CS 474 Project: Testing Teammates Software

Marcus Henke
Computer Science Department
Boise State University
Boise, ID, United States
marcushenke@u.boisestate.edu

Oscar Avila
Computer Science Department
Boise State University
Boise, ID, United States
oscaravila@u.boisestate.edu

Dylan Hartley
Computer Science Department
Boise State University
Boise, ID, United States
dylanhartley@u.boisestate.edu

ABSTRACT

This final report for the CS474: Software Quality class at Boise State University documents the challenges of defining and inserting verification properties in an already well-established and large software repository. The requirements for the project involved picking a medium-sized codebase between 8-12 thousand lines of Java code. The team worked on a large-scale web application titled *Teammates*, a program designed to help manage peer evaluations and other feedback paths of students in educational environments [1]. The project required the team to use verification techniques to improve the quality of the program, which proved to be difficult for well-tested and well-designed software. While most of the new testing properties implemented by the team did not provide much help to the already-robust test suite, the introduction of sequencing properties provided the program with a new window of opportunity for verification since it had no sequencing properties before the completion of this project. The overall goal of the project was to allow for teams to develop skills in the field of software quality and to practice defining and implementing verification techniques.

CCS CONCEPTS

Software Validation and Verification • Software Defect Analysis • Software Testing

KEYWORDS

Software, Testing, Teammates, Verification, Validation

ACM Reference format:

Marcus Henke, Oscar Avila Dylan Hartley. 2020. CS 474 Project: Testing Teammates Software. In *Proceedings of ACM Woodstock conference (WOODSTOCK'18). ACM, New York, NY, USA, 2 pages.* https://doi.org/10.1145/1234567890

1 INTRODUCTION

The final project for CS 474: Software Quality required a codebase between 8-12 thousand lines of Java code, which led a quite restrictive and small pool of viable candidates. The goal was to find a set of 20 properties in the program; 5 of which must be sequential. This team's project selection, Teammates, is a massive repository—home to over 95,000 lines of Java code which can be seen further in this document in Figure 3. While it was an easy task to find testable functionalities in this large codebase, it remained difficult to find testable functionalities that weren't already being covered by the program's vast testing package.

The strategy was to first get a high-level understanding of the project and its purpose, then divide and conquer the search for possible testable functionalities by splitting the search by package. When a large enough pool of testable areas was found, the team reconciled and discussed which of the findings to implement for the project.

2 RELATED WORK

The final project for CS 474: Software Quality required a codebase between 8-12 thousand lines of Java code, which led a quite restrictive and small pool of viable candidates. The goal was to find a set of 20 properties in the program; 5 of which must be sequential. For the 5 sequential testing methods our team applied the testing orientation of JavaMOP which is a monitoring-oriented programming environment that is based around java. The team considered testing the non-sequential properties with with Java Modeling Language (JML), a behavioral interface specification language that you can use to specify the behavior of java modules, but for the sake of simplicity the simple Java assertion testing was used in its place.

2.1 MOP and JavaMOP Subset

MOP is an excellent choice for applying a sequential method testing suite to a program in Java. A sequential method test is one that monitors data as it accrues and once the data is gathered it employs optional stopping rules, also known as error-spending functions, which guarantee the overall structure of the program is executing the way that you want. Applying this type of testing method to a program will ensure that certain methods within the program occur at specific times. For example, if you want one method to be executed only after another method is executed then sequential testing will ensure that this occurs, and if it does not then an error will be thrown for the program telling the person executing the code that the methods executed out of order. We applied this specification with a MOP environment to monitor what was sequentially occurring within the program.

