column Value
ID:	1
Name:	Code Debt
Location:	src/com/coder/hms/beans/RoomCountRow.java:79
Responsible /author:	Coder ACJHP
Dimension:	Code Debt
Date/time:	April 15, 2019, 5:29 PM
Context:	Code Duplication
Propagation rules:	Redundant string literals should be extracted to a constant if they will change consistently.
Intentionality	Unintentional

Table column name	Table Column Value
ID:	2
Name:	Code Debt
Location:	src/com/coder/hms/beans/Blockade.java:58
Responsible /author:	Coder ACJHP
Dimension:	Code Debt
Date/time:	April 15, 2019, 5:29 PM
Context:	Names must not only differ in casing
Propagation rules:	Identifiers in the code must not differ only in their casing , as they can lead to confusion.
Intentionality	Unintentional

Table column name	Table Column Value
ID:	3
Name:	Code Debt
Location:	src/com/coder/hms/beans/RoomCountRow.java:68
Responsible /author:	Coder ACJHP
Dimension:	Code Debt
Date/time:	April 15, 2019, 5:29 PM
Context:	Unnecessary parameterless super constructor call
Propagation rules:	Super constructors with no parameters need not to be explicitly called.
Intentionality	Unintentional

Table column name	Table Column Value
ID:	4
Name:	Code Debt
Location:	src/com/coder/hms/entities/Company.java:208
Responsible/ author:	Coder ACJHP
Dimension:	Code Debt
Date/time:	April 15, 2019, 5:29 PM
Context:	Missing braces for block statements
Propagation rules:	Creates findings for statement blocks in loops
Intentionality	Intentional

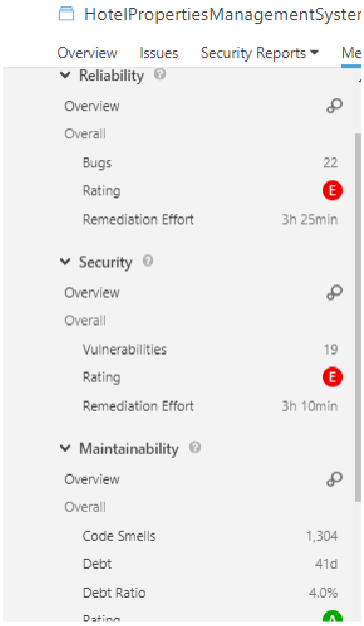
Table column name	Table Column Value
ID:	5
Name:	Documentation Debt
Location:	src/com/coder/hms/beans/Blockade.java:58
Responsible /author:	Coder ACJHP
Dimension:	Documentation Debt
Date/time:	April 15, 2019, 5:29 PM
Context:	Interface Comment Missing
Propagation rules:	Missing interface documentation makes it hard to get a quick overview of the interface.
Intentionality	Unintentional

Table column name	Table Column Value
ID:	6
Name:	Documentation Debt
Location:	src/com/coder/hms/beans/Blockade.java:23
Responsible /author:	Coder ACJHP
Dimension:	Documentation Debt
Date/time:	April 15, 2019, 5:29 PM
Context:	Task Tags
Propagation rules:	Task tags can be harmful if there is not process to get rid of them, better it is placed in issue tracker
Intentionality	Unintentional

3.1.5 Technical Debt Estimation

It is found that there are various code smells, vulnerabilities and bugs in the project that leads to the accumulation of TD.

SonarQube can be used to estimate the technical debt. The estimation is generally measured in terms of effort. It is found that the effort required to fix any bug in the project took 3h 25min effort, in order to remove vulnerabilities an effort of 3h 10 min is required. For a whole code smell an effort of almost 41 days is required to remove the technical debt. Therefore, an extra time of 6h 35min is required to fix both bugs and vulnerabilities.



3.1.6 Technical Debt Monitoring

Monitoring of technical debt is done using teamscale. Team scale helps to depict the lines of code vs. Findings Trend which is depicted in the graph above. The blue line depicts the lines of code with respect to the left bar and the red line depicts the number of findings with respect to the right bar. It is depicted that the lines of code and number of findings have been stable from 17:22 – 17:38. However, the number of findings are found ahead of the lines of code. For a good quality code and TD free code red line should always be below the blue line with some gap. As the value of both the variable draws closer, it can be said the quality of code is deteriorated and some quality measures to reduce the TD must be taken immediately. On the basis of this information various approaches could be used like threshold can be set of findings with respect to lines of codes. While monitoring we can track the existing number of findings and new findings generated on addition of new code and decide to whether work on it or not.

3.1.7 Technical Debt Repayment

Technical debt repayment is one of significant TD Management techniques because paying back the principal will keep technical debt under control. It also allows the programmer to focus on other issues such as developing the software or adding new features. There can be certain ways to repay the technical debt.

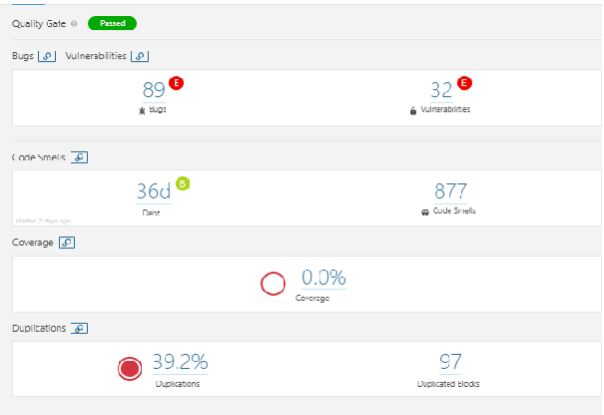
- Refactoring
- Extract Method

Refactoring : Most of the debt occurs due to code smells in the project. Therefore, there is a need to refactor these code smells. Developers should try to minimize the code duplications in order to remove the duplications from the code. Developer should try to improve various bad practises used to prevent the occurrence of code anomalies.

Extract Method: In order to prevent the code duplication there is a need to extract the method into separate methods. This may cause the insertion of some external utility class that has a method.

3.2 Project :Library Management System

3.2.1 Analysis by SonarQube

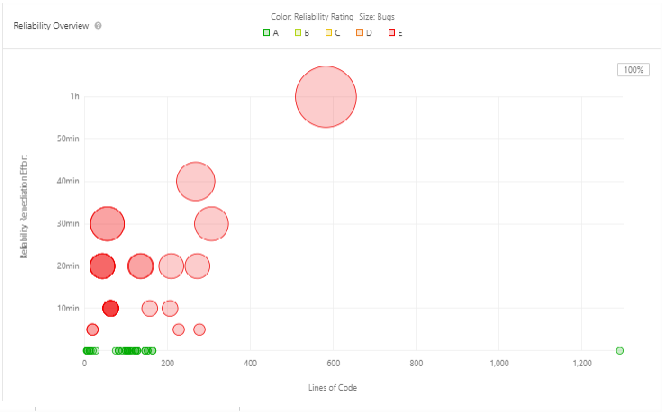


Library Management System is a java application made to analyse the activities of a library. The project has almost 8k lines of code with 61 files and 56 classes. The tool shows that there 89 critical bugs and 32 critical vulnerabilities. The code smells have a debt of 36 days with almost 877 code smells. The percentage of duplication present is 39.2% and there are 97 duplicated blocks of code.

