Concordia University
Engineering and Computer Science

SOEN 6431 : SOFTWARE COMPREHENSION AND MAINTENANCE

Summer 2022

Deliverable - 1 : Reengineering Oppurtunity

Github Link

Authors

Manimaran Palani
Iphigenia Pappas
Heet Patel
Kevinkumar Patel
Venis Patel

https://www.overleaf.com/project/610304de4e6b8d24f7c781b6

Contents

1. Abstract

For a software system to be able to successfully sustain in the market, constant evolution becomes of utmost importance. Evolution refers to continuous developments and updates in the originally created software system to match the ever-changing market demands at any given point of time. This process of refining the software to keep it abreast of the trends and demands is termed as Software Maintenance. To carry out a successful maintenance by implementing positive changes in the software systems, the analysis of source code and documentation and having a comprehensive understanding of the system holds the most importance.

For this assignment, we choose a candidate R from five different repositories chosen by respective team members. We dive into the detailed analysis of the 'undesirables' in the repositories using TeamScale before tapering down to the candidate R. We further explain the repudiation rationale for the same. The chosen candidate R provides us with the opportunity to reengineer the software program by rectifying the 'undesirables' and thus improving its efficiency and maintainability.

2. Introduction

Software Maintenance plays a massive role in always keeping the software in a 'good shape'. This inturn benefits the maintainers of the software in by-passing the whole cost and hassle of creating a new software system from scratch after being outdated and also aids in keeping it up-to-date according to the market needs. Maintenance also ensures solid quality of code and maintainability. This is very useful in safeguarding the software system from having potential 'undesirables' which can further lead to a decline in quality and efficiency of the source code.

A reengineering process on the source code and documentation of a software can further widen the horizon for itself in terms of its usability, maintainability and efficiency. We carry out an initial analysis using TeamScale of five different repositories by taking note of five unique 'undesirables' for each one of them. We then analyze, based on the quality of the source code and documentation, the reasons for a repository to be an ideal candidate R for reengineering.

Upon successful identification of the candidate R, we buckle down to investigating the 'undesirables' in more detail. We further plan on materializing the solutions to the problems observed in the candidate R by carrying out reengineering operationalization in the upcoming assignment.

3. Project Deliberation

3.1. Communal Work and Responsibility Matrix

In order to identify our candidate system, as the project requirements state: each team member located one repository holding one software system. In order to identify the candidate repository, we scheduled a meeting and discussed reasons to reject each system. All systems had their faults, this can be found below in the Repudiation rationale for each of our systems. We then gave each other the responsibility to write a paragraph explaining why we rejected every system but the one.

DEJA-VU DELIVERABLE-1	Iphigenia Pappas	Manimaran Palani	KevinKumar Patel	Heet Patel	Venis Patel
Project Documentation		x	x	x	x
Project Ideation	x	x			x
Consulting discussions	x	x	x	x	x
TeamScale operation		x			

We split the team accordingly for who's most comfortable with documentation and who is most comfortable with code in order to further prepare for the next two deliverables.

3.2. Candidate System Selection Criteria

Team's Programming capability :
We assessed the experience that all the team individuals have and chosen that Java would be the foremost ideal programming language as all the team members have similar experience using Java as the development language. Any application that doesn't utilize Java given lesser priority.

Complex Architecture :
An application that's less complicated to setup is mostly preferred because it will reduce the hours to spend on setting up the project rather spending it on analysing the project to meet the metrics of candidate system.

Code quality :
The lower the code quality, the more imperative it is to make improvements in it. Typically, its determined by each team member, who rates the project's quality in the scale of identifying 25 unique undesirables. Each group member has assessed the code using the tool TeamScale and agreed collaboratively upon choosing the candidate system.