We used JavaMOP, which is a subset of MOP—a testing framework that aims to reduce the gap between formal specification and implementation. MOP also contains subsets of BusMOP and ROSMOP where BusMOP is used to monitor system buses using FPGA-based monitors and ROSMOP is integrated with ROSRV which is a runtime verification framework for the robot operating system (ROS). MOP also contains plugins that can be used in other monitoring applications, in which the plugins include finite state machines, extended regular expressions, context free grammars, and others. All these plugins can be applied in JavaMOP whereas in BusMOP and ROSMOP the available plugins are limited. In JavaMOP, monitoring is done at runtime and encourages a fundamental principle for building reliable software. These monitors are automatically synthesized from the specified properties that the developer signifies. These properties are integrated with the original system to check the dynamic behaviors at the time of execution. If the properties are violated or validated at runtime then it triggers a developer defined action, which can be any code that the developer chooses. Some popular decisions that developers choose to integrate are logging and runtime recovery. Figure 1 displays a logic matrix for the MOP languages and plugins, where the logic plugins are Finite State Machines, Extended Regular Expressions, Context Free Grammars, Past Time Linear Temporal Logic, Linear Temporal Logic, Past Time LTL with Calls and Returns, and String Rewriting Systems [4].

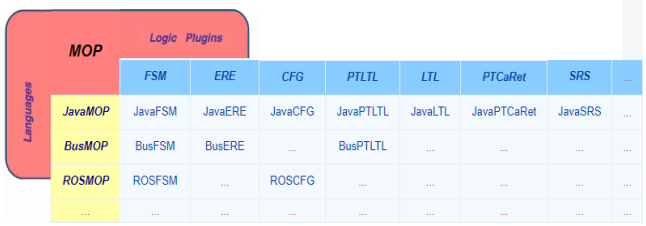


Figure 1: Logic Matrix for MOP Languages and Plugins

To test the other 20 properties that were required for this project our team had the option to work with JUnit tests; where we would have mostly worked with assertions to test input validation for user inputs. JUnit is a developer side testing foundation that is used on the Java Virtual Machine (JVM). While JUnit is up to version Junit 5 our team would have worked with JUnit 4 mostly due to assertions working with JUnit 4 just as well and integrating JUnit 4 into our tests would be easier than using JUnit 5. The reason that JUnit 4 would be easier to integrate is due to JUnit 5 being composed of several different modules from three sub-projects which are the JUnit Platform, JUnit Jupiter, and JUnit Vintage. Since our project was strictly only using assertions, we only needed the original JUnit Jupiter which is the sub-project that provides the test engine that we needed and JUnit 4 satisfied that requirement without the extra sub-projects. Assertions in JUnit supplied most of the testing that we were creating due to their availability for all primitive types, objects, and arrays which covered the input validation that we were testing [5].

2.3 JUnit Asserts

The assertions library within JUnit 4 contains methods that make it easy to test if the end result is actually what you are expecting the code to do. This attempts to stop discrepancies in people's code so that output is what you want it to be, and asserts do this very well with the assert statements that it adds. For example, asserts contains a method assertEquals() which takes the output of the method you are testing and compares it with the exacted output that you pass into it. If this assertEquals() is compared to false, then you can supply it with a trigger which then executes the code that you supply to the trigger. Popular trigger inputs are strings that explain where the error is occurring or a stack trace to point to the line where this assertEquals() is failing. There are many websites out there that will supply tutorials on how to use these assertion statements and what some popular statements that are used when creating tests for a program. If our team chose to make a test suite instead of using Java Assertions within the program, we would change them to JUnit 4 asserts to have a more fully fledge assert system. Comparatively Java Assertions was easier to implement in this case and caused less overhead. Below in Figure 2 are some popular assert statements with examples on how to import these statements into your program if you desire to write a test suite in JUnit [7].

```
import static org.junit.Assert.assertArrayEquals;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertNotSame;
import static org.junit.Assert.assertNull;
import static org.junit.Assert.assertSame;
import static org.junit.Assert.assertThat;
import static org.junit.Assert.assertTrue;
```

Figure 2: JUnit Assertions

2.4 Open Source Project

Our team's open source project selection, *Teammates*, is a massive repository—home to over 95,000 lines of Java code. While it contains over 95,000 lines of java it does not only contain Java but also contains 12.5% typescript, 7.2% HTML, 3.3% JavaScript, and 0.6% CSS where the percentages are based on lines of code [2]. While this project contains more than just Java our team was tasked with just testing the Java part of an open source project so we only looked through the 95,000 lines of code to find methods that would be important to test. The methods that we found to test were ones that mostly handled input from users of the program so that input validation could be done and less vulnerabilities would appear. For example, when the application asks for an email from the user, we take the user input and apply a regular expression to ensure that they are actually inputting an email and not inputting anything that could possibly exploit the system. For sequential testing our process was to find methods that we believed should be called before another. One example of a sequential test that we wrote had

to do with updating a user's account. The test that we wrote ensured that the getAccount() method should be called to collect the user account, before the method of updateAccount(), which would then take that user's account provided from getAccount() and allow for a change to occur on the account.