Talking about the quality attributes , SonarQube analyses 3 main attributes such as reliability, maintainability and security which depicted as follows:

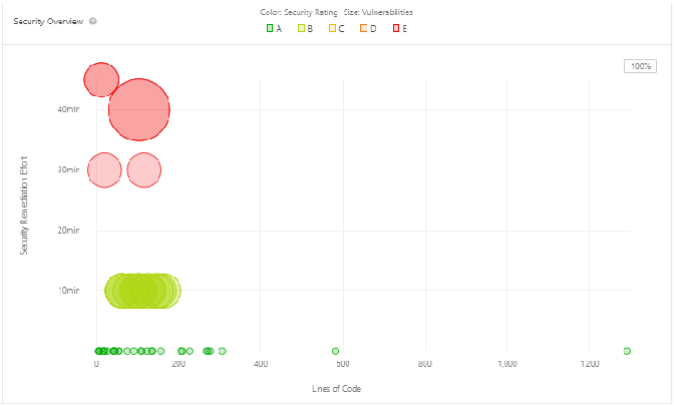
Reliability Overview:

The reliability overview states that there are almost 89 critical bugs which require a remediation effort of 7h 30 min. The above graph depicts as the number of lines of code increases the criticalness of the issue along with effort goes on increasing.



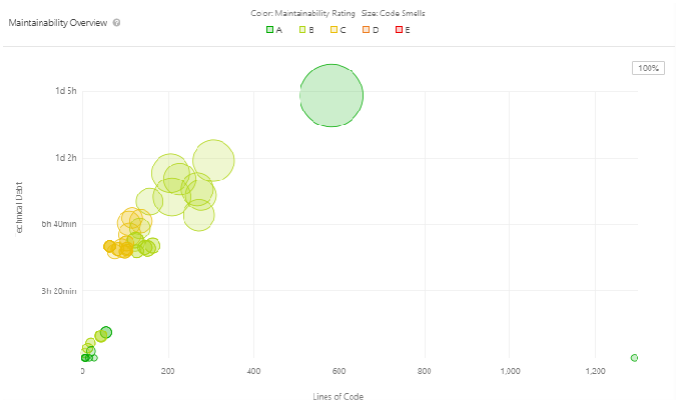
Security Overview :

For the security quality attribute there are almost 32 critical bugs which require the remediation effort of 7h 50min.

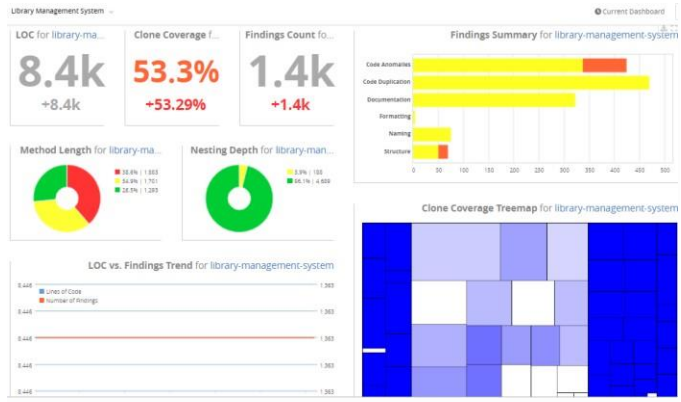


Maintainability Overview:

This graphs shows that 877 code smells need maintenance which has a debt of 35 days. Here, the debt ratio is 7.2% with a rating of B which means the rating is good as green bubbles can be seen. It requires an effort of 10 days to reach the rating of A i.e. totally free of issues.



3.2.2 Analysis by Teamscale



The library management project on analysis with teamscale depicts that there are 1.4k lines of code which leads to the accumulation of technical debt. Most of debt is due to code debt i.e. code duplication and code anomalies. Documentation debt also plays another role in contributing to technical debt. Most of the code duplication is due to redundancy of string literals or clones. Various code anomalies include bad practices such as call to printStackTrace() or static variables not properly initialized. The clone coverage for the project is 53.3%.

3.2.3 Technical Debt Identification

In this project, Teamscale can analyse both code and documentation debt. Sonarqube is used to analyse code smells and its history for the evolution of code smell. Various item types that were identified are as follows :

- **Code Debt:**
  Various attributes contributing to code debt are redundant string literals, clones, unused or duplicated imports, static variable not initialized.

- **Documentation Debt:**
  Two main attributes contributing to documentation debt are interface comment missing and task tags. These are the tags leading to documentation debt.

3.2.4 Technical Debt Representation

Table column name	**Table Column Value**
ID:	1
Name:	Code Debt
Location:	src/AdminSuccess.java:52
Responsible /author:	Not Assigned
Dimension:	Code Debt
Date/time:	April 15, 2019, 1:40AM
Context:	Redundant string literals
Propagation rules:	Redundant string literals should be expected to change into constant if they change consistently
Intentionality	Unintentional

Table column name	**Table Column Value**
ID:	2
Name:	Code Debt
Location:	src/AdminLogin.java:21
Responsible /author:	Not Assigned
Dimension:	Code Debt
Date/time:	April 15, 2019, 1:40AM
Context:	Properly initialize static variable
Propagation rules:	Static variables that are not private or protected should be initialized directly or by static initializer
Intentionality	Unintentional

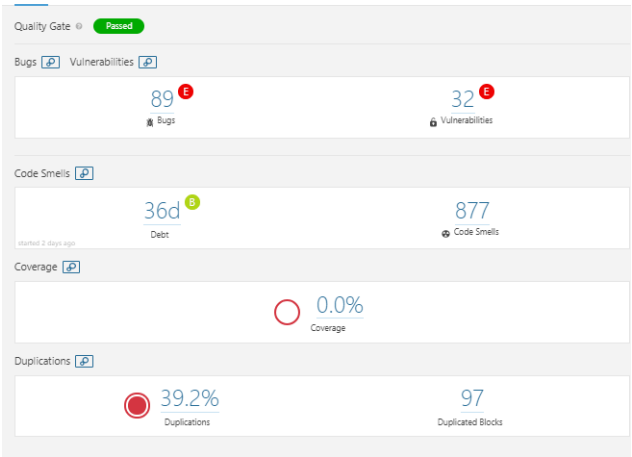
Table column name	**Table Column Value**
ID:	3
Name:	Code Debt
Location:	src/AdminLogin.java:36
Responsible /author:	Not Assigned
Dimension:	Code Debt
Date/time:	April 15, 2019, 1:40AM
Context:	Call to printStackTrace()
Propagation rules:	It is recommended to use logging framework
Intentionality	Unintentional