4. System Description

4.1. Online Banking System

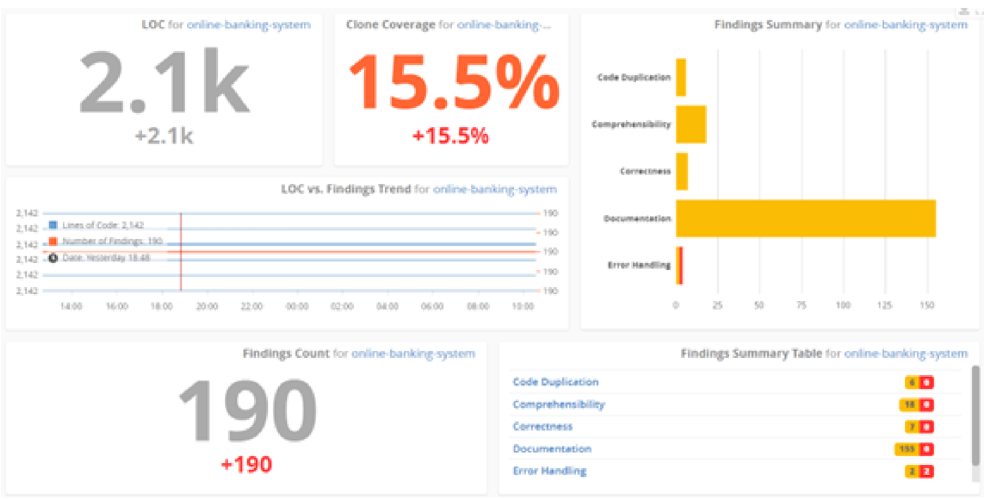
Manimaran Palani - 40167543

Project Description:
The Online Banking System is a banking portal on the web which manages the customer profile and manages their transactions. It is highly scalable and secured with the help of Spring Security. The main feature of this project is validation of login form, viewing customer profile, viewing transaction details of the customer ,Viewing balance of the customer, Approval of the changes in the address by the customer. The core objective of this project is to maintain a personal account in the bank. The system also provides the access to the customer to create an account, deposit/withdraw the cash from the account, also to view reports of all the accounts available.

System Source Code :
https://github.com/ryhan000/Online-Banking-System.git

System Stack :
Spring Boot, Spring Security, Thyme leaf, Spring Data JPA, Spring Data REST, JavaScript, jQuery.

Repudiation Rationale :
The above project is chosen as the candidate R system due to various factors and the details are provide in the section (6). The below picture depicts the software quality report generated by integrating the application with the tool TeamScale which analyse, monitor, and improve the quality of the application.



Findings Summary Table for online-banking-system

Code Duplication	6	0
Comprehensibility	18	0
Correctness	7	0
Documentation	155	0
Error Handling	2	2

Figure 1: Code Quality Report of Online Banking System - TeamScale Dashboard

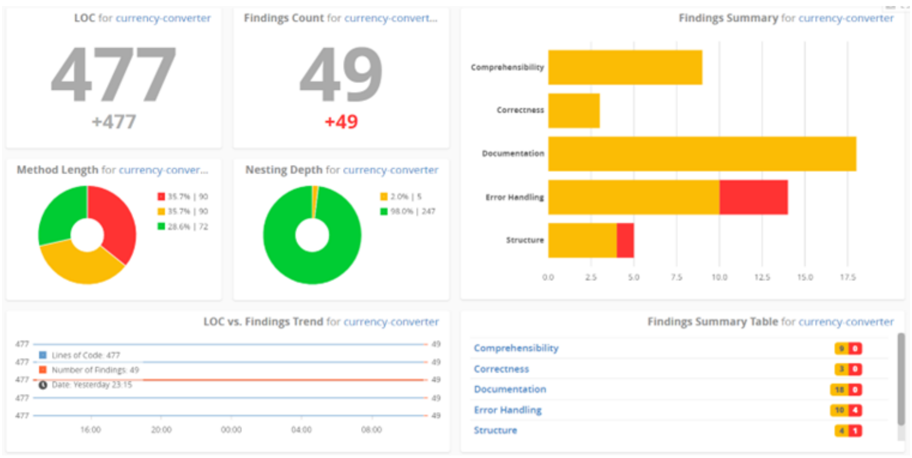
4.2. Currency Converter **Iphigenia Pappas - 40077089**

Project Description:
Currency converter (or currency exchange) is a mini project coded in Java programming language. This simple application provides a web-based interface for exchanging/converting money from one currency to another currency. It is simply a calculator-like app developed using Ajax, Java servlets web features. In this application, there is regular update about currency of every country by which it displays present currency market value and conversion rate. Such application can be used by any user, but it is mainly useful for business, shares, and finance related areas where money transfer and currency exchange take place daily.

