UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF COMPUTER SCIENCE

BSC COMPUTER SCIENCE: 6COM1031

Module: Software Engineering Practise

Assignment 1 - Code Review and Critique

Helitha Rupasinghe

Level 6

Academic year 2020-21

Contents

1.0 The code Review Process

The goal of this report is to act as a guide to identify the types of weaknesses that exist within the **Tinee Client Prototype**. The main task involves both manual and automated testing of source code to identify specific issues with regards to existing functionality and future functionality. A formal code review will identify every issue in the code and the risk associated with the code so that mitigation strategies can be formulated. Overall, the critique of the application in question requires manual investigation of java code that includes looking at source files to review the entire application.

1.2 Disclaimers

The information I present in this document is confidential and is provided by University of Hertfordshire. Code Reviews conducted are part of the analysis that target the weaknesses of the application under review which is reflected in this report. Additionally, new weaknesses can be discovered as new tests are run and so this report should be considered a guide and not a 100% representation of the weaknesses threatening your systems.

Address questions regarding legitimate use of this document to the postcode below:

University of Hertfordshire
Hatfield
Hertfordshire
AL10 9AB
UK

1.3 Findings Summary of Tinee Client Prototype

Tinee Client prototype is an application that is required to be developed as an interactive command line utility. As stated in the specification, Tinee is a simple text-based ticket system for use by the organisation for users to create tickets and log various problems. The goal of the application is to be integrated and embedded into other development projects or other applications.

1.4 Scope

The current scope of this review is to identify the type of weaknesses that exist within the Tinee Prototype. This will be conducted using testing procedures that involve the use of static code analysis tools under a limited time-frame whilst providing solutions to constraints found within the prototype.

1.5 Purpose

This report states the requirements for starting the Tinee client + server and critiques the functionality of the application . The current prototype consists of a basic framework that reads user commands from the standard input stream '**System.in'** in java, output on the standard output and error streams '**System.out'** and '**System.err'**. Currently, the server acts as a database of tickets identified by **tags: tickets** act as a message line (list of **tines**), where each **tine** contains the username of its author **and** a **single line of text as the message body.** This means that a client connected to a server should be able to '**read existing tines'** for a tag and be able to '**add new tines'**.

The goal of this section is to clarify the basic client-server architecture for the Tinee Server. The current prototype accepts only a few commands which are to terminate the app: **read** the current tines on the server, **Start** drafting new tines on the server, **Add** a line of text to the current draft and **Push** the drafter tines to the server. In essence, figure 1 is highlighting that basic relationship

between the client and the server which starts with the client submitting a request through the command line and the server responding to the submitted request.

Figure 1: Sequence diagram to highlight the interaction between the client and the server. (Generated on word).2021

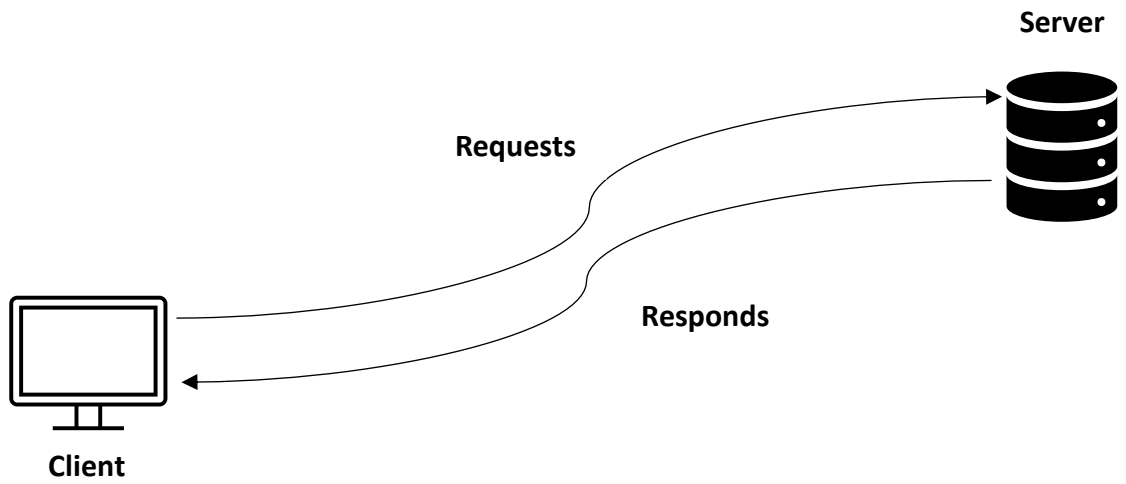


Table 1Relationship between user input and behaviour . 2021

Command requests from Client	**Client State (Current -> Successor)**	**Behaviour**
Exit	[CurrentState] → [Terminated]	Terminate the App.
Read [mytag]	Main → Main	Read the current tines for [tag] on the server
Manage[text]	Main → Drafting	Start drafting new tines to add to [tag]
line [tag]	Drafting → Drafting	Add a line of text to the current draft
Undo	Drafting → Drafting	Undo appropriate commands in drafting mode
Push	Drafting → Main	Push the drafted tines to the server
Show	Main → Main	Get the set of ticket tags and creators from the server.
Discard	Drafting → Main	Discard all drafted tines without pushing them to the server

The relationship between client and the server can be broken down to sequence the user inputs, the prototypes 'states' and their behaviour can be highlighted in table 1 with green being the functionality currently accepted by the prototype and red emphasizing requirements that are yet to be implemented. The initial state that is visible on the command line when running the prototype is 'Main' where the protocol for the user commands can be considered as a state machine that determines when commands are valid in this case is between the states '**Main'** and '**Drafting**'. Additionally, the commands highlighted in table 1 support all existing and future functionality. This

means that the system will be able to smoothly handle user input errors for e.g. mistype commands or invalid arguments. Forthcoming functionality, to be added will look at implementing a **GUI** for the Tinee client alongside the CL utility: support for **internationalisation** where UI messages can alternative between languages, the UI will be able to report usage errors and continue running without crashing the session given that the user provides correct information.

