**Software Engineering:**

**Test Plans & Test Cases**

Samantha M. Hipple

School of Technology and Computing, City University of Seattle

CS 504: Software Engineering

Kendra Schraml

May 27, 2022

**Software Engineering:**

**Test Plans & Test Cases**

Within the field of software engineering, testing plays an extremely vital role in ensuring that a software product does what it is supposed to do. Computer scientist, Tom Kilburn, is credited with writing the first piece of computer software back in 1948. Kilburn’s program was designed to perform mathematical calculations using machine code instructions. Alongside this initial product, came our first introduction to software testing in the form of debugging ([source](https://www.ibm.com/topics/software-testing)). Debugging is the process of isolating and fixing any faults or errors found within a software system and was the main method of testing during the industry’s first few decades ([source](https://www.ibm.com/topics/software-testing)). In modern day practice, debugging is not even considered a part of software testing, but instead is applied after testing has discovered any errors or bugs in the system.

**Traditional vs Continuous Testing**

Traditionally, the software testing process was separate from the rest of development and performed near the end of the SDLC, after the product has been built and executed. Oftentimes, software testers were only given a short window of time in which they could test the final product before the expected release date. Any defects found would then cause either a need to postpone the release date to fix the errors or a need to push the release of a defected product. Moving the testing activities to earlier phases of the SDLC has helped keep testing efforts a priority throughout development, instead of as an afterthought to development ([source](https://www.ibm.com/topics/software-testing)).

By the 1980s, software development teams began moving away from only focusing on fixing software bugs, to testing their applications in real-world settings. By the 90s, software testing had been integrated into a larger, overall process called quality assurance ([source](https://www.ibm.com/topics/software-testing)). Software quality assurance (SQA) is a focus throughout the entire software development process that includes: (1) a quality management approach that matches planned engineering activities with SQA team members of equal or greater skill-level, (2) pre-determined check-points to evaluate project performance data, (3) incorporation of a multi-testing strategy, (4) measuring change impacts to ensure that any fixes remain compatible with the whole project, and (5) managing good relations within the working environment to keep communication lines open and honest between teams ([source](https://www.geeksforgeeks.org/software-engineering-software-quality-assurance/)).

As Agile and DevOps development strategies were embraced by enterprises worldwide for their ability to enhance competitive advantages through improved delivery speeds and product quality, there was an added focus on the significant, positive impacts of continuous integration (CI), continuous delivery (CD) and continuous testing (CT) throughout the SDLC. Although the importance of CI and CD tools and implementations by the development team should not be understated, without CT ensuring that the integrated changes are compatible with the overall system, CI and CD would be meaningless. Continuous testing is comprehensive in its scope - including teams, tools, testers, and services - and puts in place processes, systems, and automation that enables an accelerated time to market; a constant feedback loop from within and outside (e.g., end-users) the development teams; and desirable business outcomes such as the development of high-quality products and services, operational efficiency, responsiveness, competitive differentiation and enhanced customer service ([source](https://devops.com/continuous-testing-complementary-to-agile-and-devops/)).

**Why is Software Testing Important?**

It is hard to argue against the need for quality control measures when building software products. Mistakes as simple as a typo, a misplaced indent or curly bracket, unexpected user inputs, etc. can easily break an entire program if they are not corrected prior to integration. Modern day customers expect software that is able to operate across multiple platforms, devices, browsers, and networks; applications that are high-performing, navigable, scalable, secure, user-friendly, and fast-loading. Late deliveries and software defects can have a significantly negative effect on customer confidence in a software company.

Although software testing does create an additional startup cost for the development company, the money saved by identifying and fixing defects before a product is deployed can be exponential. The earlier a system fault is discovered in the software development life cycle (SDLC), the cheaper is it will be to fix. “IBM estimates that if a bug costs $100 to fix in Gathering Requirements phase, it would be $1,500 in QA testing phase, and $10,000 once in Production” ([source](https://smartbear.com/blog/software-bug-cost/)). When the development process leaves ample room aside for testing, high-quality, reliable software applications can be delivered with few errors.

**Types of Software Testing**

The two main categories in software testing are (1) functional testing and (2) nonfunctional testing. Functional testing inspects each and every functional component of a software system to ensure that the product does what it is designed to do. Nonfunctional tests are performed after functional testing and focus

**Figure 1**

*Types of Software Testing Diagram (*[*source*](https://hackr.io/blog/types-of-software-testing)*)*

A picture containing diagram

Description automatically generated

on polishing the end product. Figure 1 above provides a visualization of what types of tests are commonly involved in these two categories ([source](https://hackr.io/blog/types-of-software-testing)).

Following the diagram from left to right under the functional testing branch, we can see the scope of each subsequent strategy incorporates more and more of the overall system. Unit testing focuses on validating the smallest, testable components of the overall system (e.g., class methods within a module). Integration testing is meant to ensure that the software’s units function appropriately when working together. Regression testing ensures that newly added features do not break or degrade the system’s overall functionality. Lastly, acceptance testing is done to ensure that the whole system works as intended. Functional tests can be performed either manually or with the use of automated tools ([source](https://hackr.io/blog/types-of-software-testing)).

In review of the nonfunctional testing branch, we see that performance testing contains a sub-branch that includes load testing, stress testing, scalability testing, and volume testing. This is because performance testing is meant to verify how the system performs under different workloads. Load testing is used to determine how much load a system can handle before performance begins to degrade. Stress testing is used to determine how much strain the system can handle before it fails. Scalability testing checks how the system manages increases in the number of users, data, and traffic ([source](https://hackr.io/blog/types-of-software-testing)).