System Source Code :
https://github.com/projectworldsofficial/currency-converter-in-java.git

System Stack :
Java Servlets(Java), Ajax

Repudiation Rationale :
This repository had many issues, however the large majority of things that stuck out were features that were not implemented properly. I noticed many defects. Firstly, on top of all the maintenance that needs to be done to the code, the bigger issue was a performance issue, this task could have been done in O(1) rather than O(n) by using a switch statement. The interface defects also bothered me, as much work could have been done to make it more user friendly and allow it to perform multiple conversions at once. There are also multithreading defects as we cannot execute multiple tasks at once. The problem with multithreading, however, is that sometimes there is the condition of deadlock where starvation is created, and if not handled and debugged properly, can end up leading to a failure of the system. Most of the issues had to do with the design of the software rather than the actual code itself, and as this is not the end goal of this assignment (to add enhancements and change the software) therefore we rejected this Software chosen.


LOC for currency-converter
477
+477
Findings Count for currency-convert...
49
+49
Findings Summary for currency-converter
Comprehensibility
Correctness
Documentation
Error Handling
Structure
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
Method Length for currency-conver...
35.7% | 90
35.7% | 90
28.6% | 72
Nesting Depth for currency-converter
2.0% | 5
98.0% | 247
LOC vs. Findings Trend for currency-converter
Lines of Code: 477
Number of Findings: 49
Date: Yesterday 23:15
16:00 20:00 00:00 04:00 08:00
Findings Summary Table for currency-converter

Comprehensibility	9	0
Correctness	3	0
Documentation	18	0
Error Handling	10	4
Structure	4	1

Figure 2: Code Quality Report of Currency Converter - TeamScale Dashboard

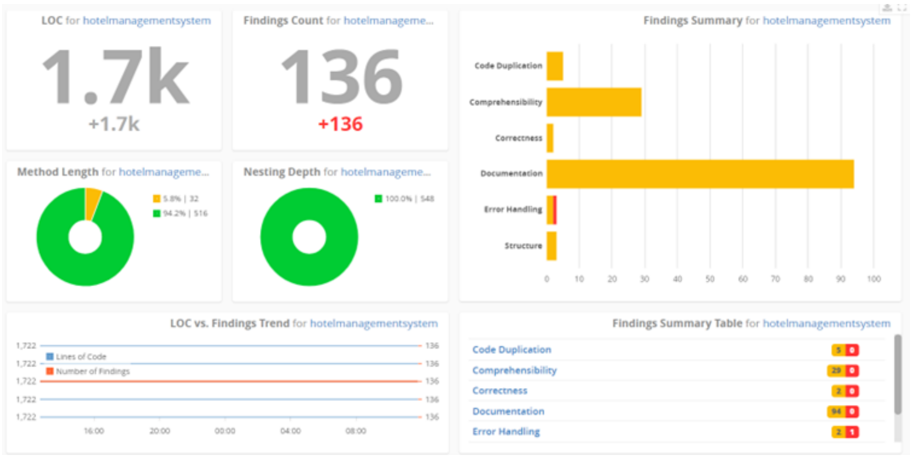
4.3. Hotel Management System **Heet Patel - 40179213**

Project Description:
Hotel Management System is developed in java platform. This is a Web Base Management System developed for Managing Hotel Day. This application covers all activities in the hotel including Registrations, Employee Management, Bill Printing, Account Control, Result analysing, Food and Beverage Details, and Issuing Rooms. The system role is divided into two types: manager (administrator) and employee (ordinary user), of which manager (administrator) has the right to view all bookings, delete rooms, view employees, add employees and other functional permissions Employees (ordinary users) have the right to view available rooms, customer reservations, modify reservations, delete reservations, register new customers and other functional permissions (tasks are self-drafted).