Table column name	Table Column Value
ID:	4
Name:	Code Debt
Location:	src/AdminSuccess.java:52
Responsible /author:	Not Assigned
Dimension:	Code Debt
Date/time:	April 15, 2019, 1:40AM
Context:	Unwanted method calls
Propagation rules:	Debugging methods should be avoided because they introduce unwanted side effects
Intentionality	Unintentional

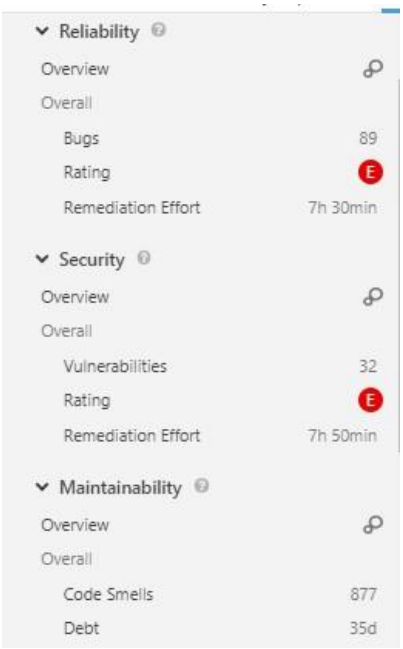
Table column name	Table Column Value
ID:	5
Name:	Documentation Debt
Location:	src/Library/AddQuestion.java:277
Responsible /author:	Not Assigned
Dimension:	Documentation Debt
Date/time:	April 15, 2019, 1:40AM
Context:	Task Tags
Propagation rules:	Their accumulation is not preferred, therefore, its better to place them in issue tracker.
Intentionality	Unintentional

Table column name	Table Column Value
ID:	6
Name:	Documentation Debt
Location:	src/AdminSuccess.java:26
Responsible /author:	Not Assigned
Dimension:	Documentation Debt
Date/time:	April 15, 2019, 1:40AM
Context:	Interface comment missing
Propagation rules:	Missing interface documentation makes it hard to get the overview of the API or class.
Intentionality	Unintentional

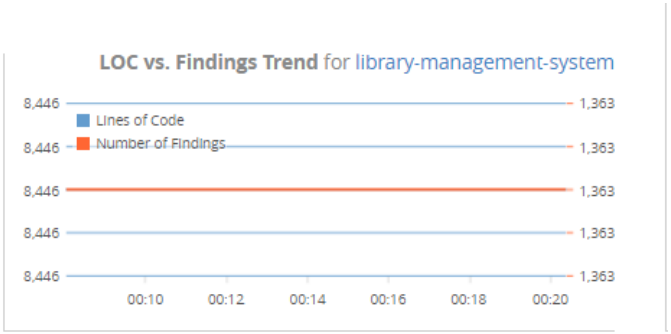
3.2.5 Technical Debt Estimation



For the estimation of technical debt with sonarqube, it is found that there are 89 critical bugs which requires an effort of 7h 30 min. There are 32 critical vulnerabilities which requires 7h 50 min effort. There are 877 code smells which have a debt of 36 days. Extra effort required is for bugs and vulnerabilities. Almost extra 15hr is spend on them. If $500 is taken as the cost for person-day then for extra 15hours it would require 7500$ to fix the code smells and bugs.



3.2.6 Technical Debt Monitoring



Monitoring of technical debt is done using teamscale. Team scale helps to depict the lines of code vs. Findings Trend which is depicted in the graph above. The blue line depicts the lines of code with respect to the left bar and the red line depicts the number of findings with respect to the right bar. It is depicted that there are 8446 lines of code and 1363 findings which have been stable all the time. Both the metrics are quiet consistent with respect to each other. For a good quality code and TD free code red line should always be below the blue line with some gap. As the value of both the variable draws closer, it can be said the quality of code is deteriorated and some quality measures to reduce the TD must be taken immediately. On the basis of this information various approaches could be used like threshold can be set of findings with respect to lines of codes. While monitoring we can track the existing number of findings and new findings generated on addition of new code and decide to whether work on it or not.

3.2.7 Technical Debt Repayment

Technical debt repayment is one of significant TD Management techniques because paying back the principal will keep technical debt under control. It also allows the programmer to focus on other issues such as developing the software or adding new features. There can be certain ways to repay the technical debt.

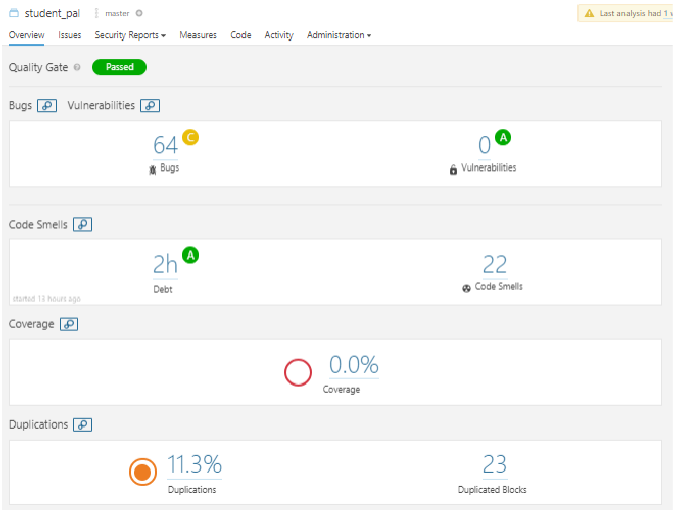
- Refactoring
- Extract Method

Refactoring : Most of the debt occurs due to code smells in the project. Therefore, there is a need to refactor these code smells. Developers should try to minimize the code duplications in order to remove the duplications from the code. Developer should try to improve various bad practises used to prevent the occurrence of code anomalies.

Extract Method: In order to prevent the code duplication there is a need to extract the method into separate methods. This may cause the insertion of some external utility class that has a method.

3.3 Project: Student Pal

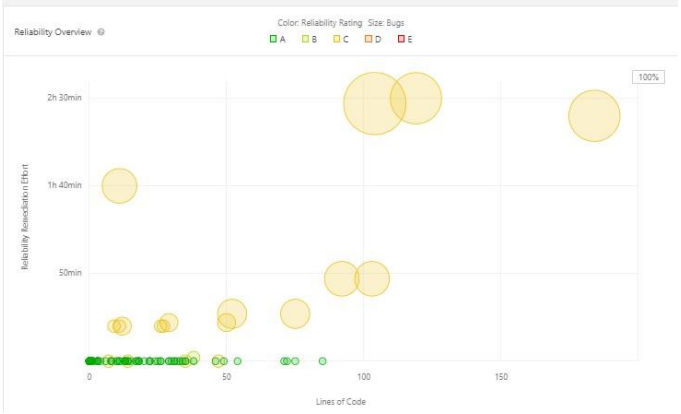
3.3.1 Analysis by SonarQube



Student Pal is a python project developed to manage the activities of students. It has almost 2400 lines of code with 56 classes. Through SonarQube it is found that there are 64 bugs in the project which are not that critical and 0 vulnerabilities which is good for the developer. It would require less time to remove the technical debt as there are less bugs and vulnerabilities. The code smells are almost 22 in number which occurs for almost 2 hours. The number of duplications are 11.3% with just 23 duplicated blocks of code.