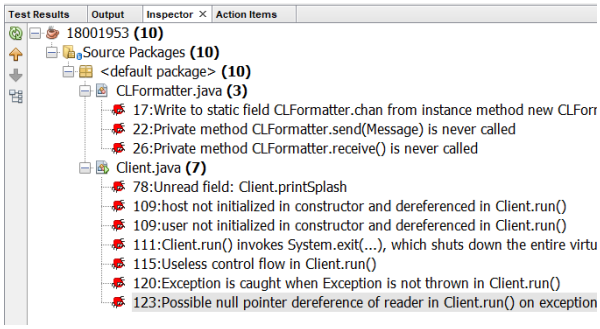
2.0 Analysis of Software Design

The purpose of the section is to identify the correctness of the prototype made by the developers using concepts such as coupling and cohesion. Cohesion showcases the modules functional strength which refers to the relationship of the module whereas Coupling is the measure of linkage between the modules. The end design should use low coupling and high cohesion to be a good software application[5].

The behaviour of the application is deemed correct when it behaves as intended by the developers in this case the prototype must work as intended by the specifications. The NetBeans IDE provide tools that can help to identify defects focus on testing the application, using Junit: TestNG as well as code analysers such as EasyPMD and Find Bugs[1][2][3]. Overall, these static analysis tools are integrated to the IDE that use code patterns to analyse syntax errors amongst other problems.

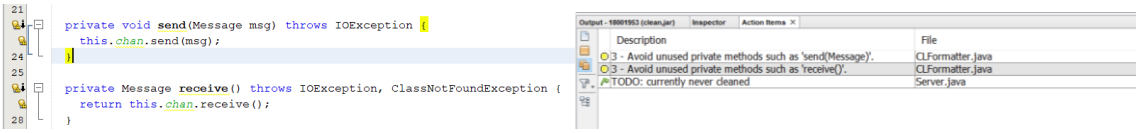
This report will focus on two static analysis tools which are Find Bugs and EasyPMD. Find Bugs generates a bug report and displays it within the inspector window of the Netbeans IDE that categorises all problems from bugs to suspicious code[1][2]. Similarly, EasyPMD performs code analysis using PMD to show the results in the editor and within the action items window[3].

The initial checks run of Find bugs involved manual testing of the Java source code for the prototype application. The FindBugs tool identified 10 issues referenced to two classes that included the 'Client.java' and 'CLFormatter.java' as shown in figure 2. The majority of the issues, seven in total highlighted within the 'Client.java' class focused on uninitialised fields(line 109): unread fields (line 78), useless control flow(line 115) in client.run() and found issues with Client.run() invoking System.Exit() which shutdown the entire virtual machine(line111). Furthermore, bugs were found at line 120 within the run method that contained a try-catch block that catches exception objects but the exception caught is not being thrown in Client.run(). An alternative approach to the bug found at line 20 would be to explictily catch RunTime Exceptions , rethrow it and then catch all non-runtime exceptions showcased in figure 21. Additional, checks found '3' issues in total with the 'CLFormatter.java' class that follow bad coding practise for private methods at lines 22, 26 and instance methods writen to static fields at line 17 (bad practise). Overall, findings discovered should be remediated immediately to secure the application and prevent vulnerabilities.



(Figure 2 : Project view identified issues with two classes (Generated on Netbeans) . Date Accessed: 20/2/2021)

The specific findings run on EasyPMD found confusing behaviour with unused private methods , and fields. The checks run using EasyPMD were partially useful as they identified confusing behaviour 'buggy issues' with unused private methods for 'receive()' (line29) and 'send(message)' (line25) within the 'CLFormatter.java' class highlighted by figure 3 which goes against good coding practises. However, upon customising additional rulesets (figure 22) highlighted additional 'buggy issues' in figures 23 and 24 for raw/generic exception types: violation of Law of Demeter, string comparisons and empty catch blocks that should be remediated immediately by the developers to prevent vulnerabilities.



(Figure 3: Project view identified issues with one class (Generated on NetBeans) . Date Accessed: 22/2/2021)

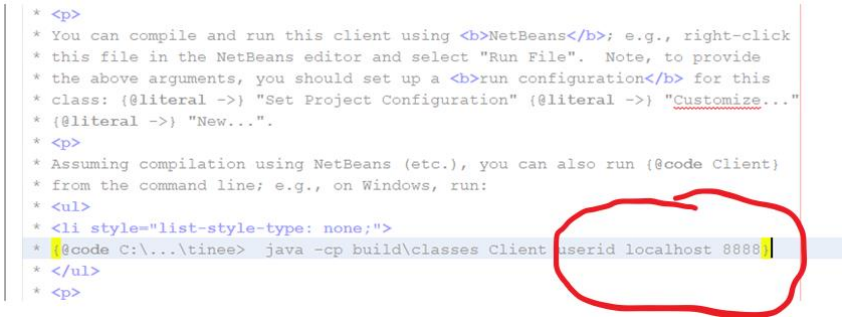
3.0 Functionality of Tinee Client Application

The prototype is written in the NetBeans IDE using Java that carries essential tools for creating: editing , refactoring , building , debugging , profiling , testing and code analysis.

This section of the report will focus on running through the prototype from the perspective of the user by using the NetBeans IDE to showcase the command line arguments, client state and the current behaviour of the prototype using commands that currently are limited to accepting three arguments such as read[mytag]: manage[mytag] and exit etc (figure 8) that are visible to the user upon compiling the application.