System Source Code :
https://github.com/jsanhuo/HotelManagementSystem.git

System Stack :
HTML5, CSS3, JavaScript, Bootstrap3, JDK1.8 + Spring Boot, Mybatis (Persistence Layer Framework), Druid (Database Connection Pool).

Repudiation Rationale :
The Hotel Management System followed a similar structure as our chosen candidate, it was easily to categorize the system in two groups which includes manager (administrator) and employee (ordinary user). This system was also split into many different functionalities that have been listen in the paragraph above entitled Project Description. There were three particular reasons we rejected this system. Firstly it had far too many components and was difficult to follow, what made it even worse to follow is that the comments and documentation of the repository seem to have been done in a language that none of our team mates are familiar with. This makes it that much more difficult to follow the code. We also took into consideration that the number of undesirables was less than the chosen candidate. This was part of our reason for rejection, but certainly not the main one.


LOC for hotelmanagementsystem
1.7k
+1.7k
Findings Count for hotelmanageme...
136
+136
Method Length for hotelmanageme...
5.8% | 32
94.2% | 516
Nesting Depth for hotelmanageme...
100.0% | 548
Findings Summary for hotelmanagementsystem
Code Duplication
Comprehensibility
Correctness
Documentation
Error Handling
Structure
LOC vs. Findings Trend for hotelmanagementsystem
Lines of Code
Number of Findings
16:00
20:00
00:00
04:00
08:00

Findings Summary Table for hotelmanagementsystem

Code Duplication	5	0
Comprehensibility	29	0
Correctness	2	0
Documentation	94	0
Error Handling	2	1

Figure 3: Code Quality Report of Hotel Management System - TeamScale Dashboard

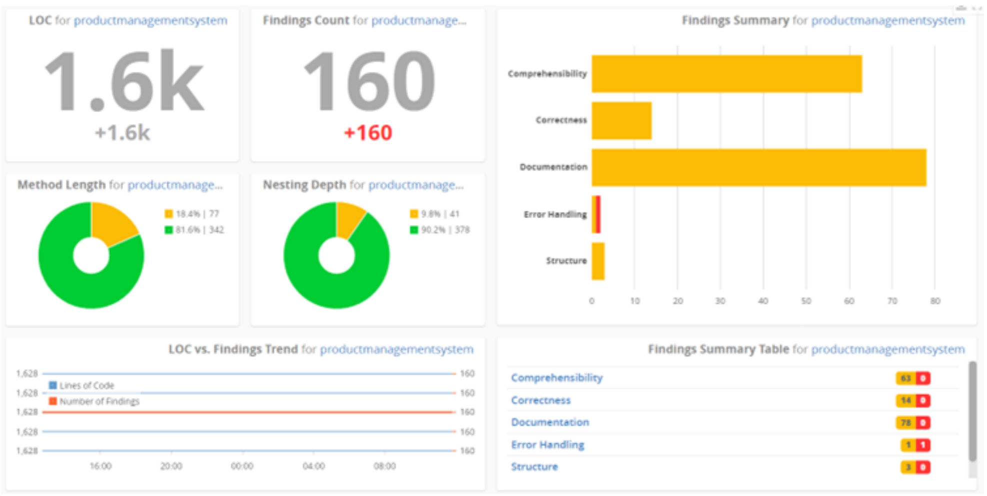
4.4. Product Management System KevinKumar Patel - 40194915

Project Description:
It is a marketplace where customer can place order and Admin can manage inventory of products, view order and generate the report.

System Source Code :
https://github.com/anantjain6/ProductManagementSystem.git

System Stack :
Java 8, Spring MVC, Spring Data, Spring Boot, Hibernate JPA, H2 In-memory Database, Maven for Dependency Management.