Talking about the quality attributes , SonarQube analyses 3 main attributes such as reliability, maintainability and security which depicted as follows:

Reliability Overview :

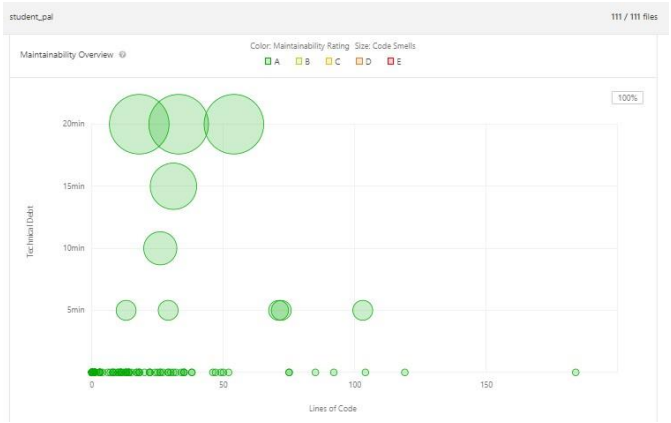


Reliability is seen as the worst in the project so far as the number of bugs are more with rating C which require a remediation effort of 1d 5h.

Security Overview :

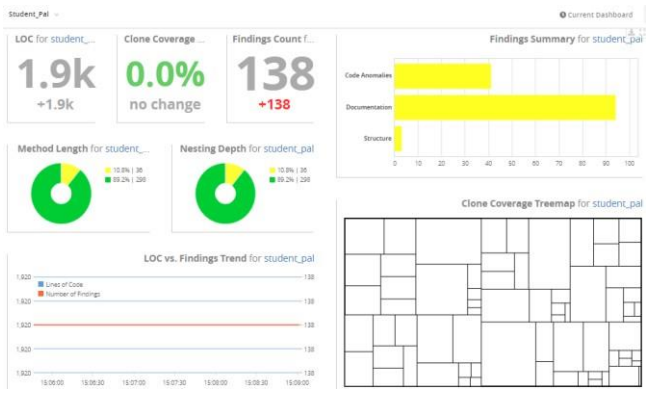
Since the number of vulnerabilities are 0 therefore the effort required to fix it is 0 hence all green bubbles. This shows security is best quality attribute in the project with no critical issues.

Maintainability Overview:



Code smells are less that is. the number is just 22 with A rating. That means the effort required for maintenance is very less but still more than security.

3.3.2 Analysis by TeamScale



Analysis by teamscale shows that there 138 lines of code which can lead to the accumulation of technical debt. Most of the debt is due to documentation and code anomalies and a very less portion by structure. Various items contributing to documentation debt are interface comment missing which is found in many files.. Code anomalies is caused by bad practices i.e. due to the assignment of the variable to itself. Documentation contributes to 68.1% of the total debt and code anomalies contributes to 29.7% of the debt. Rest is contributed by structure debt.

3.3.3 Technical Debt Identification

In this project, Teamscale can analyse both code and documentation debt. Sonarqube is used to analyse code smells and its history for the evolution of code smell. Various item types that were identified are as follows :

- **Code Debt:**
  Various attributes contributing to code debt are when variables are assigned to itself like group assigned to itself, pk assigned to itself etc.

- **Documentation Debt:**
  The major attributed leading to documentation debt is interface comment missing which have been seen in many files.

3.3.4 Technical Debt Representation

Table column name	Table Column Value
ID:	1
Name:	Code Debt
Location:	groups/views.py:59
Responsible /author:	Not Assigned
Dimension:	Code Debt
Date/time:	April 15, 2019, 1:39AM
Context:	Assignment of a variable to itself
Propagation rules:	Assignment of variables can hint at bugs.
Intentionality	Unintentional

Table column name	Table Column Value
ID:	2
Name:	Code Debt
Location:	groups/views.py:37
Responsible /author:	Not Assigned
Dimension:	Code Debt
Date/time:	April 15, 2019, 1:39AM
Context:	Assignment of a variable to itself
Propagation rules:	Assignment of variables can hint at bugs.
Intentionality	Unintentional

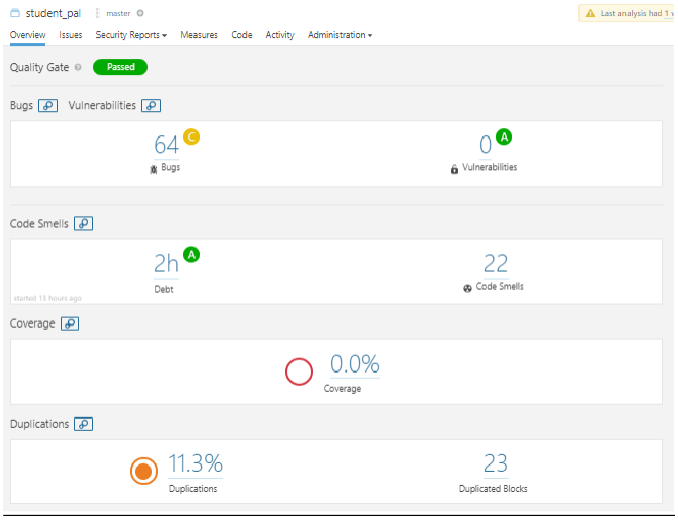
Table column name	Table Column Value
ID:	3
Name:	Code Debt
Location:	groups/views.py:65
Responsible /author:	Not Assigned
Dimension:	Code Debt
Date/time:	April 15, 2019, 1:39AM
Context:	Assignment of a variable to itself
Propagation rules:	Assignment of variables can hint at bugs.
Intentionality	Unintentional

Table column name	Table Column Value
ID:	4
Name:	Documentation Debt
Location:	groups/models.py:20
Responsible /author:	Not Assigned
Dimension:	Documentation Debt
Date/time:	April 15, 2019, 1:39AM
Context:	Interface Comment Missing
Propagation rules:	Assignment of variables can hint at bugs.
Intentionality	Unintentional

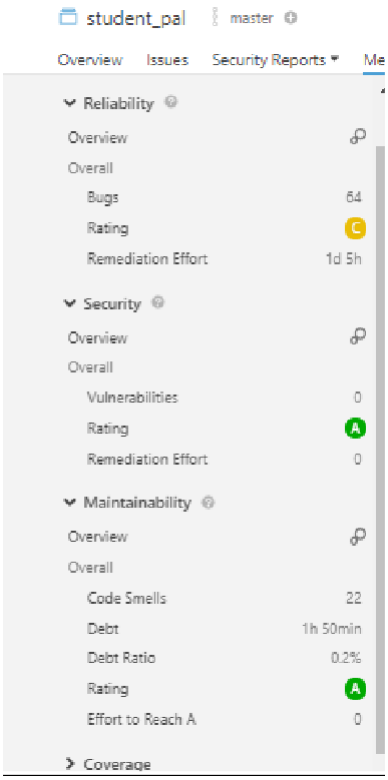
Table column name	Table Column Value
ID:	5
Name:	Documentation Debt
Location:	groups/models.py:23
Responsible /author:	Not Assigned
Dimension:	Documentation Debt
Date/time:	April 15, 2019, 1:39AM
Context:	Interface Comment Missing
Propagation rules:	Assignment of variables can hint at bugs.
Intentionality	Unintentional