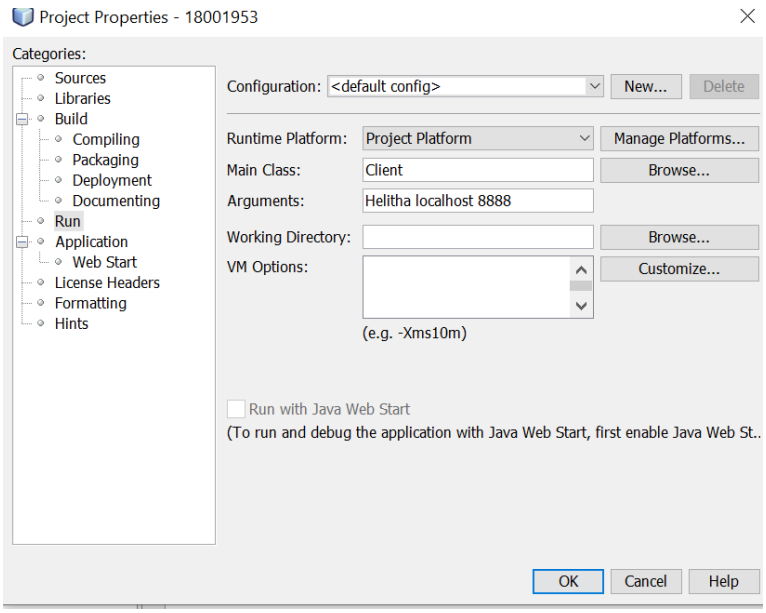
3.1 How to build the Tinee Client and Server?

In order, to start the application the required command line arguments that are needed at runtime are highlighted in figure 4. Figure 4 shows that a 'userID': a 'localhost' and 'portnumber 888' are required by the client.



(Figure 4: Runtime Arguments required by the client (Generated on Netbeans) . Date Accessed: 22/2/2021)

The NetBeans IDE contains a project properties tab which contains a field for arguments required by the Client class at runtime shown by (figure 5). The arguments used by the client are as follows for userid that requires a name which in this case is 'Helitha': using localhost (current hostname) and the required port number from (figure 4) that was '8888'.



(Figure 5: Runtime Arguments required by the client (Generated on NetBeans) . Date Accessed: 22/2/2021)

As specified by the specification, the Tinee client is to be developed as a command line utility. This means that in order to start the server, a set of arguments are required which are highlighted by figure 6. Figure 6 highlights the code required to start the server as follows '**java -cp build\classes sep.tinee.server.Server 8888**' in command prompt that acts as a command line interpreter application that is available in nearly all windows operating systems.

```
* <p>
* You can compile and run this server using <b>NetBeans</b>; e.g., right-click
* this file in the NetBeans editor and select "Run File".  Note, to provide
* the above argument, you should set up a <b>run configuration</b> for this
* class: {@literal ->} "Set Project Configuration" {@literal ->} "Customize..."
* {@literal ->} "New..." -- the "Main Class" is
* {@code sep.tinee.server.Server}, and write the port value (e.g.,
* {@code 8888}) in "Arguments".
* <p>
* Assuming compilation using NetBeans (etc.), you can also run {@code Server}
* from the command line, e.g., on Windows, run:
* <ul>
* <li style="list-style-type: none;">
* {@code C:\...\tinee> java -cp build\classes sep.tinee.server.Server 8888}
* </ul>
```

(Figure 6: Arguments required to start the server (Generated on NetBeans) . Date Accessed: 22/2/2021)

The effects of using command prompt secured a successful connection shown by figure 7 that correlates the results inputted in the client on CMD (Command Line Interpreter).

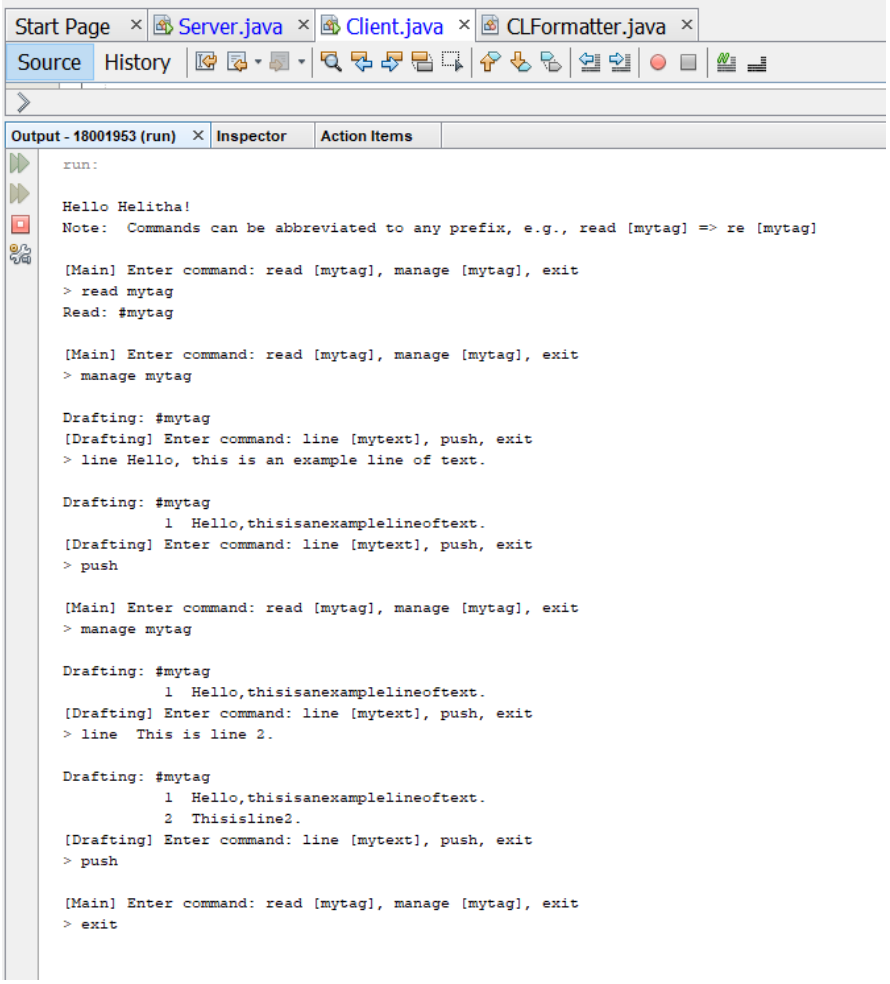
```
:\Users\Helitha>cd C:\Users\Helitha\Documents\NetBeansProjects\18001953

:\Users\Helitha\Documents\NetBeansProjects\18001953>java -cp build\classes sep.tinee.server.Server 8888
Client /127.0.0.1:51078) Accepted connection.
Client /127.0.0.1:51078) Received: READ 0 mytag
Client /127.0.0.1:51078) Sent: READ-REPLY
Client /127.0.0.1:51078) Received: BYE
Client /127.0.0.1:51078) Closing connection.
Client /127.0.0.1:51078) Session ended.
```