Repudiation Rationale :
The product management system was far too complex for the level of this course. Simply understanding the multiple technologies used that can be found below require more time to learn:
* Maven for Dependency Management
* Spring MVC for Web application development
* Spring Data JPA for Creating Custom Repository
* Spring Boot for Autoconfiguration
* Spring Security for Authentication Authorisation
* Hibernate Validator for form data validation
* H2 In-memory Database for Storing data
* Java Mail API to send HTML E-Mail over SMTP
* JSTL
We want to concentrate our efforts in the maintenance of the code within the scope of this course, rather than spend countless hours trying to learn multiple new technologies. This would have complicated the end goal of this assignment and most importantly this course.


LOC for productmanagementsystem
1.6k
+1.6k
Findings Count for productmanage...
160
+160
Findings Summary for productmanagementsystem
Comprehensibility
Correctness
Documentation
Error Handling
Structure
0
10
20
30
40
50
60
70
80
Method Length for productmanage...
18.4% | 77
81.6% | 342
Nesting Depth for productmanage...
9.8% | 41
90.2% | 378
LOC vs. Findings Trend for productmanagementsystem
1,628
Lines of Code
Number of Findings
160
16:00
20:00
00:00
04:00
08:00
Findings Summary Table for productmanagementsystem

Comprehensibility	63	0
Correctness	14	0
Documentation	78	0
Error Handling	1	1
Structure	3	0

Figure 4: Code Quality Report of Product Management System - TeamScale Dashboard

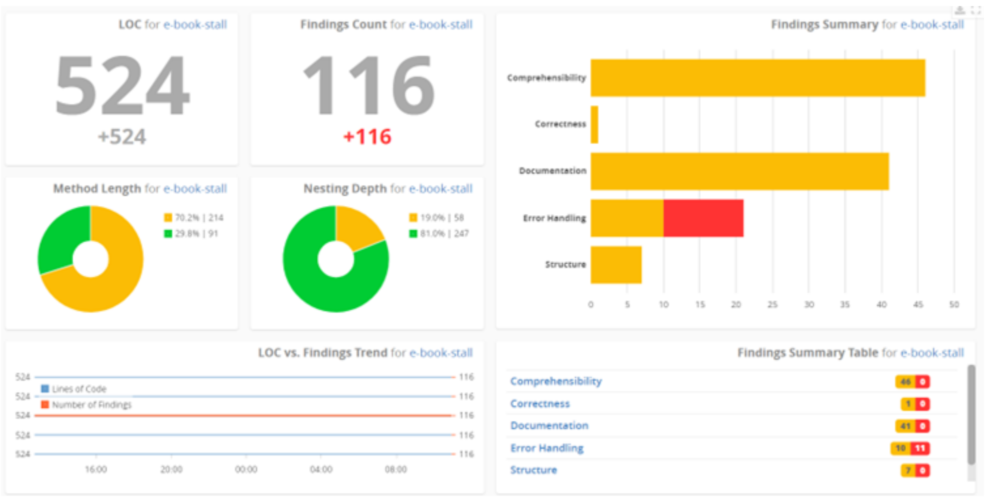
4.5. E-book stall **Venis Patel - 40170617**

Project Description:
The application is built to create an online platform to sell books, The application comprises of functionalities like maintaining books selling history, adding, and managing books, updating the availability of books, filtering the books based on author or the publisher and the simulation of online payment method to buy the book. The application is built in Generic Servlets in Java, JDBC and MySQL.

System Source Code :
https://github.com/shashirajraja/onlinebookstore.git

System Stack :
Html, CSS, JavaScript, Java [JDK 8+], JDBC, Servlet, MySQL, Apache Maven.

Repudiation Rationale :
project was well made, this exact project is often given in courses to test object oriented programming skills. Jenna had to make a project almost exactly like this one. However, that being said, the system was far too basic. It concentrates more on OOP than actually building a software. This surely also explains why the number of undesirables was far less than other projects, because it's not nearly as complex as we would need it to be in order to apply multiple fixes. We also couldn't find many undesirables, the main issues include changing the code for it to be better optimized, which is not the purpose of this assignment. Lastly, we also rejected this software because 50 percent of it was front end. It is far too much a surface level project for the scope of this course.



