**City University of Seattle**

**CS 504 - Software Engineering Summer 2021**

**Independent Project 4**

**Vaccine Scheduler**

**Anil Erturk**

**Introduction**

In order to understand software test metrics, the purpose of software tests needs to be understood first. The purpose of software testing is not to make the software bug-free. With the size of software projects, that is impossible. The purpose of software testing is systematically checking the software to ensure that it is above a certain quality. The amount of resources that are spent on testing determines the quality to be ensured. The more tests there are, the more situations can be checked. Also, the tests are not used one time. After changes to the software, the team can run the tests again to ensure that the quality of the software didn’t decrease. This is called regression testing.

So, since the purpose of software testing is “systematic checking of the software to ensure that it is above a certain quality”, metrics are required for testing to be quantifiable, systematic, and objective.

**Test Metrics**

Test metrics are generated at the end of the development cycle. The test metrics tell the team how much of the code was tested, how many different cases were tested, the percentage of passed tests to failed tests, etc. These results tell the team whether the code was tested enough and how well the code has performed. This also allows the team to determine whether they successfully completed tasks that correspond to the requirements.

Test metrics also allow the team to determine how much resources are required for fixing the bugs.

Analyzing the code regularly and systematically will also allow the team to recognize patterns and avoid them in the future. A great example of this is the histogram of exceptions that happen in software. This lets the software team know the most frequent errors so the software team can keep especially those errors in mind during development and testing (Rollbar, 2021). Some of the common exceptions are NullPointerException, ArrayIndexOutOfBoundsException, and ArithmeticException. The NullPointerException is thrown when a java program attempts to process an object which contains a null value. The ArrayIndexOutOfBoundsException occurs when asking for a position that does not exist within the size of the array. The ArithmeticException occurs when an exceptional arithmetic condition has occurred. For example, this type of exception often happens when a program attempts to divide by zero.

Test metrics will test functional and non-functional requirements. For every method, there will be an acceptable run time. If the method takes too long, the user experience will suffer. During testing, if the method runs slow enough for the user to notice it, the development team will address the issue by either moving away steps from that section of the software or by optimizing the steps.

Test cases will also test for other non-functional requirements such as common errors. In the case of the vaccine scheduler web application, the special characters in inputs texts need to be escaped to prevent them from causing problems. The common exceptions that were mentioned earlier will also need to be tested.

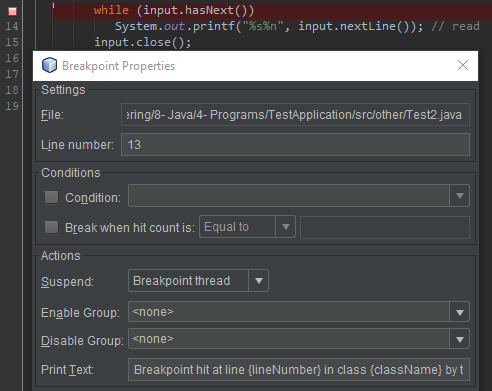
Test cases also need to cover functional requirements such as user stories. The functionality tests must pass. Non-critical errors such as front-end issues are tolerable. When it comes to the vaccine scheduler, the functionalities such as giving the user the list of closest vaccination locations sorted ascending by distance. Another functionality test will be the user being able to choose all the combinations of legal options given to the user and being able to progress through the website. The user should be able to reserve a schedule for all available vaccines. The user will be able to select different types of COVID-19 vaccines. The user should be able to select which dose to take. The user should be able to select a date and time slot. The users should be able to enter their information. And the user should be able to submit all these choices to the server.

Another important test metric is the rate of users who find the software intuitive. One big example of bad GUI design is using toggle buttons that look like normal buttons. These toggle buttons change color when clicked, otherwise, they look just like buttons. This confuses the user because the user thinks the toggle button is a button. Instead of using toggle buttons that look just like normal buttons, toggle buttons that have a circle that slides to left and right when clicked is a better design. The left and right states of the toggle button should be distinct in color to let the user know whether the toggle button is enabled or disabled.

Another test metric will be the number of tests per class/method/line. As expressed earlier, the purpose of software testing is to ensure a certain level of quality. The same level of quality will often require a similar rate of test to code.

Another test metric will be the number and rate of bugs and failed tests per test. No metric alone shows the whole story. The team might get a low bug and failed test rate. But that might be caused by the low number of tests. Therefore, it is important to have multiple metrics that check each other. For example, in science, an experiment might produce a certain result, but this could be a fluke due to the small sample size.

Another test metric will be class/file coverage and line coverage. Coverage means hitting that entity at least once. A histogram for these coverages could be used alongside the coverage metric to make sure the line is being tested in as many scenarios as possible and not just one. Coverage of course is an important metric because a line of code can not be tested if it is not executed. Coverage tells the team that this section was tested at least once. Again, this ensures the team that the software is above a certain level of quality. If the team is expected to be at an even higher quality, the coverage metric can be supported with a histogram metric that shows how many times that line or class was hit. IDEs (Integrated Development Environment) give the developers testing and debugging tools. Developers are capable of setting up breakpoints that execute at every hit, at a certain number of hits, or only when another condition is met.



**References**

Rollbar. (2021). Most Common Java Exceptionsy

https://rollbar.com/blog/most-common-java-exceptions/