(Figure 7: Results of starting the server (Generated on NetBeans) . Date Accessed: 22/2/2021)

3.2 Analysis of commands currently supported by the current Tinee Application

As shown in table 1, each command identified by the specification has a set of behavioural rules that need to be requested by the client that the server responds to. The goal of this section is to show the current working modules of the Tinee prototype.

The commands that are tested in this section are as follows read [mytag] : manage [mytag], line[mytext] , push and exit. Figure 8 highlights the flow of execution beginning with manage mytag that allows the client to enter drafting mode. Once, entering drafting mode the user can access more commands such as line [mytext]: push and exit. The line[mytext] command allow the user to add a line of text to the current draft. For e.g. 'line Hello, this is an example line of text', 'line This is line 2' which then requires a 'Push' command to add the drafted tines to the server that is followed by the 'exit' command to terminate the app. The actions of the inputted tines carried out on NetBeans are visible on CMD (command line interpreter) shown by figure 9.



(Figure 8: Project output for the client class (Generated on NetBeans) . Date Accessed: 22/2/2021)

```
:\Users\Helitha\Documents\NetBeansProjects\18001953>java -cp build\classes sep.tinee.server.Server 8888
Client /127.0.0.1:51078) Accepted connection.
Client /127.0.0.1:51078) Received: READ 0 mytag
Client /127.0.0.1:51078) Sent: READ-REPLY
Client /127.0.0.1:51078) Received: BYE
Client /127.0.0.1:51078) Closing connection.
Client /127.0.0.1:51078) Session ended.
Client /127.0.0.1:51179) Accepted connection.
Client /127.0.0.1:51179) Received: READ 0 mytag
Client /127.0.0.1:51179) Sent: READ-REPLY
Client /127.0.0.1:51179) Received: PUSH #mytag @Helitha Hello,thisisanexamplelineoftext.
Client /127.0.0.1:51179) Received: PUSH #mytag @Helitha Hello,thisisanexamplelineoftext. @Helitha Thisisline2.
Client /127.0.0.1:51179) Received: BYE
Client /127.0.0.1:51179) Closing connection.
Client /127.0.0.1:51179) Session ended.
```

(Figure 9: Project output on CMD (Generated on NetBeans) . Date Accessed: 22/2/2021)

The robustness of the current prototype is in question when the user mistypes commands or use invalid arguments shown by figure 10 which is reported within NetBeans output window when compiling the client class. In this scenario, NetBeans correctly identifies the error with processing capitalised user input prior to entering drafting mode that is displayed in the output window without crashing the session using the standard output stream to show the result of the invalid argument that is not accepted by the application. Overall, as specified by the specification the current prototype supports user input errors as part of its existing functionality that is demonstrated successfully in figure 10 without crashing the session.

```
Output - 18001953 (run)   Inspector   Action Items

run:

Hello Helitha!
Note:  Commands can be abbreviated to any prefix, e.g., read [mytag] => re [mytag]

[Main] Enter command: read [mytag], manage [mytag], exit
> MANAGE MYTAG
Could not parse command/args.
```

(Figure 10: Project output on Netbeans (Generated on NetBeans) . Date Accessed: 22/2/2021)

3.3 Constraints within the Prototype Application

With regards to the specification, a constraint falls upon the organisation with implementing **support for internationalisation** as the code reflects hard coded commands that are written in English highlighted in figure 11. Figure 11 highlights how the developers have hard coded the commands in English for e.g. “Main”: “Drafting” , “Line” and “Push” which would requires the developers to extend the current class to accommodate for international users. A possible solution would be to create a resource bundle for each language that lets the Java runtime environment determine the appropriate language for incoming client requests.

```
else if (state.equals("Main")) {
  if ("manage".startsWith(cmd)) {
    // Switch to "Drafting" state and start a new "draft"
    state = "Drafting";
    draftTag = rawArgs[0];
  } else if ("read".startsWith(cmd)) {
    // Read tines on server
    helper.chan.send(new ReadRequest(rawArgs[0]));
    ReadReply rep = (ReadReply) helper.chan.receive();
    System.out.print(
        helper.formatRead(rawArgs[0], rep.users, rep.lines));
  } else {
    System.out.println("Could not parse command/args.");
  }
} // "Drafting" state commands
else if (state.equals("Drafting")) {
  if ("line".startsWith(cmd)) {
    // Add a tine message line
    String line = Arrays.stream(rawArgs).
        collect(Collectors.joining());
    draftLines.add(line);
  } else if ("push".startsWith(cmd)) {
    // Send drafted tines to the server, and go back to "Main
```

(Figure 11: Client.java class view (Generated on NetBeans) . Date Accessed: 22/2/2021)

A design constraint that is brought to attention within the client.java class is that the current prototype uses no access modifiers which doesn't support data encapsulation since the data is not hidden. A possible solution is to change the access modifiers for the variables defined in the client class to private to achieve encapsulation highlighted by figure 12.

```
public class Client {

 String user;
 String host;
 int port;

boolean printSplash = true;

    /**
     *

    public Client() {
  }*/
```

```
public class Client {

 private String user;
 private String host;
 private int port;

 private boolean printSplash = true;
```

(Figure 12: Client.java class view (Generated on NetBeans) . Date Accessed: 22/2/2021)

Additionally, the specification outlines an alternative user interface that focuses on a graphical UI (GUI) for the Tinee client alongside the CL utility. This means that the current design of the prototype will need to be extended to facilitate this option for the future . A constraint with the current command line application is that the current prototype is heavily coupled to its user interface from the use of standard output streams such as 'System.out.print.ln' being used frequently highlighted in figure 13. A possible solution is to modularise the current client.java class logic using MVC(Model-View-Controller) to facilitate a GUI alongside the Command Line utility. Successful implementation of this design pattern will allow for decoupling between the **view** that represents the visualisation of the data and the **model** that represents the logic alongside the **controller** that controls the data flow into the **model** which updates the **view** whenever data changes.