Comprehensibility	46	0
Correctness	1	0
Documentation	41	0
Error Handling	10	11
Structure	7	0

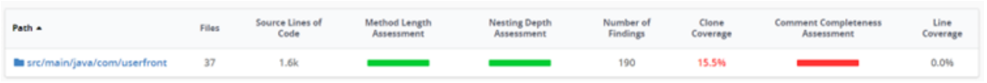
Figure 5: Code Quality Report of E-Book Stall - TeamScale Dashboard

5. Candidate System Description

The Online Banking System is a banking portal on the web which manages the customer profile and manages their transactions. It is highly scalable and secured with the help of Spring Security. The main feature of this project is validation of login form, viewing customer profile, viewing transaction details of the customer ,Viewing balance of the customer, Approval of the changes in the address by the customer. The core objective of this project is to maintain a personal account in the bank. The system also provides the access to the customer to create an account, deposit/withdraw the cash from the account, also to view reports of all the accounts available.

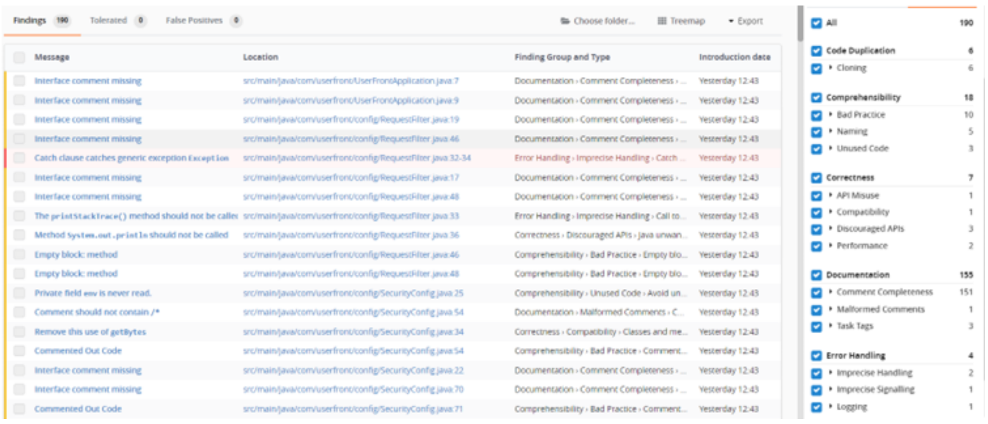
6. Candidate System Rationale

The reason we chose the online banking system as our candidate system is not only because it was the better one of the choices from the software provided by the team (rejected reasons can be found below for all other systems) but for several other important reasons. To name a few, this system met all the requirements and more. This was complex enough and for us to have a lot to work on, without being too much where we would get lost in the code. There is not too much spaghetti code, although it may be optimized to less lines of code. This project was between 1000 to 2000 lines of code which was the target desire for our system. We also were easily able to locate the 25 undesirables to fix for this system, each team member was able to identify 5 distinct undesirables that we can easily distribute to the team to provide a fix. Lastly, this system was written in large majority in our programming language of choice which is Java. The architecture of this system also contains multiple distinct aspects that we can easily categorize in two main groups: managing customer profile and managing transactions. This system also contains a certain level of security as it maintains credential information by validating a log in form. This, then allows us to do other actions such as viewing customer profile, viewing transaction details of the customer, viewing balance of the customer, approval of the changes in the address by the customer. In summary, this system met all the requirements and more and was well structured enough for us to clearly identify our undesirables.



Path ▲	Files	Source Lines of Code	Method Length Assessment	Nesting Depth Assessment	Number of Findings	Clone Coverage	Comment Completeness Assessment	Line Coverage
src/main/java/com/userfront	37	1.6k	🟩	🟩	190	15.5%	🟥	0.0%