Table column name	Table Column Value
ID:	6
Name:	Documentation Debt
Location:	groups/models.py:27
Responsible /author:	Not Assigned
Dimension:	Documentation Debt
Date/time:	April 15, 2019, 1:39AM
Context:	Interface Comment Missing
Propagation rules:	Assignment of variables can hint at bugs.
Intentionality	Unintentional

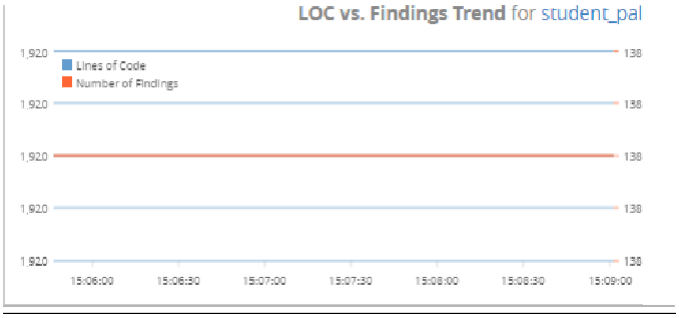
3.3.5 Technical Debt Estimation



SonarQube helps to estimate the total effort required to fix the issue. The effort is usually measured in terms of person per hour* total money required for one hour. In order to estimate technical debt one needs to know the effort spent on dealing with various issues like bugs, vulnerabilities and code smells. Here, the bugs are 64 in number which requires a effort of 1day 5 hours. However, the effort required to fix vulnerabilities is almost negligible i.e. 0 for vulnerabilities and 1hour spent for fixing code smells.



3.3.6 Technical Debt Monitoring



Monitoring of technical debt is done using teamscale. Team scale helps to depict the lines of code vs. Findings Trend which is depicted in the graph above. The blue line depicts the lines of code with respect to the left bar and the red line depicts the number of findings with respect to the right bar. It is depicted that there are 1920 lines of code and 158 findings which have been stable all the time. They haven't changed significantly over period of time. Both the metrics are quiet consistent with respect to each other. For a good quality code and TD free code red line should always be below the blue line with some gap. As the value of both the variable draws closer, it can be said the quality of code is deteriorated and some quality measures to reduce the TD must be taken immediately. On the basis of this information various approaches could be used like threshold can be set of findings with respect to lines of codes. While monitoring we can track the existing number of findings and new findings generated on addition of new code and decide to whether work on it or not.

3.3.7 Technical Debt Repayment

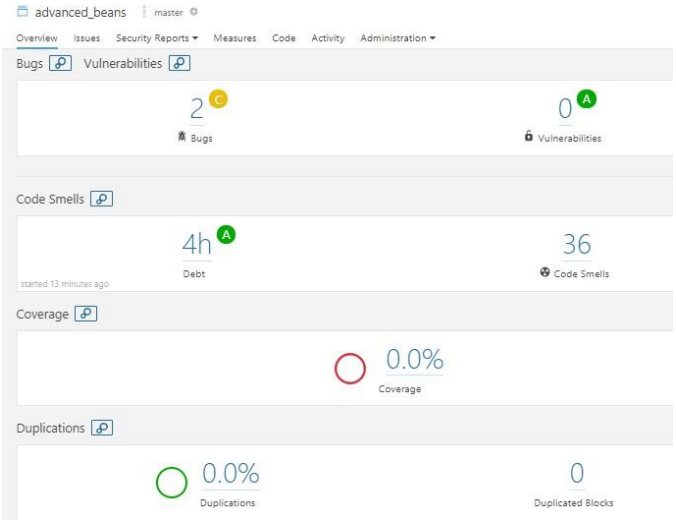
Technical debt repayment is one of significant TD Management techniques because paying back the principal will keep technical debt under control. It also allows the programmer to focus on other issues such as developing the software or adding new features. There can be certain ways to repay the technical debt.

- Refactoring

Refactoring : Most of the debt occurs due to code smells in the project. Therefore, there is a need to refactor these code smells. Developers should try to minimize the code duplications in order to remove the duplications from the code. Developer should try to improve various bad practises used to prevent the occurrence of code anomalies.

3.4 Project: Advanced Beans

3.4.1 Analysis by SonarQube



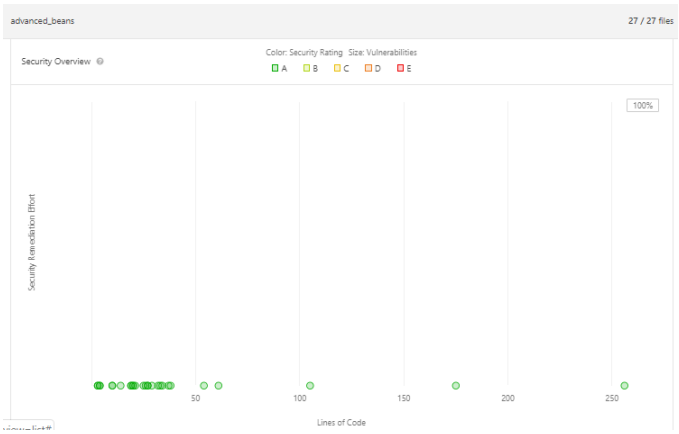
The advanced beans is a small project made in PHP which is basically a website panel for advanced ban. The project is quiet small with 1100 lines of code and 15 classes. On analysis with sonarqube, it is found that there are 2 bugs and 0 vulnerabilities in the code. The code smells are 36 in number which has a debt of almost 4h. There is zero coverage and zero duplication in the code.

Talking about the quality attributes , SonarQube analyses 3 main attributes such as reliability, maintainability and security which depicted as follows:

Reliability Overview :

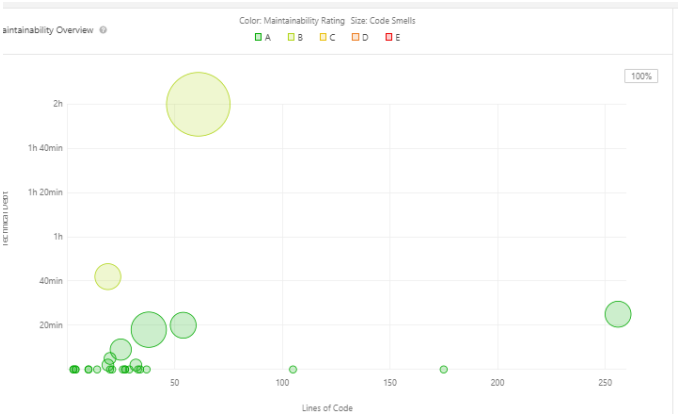
It is observed that there are 2 bugs in the projects which is found in 2 classes. These classes requires 5 min and 15 min effort to deal with the issue.