3 THE PROJECT

3.1 Description

Teammates is an open-source web application project that was designed to help students in all levels of education evaluate their peers and communicate. According to the app's "Features" page, it supports peer evaluations, multiple feedback paths, reports and statistics, profiles, downloadable data, searching, and a no-signup system that allows students to participate without any account creation. The project is sponsored by the School of Computing in the National University of Singapore and is currently ran by a core team of 7 developers. Over 100 people have provided multiple contributions to the project, and over 200 people have provided one-time contributions. The project has received numerous appraisals from university faculty across the globe, and it is also award-winning. Teammates won the "Grand Prize at the OSS Awards World Challenge 2014 and was selected as a mentoring organization for Google Summer of Code Program (2014, 2015, 2016, 2017) and Facebook Open Academy Program (2016)." [3]

3.2 Technical Overview

Teammates is built as two separate applications; a frontend and backend. The backend of the project is hosted with a Java servlet. Data is stored with Google Cloud Datastore and the servlet communicates with the cloud with a "middle-ground" Java data access API called Objectify [10]. The front-end is built with Angular, HTML, SCSS and Bootstrap. When entering the Teammates directory and running CLOC, a command-line tool that evaluates the lines of code in a directory [9], The application's codebase is clearly far more extensive than the requirements specified by the CS 474 project, as seen in Figure 3.

Language	files	blank	comment	code
HTML	570	552	80	503568
Java	758	25556	13990	96650
JSON	74	40	0	60386
TypeScript	516	3614	3492	22558
JavaScript	27	989	542	4954
Markdown	30	596	0	1380
CSS	2	308	73	1244
Sass	72	196	3	992
XML	16	60	61	869
JSP	13	87	8	818
Gradle	1	70	29	530
YAML	5	23	18	362
Bourne Shell	1	22	37	129
DOS Batch	1	24	2	74
SUM:	2086	32137	18335	694514

Figure 3: *CLOC* executed on the *Teammates* Repository

The repository is already home to several JUnit and TypeScript tests that provide roughly 64% code coverage on the entire codebase [1]. Since the CS 474 project required the team to test a Java application, the focus of the CS 474 project was centered on testing the back-end Java Servlet packages and their integration with the Google Datastore. Even when restricted to just 1 out of 14 languages used in the application, there were still over 96,000 lines of Java code to understand before implementing the verification properties.

3.3 Build Process

This section will briefly cover how the Teammates app is built. The app's frontend uses NPM (Node Package Manager) for dependency organization. There are also a few cross-environment dependencies for which the frontend must invoke the backend's Gradle dependency manager to receive these dependencies. Upon installing all necessary packages with NPM and running gradle on the backend, the dev server for the Angular frontend can then be hosted on a local port of the developer's machine [1].

Since the backend dependencies are already installed by running Gradle for setting up the frontend, the backend is ready after building the frontend. If the developer wants his/her machine to host both the frontend and backend of the application, the developer must bundle and transpire the front-end files to be public and ready for use by the backend dev server. When this Node command is complete, the backend dev server can then also serve the front end simultaneously [1].

4 TESTING

4.1 Testing Setup

For testing as mentioned in section 2.2 we used JavaMOP. To install JavaMOP the installation instructions are provided in reference [8]. As mentioned in the instructions the only prerequisites for using JavaMOP are having a Java Runtime Environment of 7 or later, Aspect J compiler 1.8.1 or later, and Runtime verification Monitor 1.3 or later [8].