(Figure 13: Client.java class view (Generated on NetBeans) . Date Accessed: 22/2/2021)

3.4 Acceptance Tests

The goal of this section is to define the importance of Acceptance testing which is a technique that is performed to determine if the software has met the requirements specification. The purpose of running acceptance tests is to evaluate the application logic with the business requirements and to verify that it has met the required criteria prior to delivering to the end user[4]. This section will provide a summary of the test cases run within the NetBeans IDE.

For this Application, the first test case was run against the main method of the Client class. The client uses the standard input stream as required by the specification which is checked within the testmain() method using the desired input. The results highlighted by figure 14 uses the input command 'exit' which correctly displays the front-end of the command line utility visible to the user.

```
    @Test
    public void testMain() throws Exception {
        String[] args ={"userid", "localhost", "8888"};

        String input= "exit\n";
        ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
        System.setIn(in);//SetsSystem.intothesuppliedstream

        Client.main(args);
        // TODO review the generated test code and remove the default call to fail.
```

```
T1InitializationTests ×   ClientTest ×
Tests passed: 100.00 %
The test passed. (0.101 s)

Hello userid!
Note:  Commands can be abbreviated to any prefix, e.g., read [mytag] => re [mytag]

[Main] Enter command: read [mytag], manage [mytag], exit
> ------------- ---------------- ---------------
```

(Figure 14: Client.java class tests run (Generated on NetBeans) . Date Accessed: 22/2/2021)

The second test case was conducted on the user entering drafting mode which is responsible for allowing the user to add text and push the drafted tines to the server. The results of the test were successful highlighted by figure 15 which correctly display the input 'manage draft' required by the specification to enter drafting mode.

```java
@Test
public void testDraftingMode() throws Exception {
    String[] args ={"userid", "localhost", "8888"};

    String input= "manage draft\nexit\n";
    ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
    System.setIn(in);//SetsSystem.intothesuppliedstream

    Client.main(args);
    // TODO review the generated test code and remove the default call to fail.

}
```

```
T1InitializationTests × ClientTest ×
Tests passed: 100.00 %
The test passed. (0.093 s)

Hello userid!
Note:  Commands can be abbreviated to any prefix, e.g., read [mytag] => re [mytag]

[Main] Enter command: read [mytag], manage [mytag], exit
>
Drafting: #draft
[Drafting] Enter command: line [mytext], push, exit
> ------------- ---------------- ---------------
```

(Figure 15: Client.java class tests run (Generated on NetBeans) . Date Accessed: 22/2/2021)

The third test case was conducted on the user adding text to the server as required by the specification. The results of the test were successful as highlighted by figure 16 which correctly displays the output text using the command 'line Hello world' in drafting mode.

```java
@Test
public void addingtext() throws Exception {
    String[] args ={"userid", "localhost", "8888"};

    String input= "manage draft\n line Hello World! \n exit\n";
    ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
    System.setIn(in);//SetsSystem.intothesuppliedstream

    Client.main(args);
    // TODO review the generated test code and remove the default call to fail.
```

```
T1InitializationTests × ClientTest ×
Tests passed: 100.00 %
The test passed. (0.088 s)

Note:  Commands can be abbreviated to any prefix, e.g., read [mytag] => re [mytag]

[Main] Enter command: read [mytag], manage [mytag], exit
>
Drafting: #draft
[Drafting] Enter command: line [mytext], push, exit
>
Drafting: #draft
          1  HelloWorld!
[Drafting] Enter command: line [mytext], push, exit
> ------------- ---------------- ---------------
```

(Figure 16: Client.java class tests run (Generated on NetBeans) . Date Accessed: 22/2/2021)

The fourth test case was conducted on the user pushing text to the server as required by the specification. The results of the test were successful as highlighted by figure 17 which correctly simulates the output text 'line Hello world' in drafting mode that requires the 'push' command to send the drafted tine to the server.

```java
@Test
public void pushingtext() throws Exception {
    String[] args ={"userid", "localhost", "8888"};

    String input= "manage draft\n line Hello World! \n  push \n exit\n";
    ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
    System.setIn(in);//SetsSystem.intothesuppliedstream

    Client.main(args);
    // TODO review the generated test code and remove the default call to fail.

}
```

```
T1InitializationTests × ClientTest ×
Tests passed: 100.00 %
The test passed. (0.125 s)

[Main] Enter command: read [mytag], manage [mytag], exit
>
Drafting: #draft
[Drafting] Enter command: line [mytext], push, exit
>
Drafting: #draft
          1  HelloWorld!
[Drafting] Enter command: line [mytext], push, exit
>
[Main] Enter command: read [mytag], manage [mytag], exit
> ------------- ---------------- ----------------
```

(Figure 17: Client.java class tests run (Generated on NetBeans) . Date Accessed: 22/2/2021)