Security Overview:



There is no vulnerability in the code thus the remediation effort required is 0 and the code is free from any issues.

Maintainability Overview:



As far as maintainability is concerned, there are 36 code smells in the project out of which the maximum effort required is 2h in one class. Other classes requires 20 min or 40 min. However, due to 0 vulnerabilities security is found the best quality attribute in the particular project.

3.4.2 Analysis by Teamscale

The analysis of the particular project is also done using Teamscale. It is found that there are 23 lines of code which leads to the accumulation of technical debt. Maximum percentage of debt is found due to code anomalies (52.2%) and other by formatting (39.1%). Rest of the debt is present due to code duplication. The clone coverage required for the project is 6.2% which means the 2 blue rectangles representing 2 classes which are shown in clone coverage tree map requires clone coverage.

3.4.3. Technical Debt Identification

In this project, Teamscale can analyse code and formatting debt. Sonarqube is used to analyse code smells and its history for the evolution of code smell. Various item types that were identified are as follows :

- **Code Debt:**
  Various attributes contributing to code debt are Assignment of a variable to itself, missing braces for block statements and due to redundancy or clones.

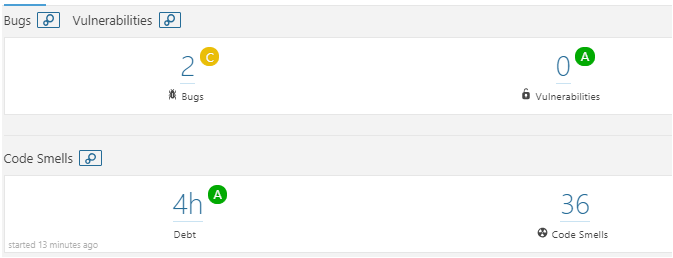
3.4.3 Technical Debt Representation

Table column name	Table Column Value
ID:	1
Name:	Code Debt
Location:	AdvancedBans/User/Language.class.php:9-25
Responsible /author:	Not Assigned
Dimension:	Code Debt
Date/time:	April 16, 2019, 1:30PM
Context:	Redundancy / Clones
Propagation rules:	Accumulation of clones can occur by copy & paste.
Intentionality	Unintentional

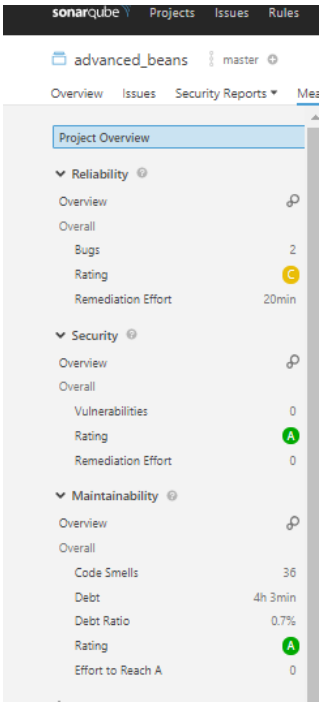
Table column name	Table Column Value
ID:	2
Name:	Code Debt
Location:	AdvancedBans/AdvancedBans.class.php:31
Responsible /author:	Not Assigned
Dimension:	Code Debt
Date/time:	April 16, 2019, 1:30PM
Context:	Assignment of variable to itself
Propagation rules:	Assignment of such variables can lead to bugs
Intentionality	Unintentional

Table column name	Table Column Value
ID:	3
Name:	Code Debt
Location:	AdvancedBans/AdvancedBans.class.php:46
Responsible /author:	Not Assigned
Dimension:	Code Debt
Date/time:	April 16, 2019, 1:30PM
Context:	Missing braces for block statements
Propagation rules:	Creates findings for statement blocks in loops and conditional statements that are not enclosed in braces - '{' and '}'.
Intentionality	Unintentional

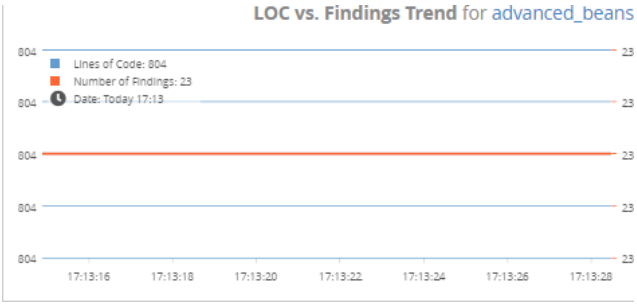
3.4.4 Technical Debt Estimation



SonarQube helps to estimate the total effort required to fix the issue. The effort is usually measured in terms of person per hour* total money required for one hour. In order to estimate technical debt one needs to know the effort spent on dealing with various issues like bugs, vulnerabilities and code smells. Here, the bugs are 2 in number which requires a effort of 20 min. However, the effort required to fix vulnerabilities is almost negligible i.e. 0 for vulnerabilities and 4hour 30 min spent for fixing 36 code smells. The debt ratio is 0.7% with a rating of A which quiet good.



3.4.5 Technical Debt Monitoring



Monitoring of technical debt is done using teamscale. Team scale helps to depict the lines of code vs. Findings Trend which is depicted in the graph above. The blue line depicts the lines of code with respect to the left bar and the red line depicts the number of findings with respect to the right bar. It is depicted that there are 804 lines of code and 23 findings which have been stable all the time. They haven't changed significantly over period of time. Both the metrics are quiet consistent with respect to each other. For a good quality code and TD free code red line should always be below the blue line with some gap. As the value of both the variable draws closer, it can be said the quality of code is deteriorated and some quality measures to reduce the TD must be taken immediately. On the basis of this information various approaches could be used like threshold can be set of findings with respect to lines of codes. While monitoring we can track the existing number of findings and new findings generated on addition of new code and decide to whether work on it or not.

3.4.6 Technical Debt Repayment

Technical debt repayment is one of significant TD Management techniques because paying back the principal will keep technical debt under control. It also allows the programmer to focus on other issues such as developing the software or adding new features. There can be certain ways to repay the technical debt.

- Refactoring

Refactoring : Most of the debt occurs due to code smells in the project. Therefore, there is a need to refactor these code smells. Developers should try to minimize the code duplications in order to remove the duplications from the code. Developer should try to improve various bad practises like redundancy, assignment of a variable to itself to prevent the occurrence of code anomalies.