The JavaMOP files contain calls to methods in other java classes. These are the methods we are observing. We want to make sure these methods get called in a certain order or not get called in a certain order. For this reason, the mop files also contain a regular expression of that order we're trying to observe. If the order is observed, we can set a trigger and count the amount of times it's observed or set the trigger to do something else like stop the execution of the program. To set this up, once the methods we want to observe are chosen and the regular expression is built we change the corresponding .mop file into actual usable code. The project must have AspectJ capabilities, or those capabilities must be enabled. Then using the rv-monitor jars and AspectJ we can obtain the .java file made from the .mop file. This new .java file is then weaved into the code and will take care of observing the methods and their sequence.

The Setup process for the Java assertions is much easier. We only need to pass a command line argument or set up the argument in an IDE. For the testing of the project we ran the assertions off the command line, so we chose the command line argument. This is

simply done by passing "-ea", which enables the assertions in the code. If this argument is not passed, then the assertions are ignored. This makes it so assertions can be left in the code even during production without any problem.

4.2 Assertions

We created 15 different assertions for this project. We focused on finding places where assumptions had been made in the code that were not checked. There are many places where null values were not being checked for that could potentially break the program, but our group had to cover multiple different assumptions per the CS 474 project requirements, so we overlooked many of those potential null values. One of the biggest vulnerabilities we saw and were checking for was escaping special characters. The developers for this project created their own method for escaping special HTML characters. This is interesting because there are already libraries and tools out there which do this. For this reason, we thought it was important to check the output of these methods to make sure the escaping of characters was done correctly. Two of these java assertions are testing the methods created for escaping HTML and HTML tags. There were other occasions of this possible vulnerability but again we wanted didn't want to spend all our time checking for these types of errors. It's also important to note that for both assumption tests we had to create new methods to call as we could not have done it with just the java assert tests.

Another property that is never tested is an account's permission level. This open source software is used for receiving feedback on classes. A class can have multiple students, teams, and an instructor. However, when an account is assigned to be an instructor for the class the account is never verified to be an instructor account. If a student is accidentally assigned to a class as the instructor there would be no warning or any errors. However, that student would have access to the information for that class and could get permissions to alter that data. For this reason we decided to create a java assert test which simply checks if the assigned account is an instructor account.

All the properties being verified with Java Assertions can be found in the following classes:

- StudentProfile.java
- CourseSummaryBundle.java
- CourseRoster.java
- SanitizationHelper.java
- FileHelper.java
- NationalityHelper.java
- FeedbackSessionDetailsBundle.java
- CourseDetailsBundle.java
- EmailSendingStatus.java
- GateKeeper.java
- AccountsLogic.java
- CommonAccountSearchResult.java

Please note that there are multiple assertions within some of the listed Java classes.