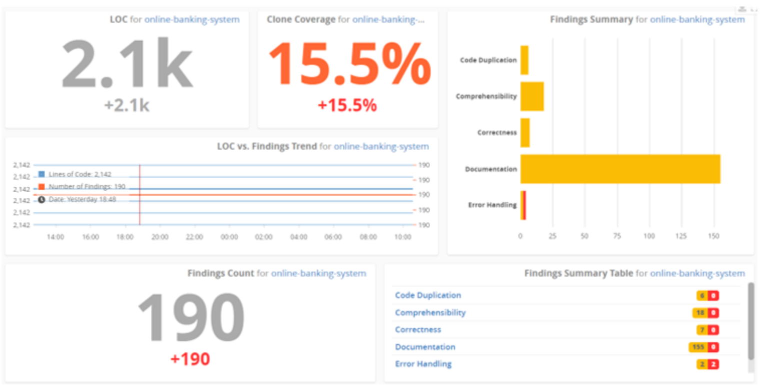
Figure 6: overall assessment generated for Candidate R online banking system from TeamScale



Message	Location	Finding Group and Type	Introduction date
Interface comment missing	src/main/java/com/userfront/UserFrontApplication.java:7	Documentation › Comment Completeness › ...	Yesterday 12:43
Interface comment missing	src/main/java/com/userfront/UserFrontApplication.java:9	Documentation › Comment Completeness › ...	Yesterday 12:43
Interface comment missing	src/main/java/com/userfront/config/RequestFilter.java:19	Documentation › Comment Completeness › ...	Yesterday 12:43
Interface comment missing	src/main/java/com/userfront/config/RequestFilter.java:46	Documentation › Comment Completeness › ...	Yesterday 12:43
Catch clause catches generic exception Exception	src/main/java/com/userfront/config/RequestFilter.java:32-34	Error Handling › Imprecise Handling › Catch ...	Yesterday 12:43
Interface comment missing	src/main/java/com/userfront/config/RequestFilter.java:17	Documentation › Comment Completeness › ...	Yesterday 12:43
Interface comment missing	src/main/java/com/userfront/config/RequestFilter.java:48	Documentation › Comment Completeness › ...	Yesterday 12:43
The printStackTrace() method should not be callec	src/main/java/com/userfront/config/RequestFilter.java:33	Error Handling › Imprecise Handling › Call to...	Yesterday 12:43
Method System.out.println should not be called	src/main/java/com/userfront/config/RequestFilter.java:36	Correctness › Discouraged APIs › Java unwan...	Yesterday 12:43
Empty block: method	src/main/java/com/userfront/config/RequestFilter.java:46	Comprehensibility › Bad Practice › Empty blo...	Yesterday 12:43
Empty block: method	src/main/java/com/userfront/config/RequestFilter.java:48	Comprehensibility › Bad Practice › Empty blo...	Yesterday 12:43
Private field env is never read.	src/main/java/com/userfront/config/SecurityConfig.java:25	Comprehensibility › Unused Code › Avoid un...	Yesterday 12:43
Comment should not contain /*	src/main/java/com/userfront/config/SecurityConfig.java:54	Documentation › Malformed Comments › C...	Yesterday 12:43
Remove this use of getBytes	src/main/java/com/userfront/config/SecurityConfig.java:34	Correctness › Compatibility › Classes and me...	Yesterday 12:43
Commented Out Code	src/main/java/com/userfront/config/SecurityConfig.java:54	Comprehensibility › Bad Practice › Comment...	Yesterday 12:43
Interface comment missing	src/main/java/com/userfront/config/SecurityConfig.java:22	Documentation › Comment Completeness › ...	Yesterday 12:43
Interface comment missing	src/main/java/com/userfront/config/SecurityConfig.java:70	Documentation › Comment Completeness › ...	Yesterday 12:43
Commented Out Code	src/main/java/com/userfront/config/SecurityConfig.java:71	Comprehensibility › Bad Practice › Comment...	Yesterday 12:43