4. TECHNICAL DEBT PREVENTION

The tools used for analysing the projects do help in preventing the technical debt. I used two tools that is teamscale and sonarqube. The tools used alone are not that effective but if used complimentary with each other can help in the prevention of accumulation of technical debt. SonarQube provides the deep information regarding the technical debt that is using various metrics like reliability, security and maintainability one gets to know how much remediation effort is required to fix the issues. SonarQube gives us the true feedback regarding the quality of our code in one place on our dashboard. One gets to know regarding the number of days of effort required to fix the issue. Teamscale on the other hand gives the full insight of every commit made to the project. If the development team use this tool right from the starting of development, teamscale would inform them regarding the every commit made from the start of the project. It tells us about the full details of code debt, documentation debt etc.

5. DISCUSSION

The first major challenge which I found while doing this project was that I was not able to find some standard tool which could give me all the details at once. I researched a bit and found different tools for different purposes but not a single tool which covers all the aspects of technical debt. The way I overcome with it was that I researched a bit about it and came up with two common tools that is Teamscale and Sonarqube. Sonarqube tells

us about the details of the debt and the remediation effort needed for the issues. Teamscale on the other hand gives an insight regarding the code debt , documentation debt, naming or formatting debts.

I found this project quiet interesting because it literally helped me to get hands on experience on these tools. I never thought about the technical debt in my projects before but after this course and this practical project, I.e. after analysing four open source projects, I got to realize how much technical debt is important in any project. Thus, developers have to be very careful while developing their code because sometimes unintentionally one has to pay a lot of money in order to get rid of the technical debt.

6. EVALUATION OF QUALITY ASSESSMENT

On evaluating the above projects it is found that Library Management System is the one which is most deprecated. There are 89 bugs , 32 vulnerabilities and high duplication of 39.2%. This leads to critical issues in the project.

Part III

7. CALCULATION MODEL

Calculation model is similar to the SonarQube calculation model which estimates the principal incurred in any project. It tells us about the rough idea of how much effort is required to fix the bugs. According to the SonarQube method, the principal is calculated as follows:

Debt= duplications + violations + comments + coverage + complexity where

- Code Duplication: Effort required to eliminate duplicate code from the system
- Violations: Effort required to eliminate violations from the system
- Comments: Effort required to document the undocumented part of the code.
- Coverage: effort required to bring test coverage up to 80%.
- Complexity: Effort required to split complex class and methods.

The formula to calculate the principal is as follows :

Duplication = cost_to_fix_one_block * duplicated_blocks

Violations = cost_to_fix_one_violation * mandatory_violations

Comments=cost_to_comment_one_API* public_undocumented_API

Coverage = cost_to_cover_one_of_complexity *uncovered_complexity_by_tests

Complexity = cost_to_split_a_method * (function_complexity_distribution ≥ 8) + cost_to_split_a_class * (class_complexity_distribution ≥ 60)

In my opinion , **architecture** is one of the important things while calculating technical debt.

So I would propose to add two more dimensions while calculating the principal that is Fan In and Fan Out which would focus on code and architecture both where

Fan In : Number of other classes that reference a class.

Fan Out: Number of other classes referenced by a class.

Thus, debt= duplications + violations + comments + coverage + complexity + Fan In + Fan Out

Effort required to fix single unit of each type will be in terms of person – hours. So, if calculated debt is divided by 8 will give Total estimated.

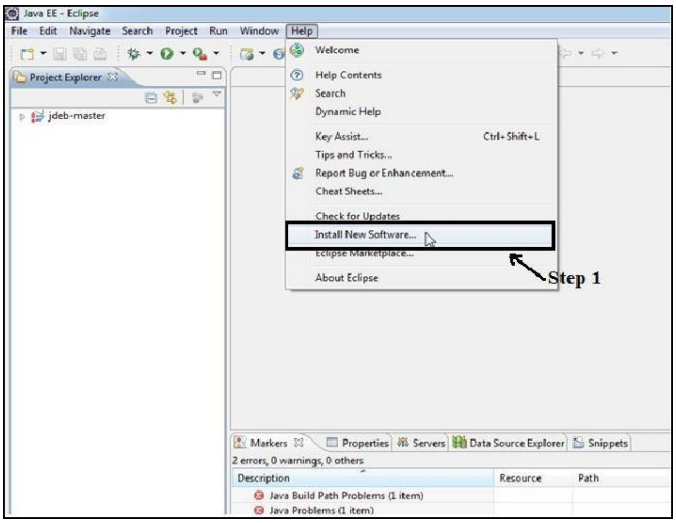
8. MORE TOOLS TO MANAGE TD

Apart from the mentioned tools , I would like to propose another tool called **"CodePro Analytix"** [3] . It is a free static code analysis tool offered by Google which aids in improving software quality. Google's CodePro offering is powered by Eclipse technology and includes Eclipse plug-ins that can be installed and used with the IBM Software Development Platform, an Eclipse-based offering. Various features of the tool is as below :

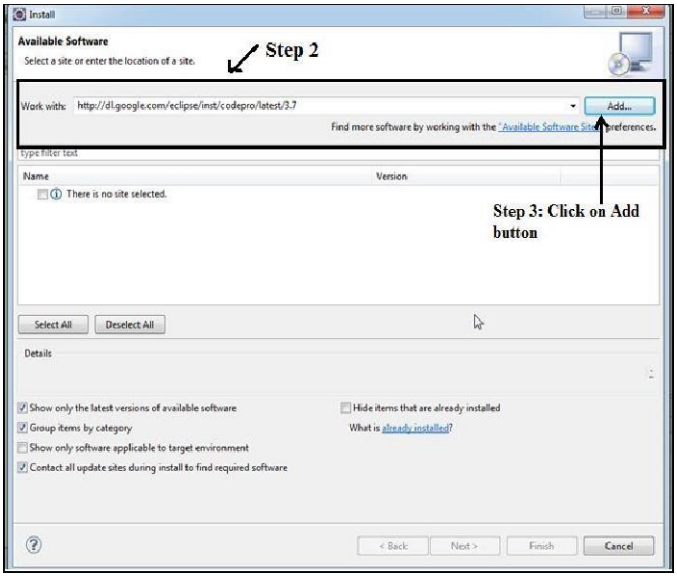
- Code Analysis: Checks for violation of predefined coding standards.
- Metrics : Assess code quality based on factors such as complexity, abstractness etc.
- JUnit Test Generation: Generate JUnit test cases for a project.
- Code Coverage : Measure of how effective the test cases are.
- Junit Test Editor : Helps in quick creation, modification and organization of test cases.
- Dependency Analysis: Analyses dependencies between projects, packages and classes.
- Similar Code Analysis : Identify similar code for refactoring.