4.3 Sequencing Properties

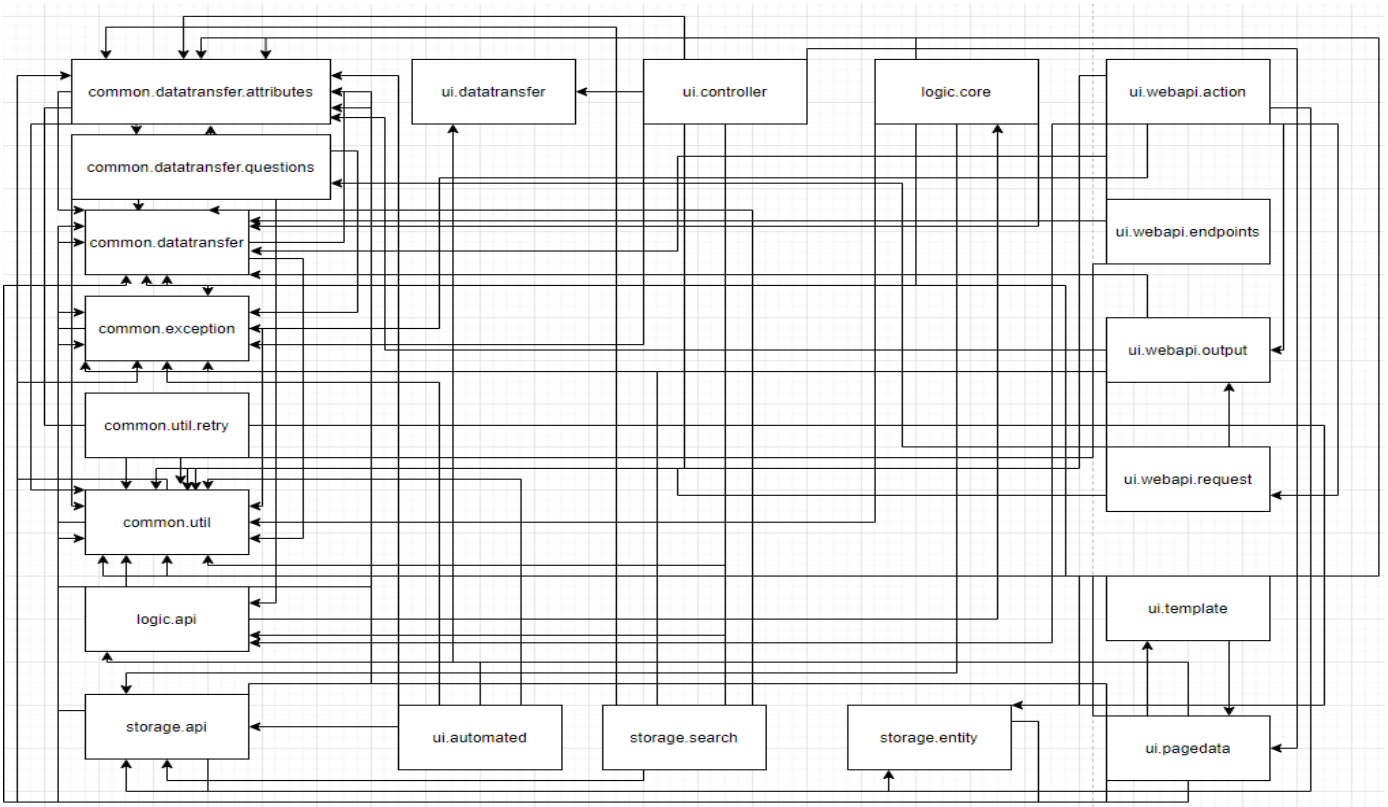
Finding sequencing properties was much more difficult than finding assumptions. This is because we must look for places in the code where methods being called in a specific order is important.

Many of these sequencing properties arise when it comes to using objects. For example, the project has *Logger* objects which are instances of the logger class. The Logger objects are of course used to log important messages such as errors, warnings, and progress notifications. However, in multiple locations these Logger objects are called without the ensuring the logger is first instantiated. Therefore, one of our sequencing methods ensures that getLogger() is called before any of the other logger methods are called. This is important as calling a logger without the logger object being initialized can throw errors or break the program.

Another important location for sequencing is when updating or deleting objects. For example, in AccountsDb.java we want to make sure getAccount() is called before updateAccount() gets called as the wrong account might get updated if the user doesn't make sure the right account is being accessed. This is also the case when deleting a student account. Under ProfilesDB.java there's a deleteStudentProfile() method. Of course, as this method implies it deletes a student's profile. For this reason, we want to make sure that getStudentProfile() is called before the student gets deleted. If the student profile isn't selected there will be an error thrown. The wrong student may also get deleted and it would take a significant amount of work to create that student again, assign them to the correct classes, teams, and to give them the correct privileges.

All the JavaMOP files can be found in the test repository under:

- storage/search/QueryBuilder.mop
- storage/common/util/Logger.mop
- logic/core/DeleteProfileMop.mop
- storage/api/deleteStudentProfileMop.mop
- storage/api/updateAccountMop.mop



4.4 Testing Diagram

This diagram was made to show the complexity within the project. The nodes are all the packages within the main java directory. It also shows us how most of the packages have calls to methods within the common package. We wanted to spread out our tests so we didn't just focus on testing within the common package.