The fifth test case was conducted on the user reading text to the server as required by the specification. The results of the test were successful as highlighted by figure 18 which correctly simulates the output text 'line Hello world' in drafting mode that uses the command 'read draft' to display text in the current draft.

```java
@Test
public void readingtext() throws Exception {
    String[] args ={"userid", "localhost", "8888"};

    String input= "manage draft\n line Hello World! \n  push \n read draft \n exit\n";
    ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
    System.setIn(in);//SetsSystem.intothesuppliedstream

    Client.main(args);
    // TODO review the generated test code and remove the default call to fail.

}
```

```
InitializationTests × ClientTest ×
Tests passed: 100.00 %
The test passed. (0.116 s)

>
Drafting: #draft
[Drafting] Enter command: line [mytext], push, exit
>
Drafting: #draft
          1  HelloWorld!
[Drafting] Enter command: line [mytext], push, exit
>
[Main] Enter command: read [mytag], manage [mytag], exit
> Read: #draft
     userid  HelloWorld!
     userid  HelloWorld!
```

(Figure 18: Client.java class tests run (Generated on NetBeans) . Date Accessed: 22/2/2021)

The sixth test case was conducted on the main loop that is used to handle user options , read user input and process the different types of inputs required by the specification. The importance of running this test is to understand that the for loop is responsible for reading and processing requests stated by the specification that demands constant user input and responses accordingly since it is a command line utility. The results of the test were successful as highlighted by figure 19 which correctly simulates the output text 'Testing loop' on an infinite loop that is exited using the return keyword to exit the program faster. The logic showcased in figure 19 is entry controlled by the use of the initialised boolean value 'done' that is set to false and then reversed to true which acts as a finite

state machine that is not supposed to end until the user has reached the end state.

```
@Test
public void testLoop() throws Exception {
  System.out.println("Main loop: print user options, read user input and process");
   for (boolean done = false; !done;) {
       System.out.println("Testing Loop");
       return;
   }
     //CLFormatter helper = String user, int port;
     // BufferedReader reader = null
     // TODO review the generated test code and remove the default call to fail.
}
```

```
ClientTest ×
Tests passed: 100.00 %
The test passed. (0.1 s)
ClientTest passed
testLoop passed (0.0 s)

Main loop: print user options, read user input and process
Testing Loop
```

(Figure 19: Client.java class tests run (Generated on NetBeans) . Date Accessed: 22/2/2021)

The seventh test case was conducted on the user leaving 'user' and 'host' empty (arguments) which is responsible for displaying error message to the server as required by the specification. The results of the test were successful as highlighted by figure 20 which correctly simulates the error message "user\host has not been set" that would be visible to the user within the output window.

```
@Test
public void testRun()
{   System.out.println("run");
     String user = "";
     String host = "";
     System.err.println("User/host has not been set.")
}
```

```
ClientTest ×
Tests passed: 100.00 %
The test passed. (0.1 s)
ClientTest passed
userhostNotBeingSet passed (0.002 s)

run
User/host has not been set.
```

(Figure 20: Client.java class tests run (Generated on NetBeans). Date Accessed: 22/2/2021)

4.0 Overall Reflections

The code review conducted in this report should be as thorough as possible in finding and reporting security weaknesses, but this does not guarantee that every single possible weakness within the prototype is found. For this reason along, this review should be considered a guide where if no issues are found then this does not implicitly mean that the prototype is 100% protected from data breaches.

This code review was finalised on 02/03/2021 by 'Helitha' at which time the weaknesses identified from the source code presented along with solutions alongside a high-level overview of how user authentication: data validation and classifications were implemented in the code. To finalise this

report, the information discussed in previous chapters should be remediated to prevent vulnerabilities and support future functionality.

5.0 References

[1]Jan Lahoda. 2020. Netbeans.org. https://netbeans.org/kb/docs/java/code-inspect.html#:~:text=Open%20the%20library%20project%20in,s)%20with%20the%20FindBugs%20configuration. Date Accessed: 20/02/2021

[2] EasyPMD. 2020. Plugins.Netbeans.org. http://plugins.netbeans.org/plugin/57270/easypmd Date Accessed: 20/02/2021.

[3] FindBugs. 2020. Findbugs.sourceforge.net. http://findbugs.sourceforge.net/ Date Accessed: 20/02/2021

[4] Acceptance testing. 2020. Tutorialspoint.com. https://www.tutorialspoint.com/software_testing_dictionary/acceptance_testing.htm#:~:text=Acceptance%20testing%2C%20a%20testing%20technique,for%20delivery%20to%20end%20users. Date Accessed: 20/02/2021

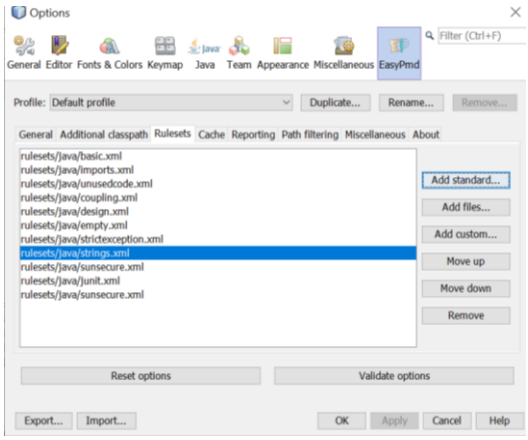
[5] Medium. 2020. GeeksforGeeks.org. https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/ Date Accessed: 21/02/2021

6.0 Appendix A

6.1 Table of Figures

```
try {
  ...
} catch (RuntimeException e) {
  throw e;
} catch (Exception e) {
  ... deal with all non-runtime exceptions ...
}
```

(Figure 21: Client.java class tests run (Generated on NetBeans). Date Accessed: 25/2/2021)



(Figure 22: Adding additional rulesets for EasyPMD (Generated on NetBeans). Date Accessed: 25/2/2021)

