CS 3250: Software Testing Assignment 11
Names: Zertash Zahid, Zoha Husain, Eza Rasheed
Computing IDs: zz9ek, zh2yn, er6qt

Option 3

We knew that software testing was very thorough but the fact that it was broken up into different certification levels was very new to us. We learned that the first level of test certification was the most basic level, which would be easy to achieve by one or two engineers, and was primarily there to create different size tests and identify flaky, nondeterministic tests. It focused on establishing the use of tools and baseline measurements (ex. code coverage, identification of broken and "flaky" tests). It was the initialization to measure the state of affairs. The second level was a little more strict, not allowing any code to be submitted that was not fully tested yet. It was more in-depth than the first level because it required certain metrics to be met but did not indicate full coverage level testing. It focuses on adopting and implementing a written policy that requires tests for all new and changed code, and sets easily-reachable test coverage goals. The third level is the highest level of test coverage(in most cases, although a level four and five have been added for more detailed metrics), and it makes sure high coverage is achieved for developer tests, has individual coverage levels, and produces productivity benefits. This is the best type of testing model to have in a long-term environment.

We all acknowledge the importance of certification levels and implementing them because although there are different roles, testing is an unavoidable aspect of development and as future engineers, whether or not we will be directly involved with testing this knowledge is very essential to developing software and its quality. Whether we are working at a startup or a well-established company, it is the job of every employee who is involved in these processes to work towards making sure everything runs efficiently, effectively, and smoothly.

Through the text, '*How Google tests software*' we identified 3 roles: software engineer, software engineer in test, and a tester. Software engineer (SWE) is the traditional developer role. SWEs write functional code that ships to users. They create design documentation, choose data structures and overall architecture, and they spend the vast majority of their time writing and reviewing code. The software engineer in test (SET) is also a developer role, except his focus is on testability and general test infrastructure. SETs are partners in the SWE codebase, but are more concerned with increasing quality and test coverage than adding new features or increasing performance. SETs write code that allows SWEs to test their features. SETs are developers who provide testing features. The TE(test engineer) role puts testing on behalf of the user first. TEs organize the overall quality practices, interpret test results, drive test execution, and build end-to-end test automation.

In this case, making sure that testing standards are up to the best of our ability will significantly impact how well a product does. For example, this summer, I interned as a solutions architect. It was my team's job to demo the software to potential clients as well as set up proof of concepts in client labs. Every quarter or so, there would be new

updates that were rolled out as the software was updated to the newest edition. I went to a meeting with the engineers and heard them talking about testing and working on bugs in the software before the release date. Sometimes, they had to push the date back because they were not done testing. Although I don't know the specific details of how exactly they tested the software, I think they were between a Level 2 and Level 3. They made sure no software was released until it had met certain guidelines. However, I do think they could benefit from having specific coverage tests and goals for each type of test. This would lead to the most overall success. Since I will not be working as part of the engineering team, I don't know how I would improve this process. However, personally, I was responsible for developing code for our testing environment. For example, I worked with Python and shell scripting languages to make certain back-end processes faster. One specific task that I worked on was resetting user sessions through a script instead of having to go through the front end. When I developed the code, it was mostly trial and error and I ran different tests on the demo software on my computer. I'm pretty sure this type of testing would fall under Level One. Although I tried to diversify my tests, and work with inputs of different sizes, I did not know anything about coverage levels so did not have any coverage level goals. My goal was to reduce the time it took so that was a metric I worked to meet. I could definitely improve my coding and developing skills by working on creating coverage criteria for the tests and working towards meeting them before any code is released to the demo environment shown to clients. An important aspect from Google's testing practices that I think could be applied in this case is testing for quality. Instead of just testing to detect errors in the code, tests should be created to prevent any future problems. Also, Google's approach is vastly different from other companies in a few different ways. First most, google has a few dedicated testers on a single team in comparison to other teams in companies. This has to do with the fact that everyone who writes code at Google is a tester, and Google engineers prefer quality over features.

Furthermore, having a structure in place (similar to a focus area at Google) could be useful to make sure there is a higher authority checking over tests. The unorganized structure could be due to us being interns and checking over our work in our own teams but being aware of how to properly test will be useful to apply in the future for full-time jobs. Furthermore, by being a thorough software tester and developer as an intern is also important because then the work can be used in real projects rolled out by the company.

Another important point used by Google testing is that instead of distinguishing between code, integration, and system testing, Google emphasizes scope over form. This allows there to be a common language that all developers and testers can understand across the company. This will allow easy communication regarding testing. Although this is not necessary to implement in our own testing and in our future careers, we think it's important to have open communication about testing and a common way to discuss it. Finally, the best way to test both small-scale and large-scale code is to try to automate it. Although manual testing does need to be used occasionally,

we think it's important for us to remember to try to automate as much as possible to save time and money.

Additionally, a Google practice that stood out to me that could be very practical and efficient if applied to my company is re-defining the tester's role. According to Google. Testers are assigned by Engineering Productivity leads who act strategically based on the priority, complexity, and needs of the product team in comparison to other product teams. Thus, by doing this, Google creates a balance of resources against actual and not perceived need. As a central resource, good ideas and practices tend to get adopted companywide.

Furthermore, Google often builds the minimum useful product as an initial version and then quickly iterates successive versions allowing for internal and user feedback and careful consideration of quality with every small step. Products proceed through canary, development, testing, beta, and release channels before making it to users. Also, the crawl, walk, run approach gives Google the chance to run tests and experiment on our applications early and obtain feedback from real human beings in addition to all the automation we run in each of these channels every day. By implementing these approaches that Google uses in my company, my company would have the opportunity to grow and gain efficiency.

Assuming that our current qualifications place us at level one of the Test Certified program, there are a variety of things we can do to move up to a higher level to improve our testing practices and code quality. In order to attain our goal to fix all broken code/tests and increase productivity and coverage by x amount to reach high coverage, we as a group first need to introduce improved unit testing practices that follow the path set by the Test Certified program, later addressing all sizes: medium(integration) and large(system). Our focus should be on fixing broken tests and writing new tests for uncovered code. We need to ensure that the code we write remains readable to other developers, in a known uniform language, depending on the company conventions. Taking into account Test Certified level two requirements, it would be helpful to create a written policy for development that states how every code change need be accompanied by tests, excluding minor refactorings that do not change existing behavior of code that is already covered. It is then essential to ensure that all the tests ran have passed, especially when changes have been made in response to review comments made previously. Finally, it is crucial to implement automated testing, which includes high levels of unit test coverage. This enables high productivity and addresses more challenging defects with a greater degree of confidence and speed. In tackling all of the above strategies to attain a higher level in the Test Certified program, it is important to note that organization is key. This means that throughout all these processes, every team member should make sure other people can see when "the build" breaks. In essence, it should be easy for others to know when and where there is a problem when testing so that the person handling that specific breakage (avoid duplicate effort) can effectively refactor the code for problematic bugs. Following this plan, level three can progressively and collaboratively be reached, depicting the model of a smooth-running testing process.