6 CONCLUSION

For a program as complex as this, it requires a user interface that is easy to use as not all teachers have extensive knowledge on how to use programs if they are too complicated. Testing is important for ease of use to ensure the program is working the way that the developers intended. Along with it working correctly, it is important to ensure that the vulnerabilities in the program are at a minimum as the information of many students and teachers will be on the line. To ensure the vulnerabilities are at a minimum, tests are required to check said vulnerabilities. Tests can also make it a lot easier to check if certain vulnerabilities are patched. With a program like this our team viewed it as very good candidate in which to write the tests that were required for our project. After reviewing our list of possible candidates to test, we found that *Teammates* was the best candidate for what we wanted to accomplish.

With *Teammates* containing almost one-hundred twenty thousand lines of code with seventy-five percent of the codebase being Java we found it rather difficult to hunt for locations in the code that would be valuable to test. With little knowledge of the codebase the search for reasonable things to test was especially difficult. The most difficult part of finding reasonable tests was on the sequential test end mostly due to not having much knowledge on the codebase. The reason that this was the most troubling was because at times we had to understand how different classes and methods interact with each other to create these different tests. These sequential tests required us to know when certain methods should be called before another, and with twenty different packages interacting with each other as seen in section 4.4, it was the most difficult part of the project.

Another aspect that made this project somewhat difficult to complete was the fact that *Teammates* already had a large test suite so finding properties that were not already being verified was rather difficult. With this large test suite our team decided to write tests that checked to see if inputs to certain methods were what they were supposed to be, and to ensure that no incorrect data or potential attacks would happen with certain method calls. Much of this was checking to see if inputs were not null or that the arrays being passed contained what they were supposed to contain. Some of our tests also included testing the format of inputs such as emails which required regular expressions to complete. The final thing that we tested for was to check if the inputs contain characters that would indicate an injection attack, characters like forward or backslashes

and less than or greater than characters. These are popular characters that would indicate an attack as backslashes would indicate looking for a possible directory while checking for less than and greater than characters will ensure that somebody is not trying to insert a script tag as an input which would allow for a script to be run on the system which could cause many unwanted things to occur.

Overall, this project was very helpful in teaching us how to contribute to a large open source project and help develop and provide some helpful tests that the original developers have not accounted for. This also helped us learn how to learn how a large-scale application works, as well as understand the overall scale of the application. Also, this project significantly helped us to understand how sequential testing works and how to apply it to a real-world application that does not currently use it. Despite the challenges of implementing all these verification properties, the experience and lessons learned from this project were worth the struggle.

REFERENCES

[1] "Teammates". *GitHub*, GitHub. 4 May 2020. https://github.com/TEAMMATES/teammates

[2] Henke, Marcus, et al. "Project Proposal" *Boise State University,* Blackboard, 08 Mar. 2020. https://blackboard.boisestate.edu/webapps/assignment/uploadAssignment?content_id=_7184285_1&course_id=_97546_1&group_id=&mode=view

[3] "Teammates". *National School of Computing*, University of Singapore, n.d. https://teammatesv4.appspot.com/

[4] "Monitoring-Oriented Programming." *FSL*, Formal Systems Laboratory , 21 Nov. 2014, fsl. cs.illinois.edu/index.php/Monitoring-Oriented_Programming.

[5] Bechtold, Stefan, et al. "JUnit 5 User Guide." *JUnit 5 User Guide*, 21 Mar. 2020.

[6] "The Java Modeling Language (JML)." *The Java Modeling Language (JML) Documentation Page*, Source Forge, 14 Dec. 2017.

[7] Stefański, Dariusz A. "Junit-Team/junit4." *GitHub*, GitHub, 19 Mar. 2016, github.com/junit-team/junit4/wiki/Assertions.

[8] Illinois Computer Science. "JavaMOP4" *University of Illinois*, 14 Jul. 2015, http://fsl.cs.illinois.edu/index.php/JavaMOP4

[9] Danial, Al, et al. "Cloc" *SourceForge*, SourceForge. 14 Jan. 2017, http://cloc.sourceforge.net/

[10] "Introduction to Objectify", *GitHub*, GitHub. 05 May 2016, https://github.com/objectify/objectify/wiki