Test Results | Action Items | Inspector

Description	File	Location
1 - Avoid throwing raw exception types.	Client.java	...tBeansProjects/TineeTest/src/Client.java:128
1 - Avoid throwing raw exception types.	Server.java	...neeTest/src/sep/tinee/server/Server.java:171
1 - Avoid throwing raw exception types.	Server.java	...neeTest/src/sep/tinee/server/Server.java:233
3 - Document empty constructor	ClientTest.java	...nsProjects/18001953/Test/ClientTest.java:25
3 - Document empty method body	ClientTest.java	...nsProjects/18001953/Test/ClientTest.java:32
3 - Document empty method body	ClientTest.java	...nsProjects/18001953/Test/ClientTest.java:39
3 - Document empty method body	ClientTest.java	...nsProjects/18001953/Test/ClientTest.java:46
3 - Document empty method body	ClientTest.java	...nsProjects/18001953/Test/ClientTest.java:53
3 - The String literal "8888" appears 5 times in this file; the first ...	ClientTest.java	...nsProjects/18001953/Test/ClientTest.java:63
3 - The String literal "localhost" appears 6 times in this file; the fi...	ClientTest.java	...nsProjects/18001953/Test/ClientTest.java:63
3 - The String literal "userid" appears 5 times in this file; the first...	ClientTest.java	...nsProjects/18001953/Test/ClientTest.java:63
3 - Potential violation of Law of Demeter (object not created loca...	ClientTest.java	...nsProjects/18001953/Test/ClientTest.java:66
3 - The String literal "UTF8" appears 5 times in this file; the first ...	ClientTest.java	...nsProjects/18001953/Test/ClientTest.java:66
3 - Potential violation of Law of Demeter (object not created loca...	ClientTest.java	...nsProjects/18001953/Test/ClientTest.java:80
3 - A method/constructor shouldnt explicitly throw java.lang.Exc...	ClientTest.java	...nsProjects/18001953/Test/ClientTest.java:93
3 - Potential violation of Law of Demeter (object not created loca...	ClientTest.java	...nsProjects/18001953/Test/ClientTest.java:97
3 - A method/constructor shouldnt explicitly throw java.lang.Exc...	ClientTest.java	...sProjects/18001953/Test/ClientTest.java:110
3 - Potential violation of Law of Demeter (object not created loca...	ClientTest.java	...sProjects/18001953/Test/ClientTest.java:114
3 - A method/constructor shouldnt explicitly throw java.lang.Exc...	ClientTest.java	...sProjects/18001953/Test/ClientTest.java:127
3 - Potential violation of Law of Demeter (object not created loca...	ClientTest.java	...sProjects/18001953/Test/ClientTest.java:131
3 - Document empty constructor	ServerTest.java	...53/Test/sep/tinee/server/ServerTest.java:25
3 - Document empty method body	ServerTest.java	...53/Test/sep/tinee/server/ServerTest.java:29
3 - Document empty method body	ServerTest.java	...53/Test/sep/tinee/server/ServerTest.java:33
3 - Document empty method body	ServerTest.java	...53/Test/sep/tinee/server/ServerTest.java:37
3 - Document empty method body	ServerTest.java	...53/Test/sep/tinee/server/ServerTest.java:41
3 - Potential violation of Law of Demeter (object not created loca...	ServerTest.java	...53/Test/sep/tinee/server/ServerTest.java:51
3 - Potential violation of Law of Demeter (object not created loca...	ServerTest.java	...53/Test/sep/tinee/server/ServerTest.java:63
3 - Potential violation of Law of Demeter (object not created loca...	ServerTest.java	...53/Test/sep/tinee/server/ServerTest.java:77
3 - Potential violation of Law of Demeter (object not created loca...	ServerTest.java	...53/Test/sep/tinee/server/ServerTest.java:90
3 - Potential violation of Law of Demeter (object not created loca...	ServerTest.java	...3/Test/sep/tinee/server/ServerTest.java:106
3 - Potential violation of Law of Demeter (object not created loca...	ServerTest.java	...3/Test/sep/tinee/server/ServerTest.java:121
3 - Possible unsafe assignment to a non-final static field in a con...	CLFormatter.java	...sProjects/TineeTest/src/CLFormatter.java:17
3 - Avoid unused private methods such as 'send(Message)'.	CLFormatter.java	...sProjects/TineeTest/src/CLFormatter.java:25
3 - Avoid unused private methods such as 'receive()'.	CLFormatter.java	...sProjects/TineeTest/src/CLFormatter.java:29
3 - Avoid appending characters as strings in StringBuffer.append.	CLFormatter.java	...sProjects/TineeTest/src/CLFormatter.java:63
3 - An empty statement (semicolon) not part of a loop	CLFormatter.java	...sProjects/TineeTest/src/CLFormatter.java:67

(Figure 23: EasyPMD Results (Generated on NetBeans). Date Accessed: 25/2/2021)