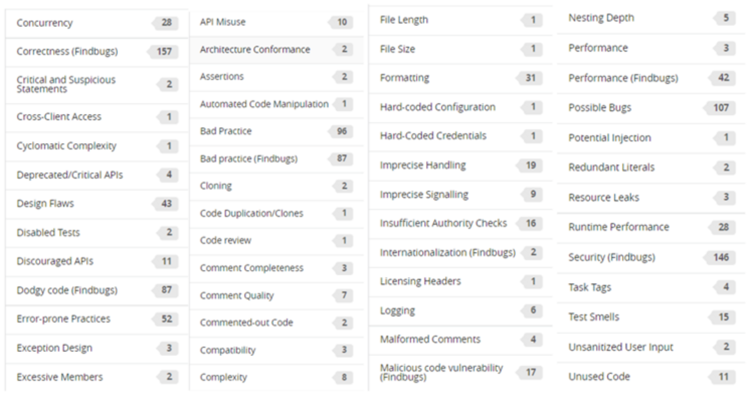
✓ **All**	**190**
✓ **Code Duplication**	**6**
✓ Cloning	6
✓ **Comprehensibility**	**18**
✓ Bad Practice	10
✓ Naming	5
✓ Unused Code	3
✓ **Correctness**	**7**
✓ API Misuse	1
✓ Compatibility	1
✓ Discouraged APIs	3
✓ Performance	2
✓ **Documentation**	**155**
✓ Comment Completeness	151
✓ Malformed Comments	1
✓ Task Tags	3
✓ **Error Handling**	**4**
✓ Imprecise Handling	2
✓ Imprecise Signalling	1
✓ Logging	1

Figure 7: Category of Findings generated for Candidate R online banking system from TeamScale


LOC for online-banking-system
2.1k
+2.1k
Clone Coverage for online-banking-...
15.5%
+15.5%
Findings Summary for online-banking-system
Code Duplication
Comprehensibility
Correctness
Documentation
Error Handling
0
25
50
75
100
125
150
LOC vs. Findings Trend for online-banking-system
Lines of Code: 2,142
Number of Findings: 190
Date: Yesterday 18:48
2,142
190
14:00
16:00
18:00
20:00
22:00
00:00
02:00
04:00
06:00
08:00
10:00
Findings Count for online-banking-system
190
+190
Findings Summary Table for online-banking-system

Code Duplication	6	0
Comprehensibility	18	0
Correctness	7	0
Documentation	155	0
Error Handling	2	2

Figure 8: Code Quality Report of Online Banking System - TeamScale Dashboard



Concurrency	28	API Misuse	10	File Length	1	Nesting Depth	5
Correctness (Findbugs)	157	Architecture Conformance	2	File Size	1	Performance	3
Critical and Suspicious Statements	2	Assertions	2	Formatting	31	Performance (Findbugs)	42
Cross-Client Access	1	Automated Code Manipulation	1	Hard-coded Configuration	1	Possible Bugs	107
Cyclomatic Complexity	1	Bad Practice	96	Hard-Coded Credentials	1	Potential Injection	1
Deprecated/Critical APIs	4	Bad practice (Findbugs)	87	Imprecise Handling	19	Redundant Literals	2
Design Flaws	43	Cloning	2	Imprecise Signalling	9	Resource Leaks	3
Disabled Tests	2	Code Duplication/Clones	1	Insufficient Authority Checks	16	Runtime Performance	28
Discouraged APIs	11	Code review	1	Internationalization (Findbugs)	2	Security (Findbugs)	146
Dodgy code (Findbugs)	87	Comment Completeness	3	Licensing Headers	1	Task Tags	4
Error-prone Practices	52	Comment Quality	7	Logging	6	Test Smells	15
Exception Design	3	Commented-out Code	2	Malformed Comments	4	Unsanitized User Input	2
Excessive Members	2	Compatibility	3	Malicious code vulnerability (Findbugs)	17	Unused Code	11

Figure 9: Quality Metrics used to assess Candidate R online banking system from TeamScale

7. References

1. D. Korolev, ”How do you define code quality?,” 09 02 2014. [Online]..
2. ISO/IEC 25010:2011,” 03 2011. [Online]. .
3. ISO/IEC 9126-1:2001,” 06 2001. [Online].
XML Tutorial,” 2017. [Online]..
4. What is the optimal size of a software development team,” 16 05 2009. [Online]..
5. R. . Norvig, The intelligent agent paradigm, 2003, pp. 27, 32–58, 968–972 .
6. ”Teamscale Documentation” [Online]..