Installation steps for codepro analytix in eclipse is as follows:

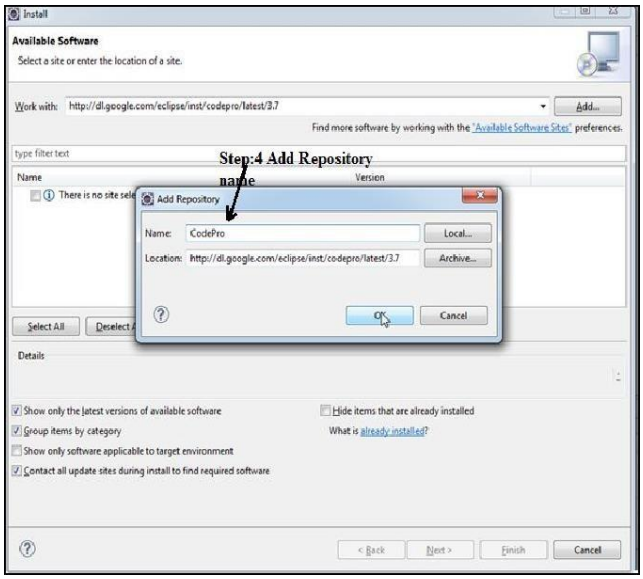
Step 1: Open Eclipse IDE -> Goto Help -> Install new Software



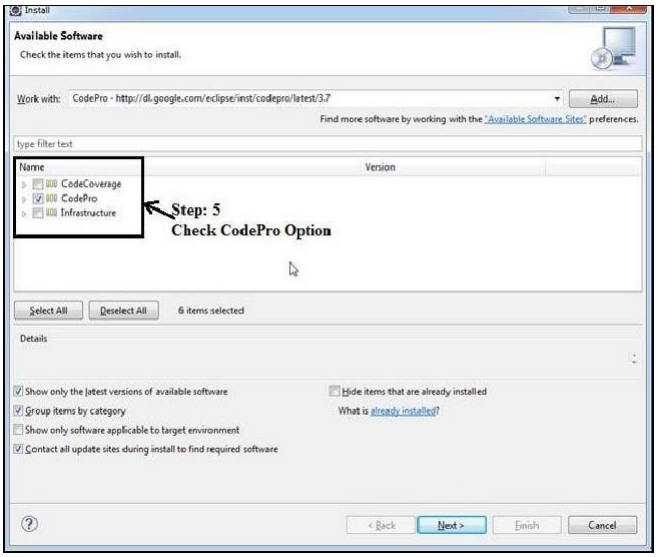
Step 2 and 3: Use the link to add CodePro in Eclipse : http://dl.google.com/eclipse/inst/codepro/latest/3.7 and click on Add button.



Step :4 Add repository name and click on Ok button.



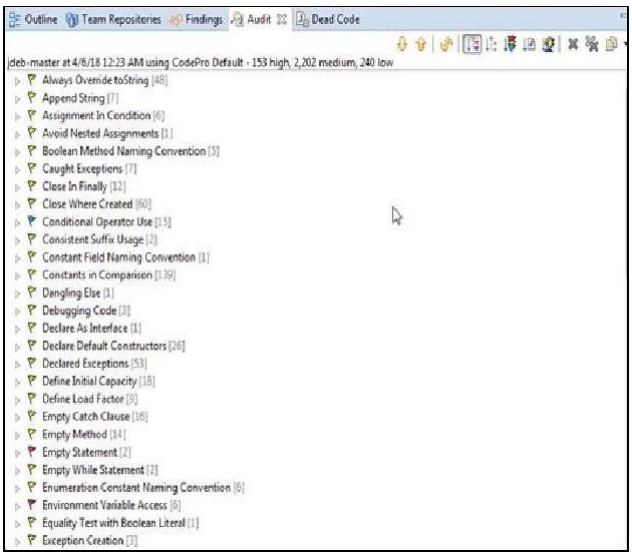
Step:5 Select CodePro option and click on Next



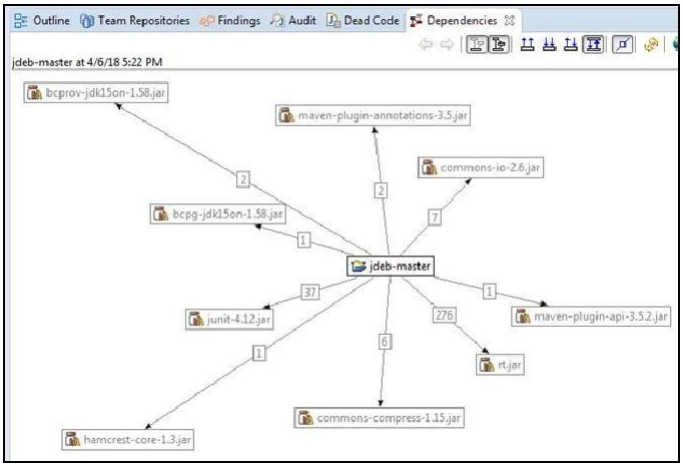
Step 6: Click on Next and accept the license agreement and after installation restart the eclipse. After restart can able to see CodePro Menu when right click on Project.

Analysis of the tool :

CodePro Analytix shows the results of the code audit in the Audit View as shown in the following tree. Root of the tree shows the name of each audit rule with a violation. After expanding, violation audit shows the list of individual violations.



Dependency Analysis gives the relationship between elements at different levels of granularity for instance packages, projects,or types. Following graph shows dependencies. The numbers on links indicates number of nodes involved in the dependency between two projects. Single click on theses boxes shows the dependency of graph at class level[3][4].



Working on this tool was a great experience as it helps to analyse the code using proper audit reports and dependency analysis which checks for the quality of the code.

PART IV

9. CONCLUSION

Working on this project helped me understand the importance of technical debt. I worked on 4 open source projects in which first two were of java, third of python and the last one of PHP. Among them, Library Management system was the most deprecated project as it requires a lot of effort to fix the issues and contain various critical bugs. We used 2 tools teamscale and sonarqube. Teamscale is a free user friendly tool which tells about the code debt, documentation debt etc. However, SonarQube focuses on the effort required to fix the issues and tells us about clone coverage. But none of the tools talked about test coverage. There are various stand alone tools which talks about tests but there is no such tool which has all the features in one. Thus, there is a need for some standard tool else all these tools if used complimentary with each other are helpful. In the first section of this project, I talked about various quality attributes like reliability, maintenance and security of all the projects. Then I identified the TD in particular project and represented it using tables. It was followed by monitoring of TD and ways to repay it. Finally, in the last part I proposed a calculation model using the SonarQube method by adding the some architecture features into the formula and proposed CodePro Analytix as the new tool to manage TD.

10. REFERENCES

[1] CQSE, "Teamscale," [Online]. Available: https://www.cqse.eu/en/products/teamscale/landing/. [Accessed 05 04 2018].

[2] “Sonarqube," SonarSource SA, [Online]. Available: https://www.sonarqube.org/. [Accessed 05 04 2018].

[3] [Online] CodePro AnalytiX, Instantiations.Copyright 2006, Available: http://www.barneyassoc.com/documents/WP- CPA-eclipse.pdf [2008]

[4] C. N. Duy, A. Marchetto, P. Tonella, M. Ceccato, “Coverage testing with CodePro Analytix,” Software Analysis and Tes1ng slides, [Online]. Available: http://selab.fbk.eu/swat/slide/CodePro_Analytix.pdf. [Nov 9, 2010]