ource | History

Client > loop > for (boolean done = false; !done;) >

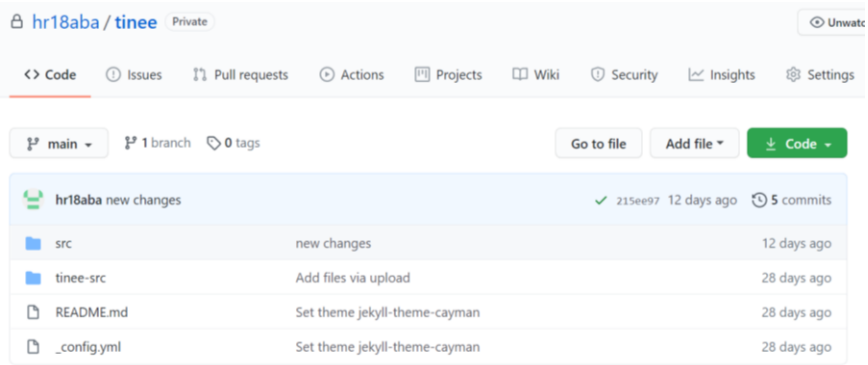
t Results | Action Items | Inspector

Description	File	Location
3 - Potential violation of Law of Demeter (object not created loca...	Client.java	...tBeansProjects/TineeTest/src/Client.java:198
3 - This abstract class does not have any abstract methods	AbstractModel.java	...ineeTest/src/sep/mvc/AbstractModel.java:35
3 - Potential violation of Law of Demeter (method chain calls)	AbstractView.java	...ineeTest/src/sep/mvc/AbstractView.java:126
3 - Potential violation of Law of Demeter (method chain calls)	AbstractView.java	...ineeTest/src/sep/mvc/AbstractView.java:146
3 - Potential violation of Law of Demeter (method chain calls)	ClientChannel.java	...ep/tinee/net/channel/ClientChannel.java:108
3 - Potential violation of Law of Demeter (method chain calls)	ClientChannel.java	...ep/tinee/net/channel/ClientChannel.java:135
3 - Potential violation of Law of Demeter (object not created loca...	ClientChannel.java	...ep/tinee/net/channel/ClientChannel.java:142
3 - Avoid using implementation types like 'LinkedHashSet'; use t...	Push.java	...est/src/sep/tinee/net/message/Push.java:110
3 - Avoid using implementation types like 'LinkedHashMap'; use t...	ShowReply.java	...rc/sep/tinee/net/message/ShowReply.java:53
3 - Potential violation of Law of Demeter (method chain calls)	Pair.java	...s/TineeTest/src/sep/tinee/server/Pair.java:63
3 - Potential violation of Law of Demeter (method chain calls)	Pair.java	...s/TineeTest/src/sep/tinee/server/Pair.java:64
3 - Potential violation of Law of Demeter (method chain calls)	Pair.java	...s/TineeTest/src/sep/tinee/server/Pair.java:81
3 - Avoid unnecessary if..then..else statements when returning b...	Pair.java	...s/TineeTest/src/sep/tinee/server/Pair.java:84
3 - Potential violation of Law of Demeter (method chain calls)	Pair.java	...s/TineeTest/src/sep/tinee/server/Pair.java:84
3 - Avoid using implementation types like 'LinkedHashMap'; use t...	Server.java	...neeTest/src/sep/tinee/server/Server.java:110
3 - Avoid empty catch blocks	Server.java	...neeTest/src/sep/tinee/server/Server.java:153
3 - Potential violation of Law of Demeter (method chain calls)	Server.java	...neeTest/src/sep/tinee/server/Server.java:156
3 - Potential violation of Law of Demeter (method chain calls)	Server.java	...neeTest/src/sep/tinee/server/Server.java:177
3 - Potential violation of Law of Demeter (object not created loca...	Server.java	...neeTest/src/sep/tinee/server/Server.java:198
3 - Potential violation of Law of Demeter (object not created loca...	Server.java	...neeTest/src/sep/tinee/server/Server.java:199
3 - Potential violation of Law of Demeter (method chain calls)	Server.java	...neeTest/src/sep/tinee/server/Server.java:205
3 - Potential violation of Law of Demeter (static property access)	Server.java	...neeTest/src/sep/tinee/server/Server.java:208
3 - Avoid if (x != y) ..; else ..;	Server.java	...neeTest/src/sep/tinee/server/Server.java:209
3 - Potential violation of Law of Demeter (object not created loca...	Server.java	...neeTest/src/sep/tinee/server/Server.java:209
3 - Potential violation of Law of Demeter (static property access)	Server.java	...neeTest/src/sep/tinee/server/Server.java:209
3 - Potential violation of Law of Demeter (object not created loca...	Server.java	...neeTest/src/sep/tinee/server/Server.java:211
3 - Potential violation of Law of Demeter (method chain calls)	Server.java	...neeTest/src/sep/tinee/server/Server.java:212
3 - Potential violation of Law of Demeter (object not created loca...	Server.java	...neeTest/src/sep/tinee/server/Server.java:215
3 - Potential violation of Law of Demeter (object not created loca...	Server.java	...neeTest/src/sep/tinee/server/Server.java:218
3 - Avoid using implementation types like 'LinkedHashMap'; use t...	Server.java	...neeTest/src/sep/tinee/server/Server.java:228
3 - Potential violation of Law of Demeter (method chain calls)	Server.java	...neeTest/src/sep/tinee/server/Server.java:231
3 - Potential violation of Law of Demeter (object not created loca...	Server.java	...neeTest/src/sep/tinee/server/Server.java:246
3 - Potential violation of Law of Demeter (object not created loca...	Server.java	...neeTest/src/sep/tinee/server/Server.java:249
3 - Potential violation of Law of Demeter (method chain calls)	Server.java	...neeTest/src/sep/tinee/server/Server.java:267
3 - Potential violation of Law of Demeter (method chain calls)	Server.java	...neeTest/src/sep/tinee/server/Server.java:275
3 - Potential violation of Law of Demeter (method chain calls)	Server.java	...neeTest/src/sep/tinee/server/Server.java:277

(Figure 24: EasyPMD Results (Generated on NetBeans). Date Accessed: 25/2/2021)

```
$ git log
commit fdd504ef40c92316a3918055bb7c4b5a91748100 (HEAD -> main, origin/main)
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Tue Mar 2 12:27:13 2021 +0000

    finalcommit

commit e68831493bdeeae75766f14e96200a2d327791aa
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Tue Feb 23 18:27:52 2021 +0000

    v1

commit 2bd46e55683c3dcd359147e43e2a075a15a1d602
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Thu Feb 18 19:11:14 2021 +0000

    Initial commit
```

(Figure 25: Final Commit to Git Repository (Generated on GitBash). Date Accessed: 02/3/2021)



(Figure 26: Final Commit to Git Repository (Generated on GitHub). Date Accessed: 02/3/2021)

```
$ git remote -v
origin  https://github.com/hr18aba/18001953.git (fetch)
origin  https://github.com/hr18aba/18001953.git (push)
```

```
$ git push https://github.com/hr18aba/18001953.git
Everything up-to-date
```

(Figure 27: Final Commit to Git Repository (Generated on GitBash). Date Accessed: 02/3/2021)