Creating a Test Plan

CS 416 : Software Engineering II

The standard waterfall method of software engineering [1] [2] (Requirements → Design → Implementation → Verification → Maintenance) applies not only to writing software (or building a house or designing a car), but also to verifying software. In other words, a Quality Engineer (referred to as a "**tester**" for the remainder of this article) needs to create a plan before any testing is actually performed.

The Testing Process

The testing process follows the same basic waterfall procedure as it does for the software development process. These phases are called Concept, Requirements, Design, Implementation, Test, Installation & Checkout, and Operation & Maintenance [3].

Concept

The concept phase of the testing process is the preparation phase, when the team is assembled and resources are collected. For small teams or small projects, this phase can be skipped. Common tasks in the concept phase include:

1. Verification and Validation Task: What is the testing process designed to accomplish?
2. Methods and Criteria: Is this manual, automated, or is some other method used?
3. Schedule: When should the test be conducted?
4. Resources: What is needed to conduct the test?
5. Risk and Assumptions: Are there any possible ramifications of the test?
6. Roles and Responsibilities: If more than one person is to conduct the test, what will each person need to do [3]?

It is not uncommon for the management team to conduct the concept phase whereas the engineering teams conduct the rest of the phases of the project.

Requirements

Recall the IEEE definition for **quality** as it pertains to a software project:

> (1) The degree to which a system, component, or process meets specified requirements.
>
> (2) The degree to which a system, component, or process meets customer or user needs or expectations [4].

This means that the definition of quality for a given software project is closely coupled with the task the software is designed to perform. In other words, it is impossible to test software without first understanding what it is supposed to do. The tester needs to fully understand the requirements and the context in which the software will be used [5].

In some situations, the tester plays an active role in the requirements phase of a project. The tester might be responsible for verifying the quality of the specification or requirements document as well as the quality of the software project itself.

Design

The design phase of the testing process mirrors the design phase of the software development process in that the deliverables consist of a plan. For software developers, that deliverable is often called a design document. For testers, that deliverable is called a **test plan**. The IEEE definition of a test plan is:

> (1) A document describing the scope, approach, resources, and schedule of intended test activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning.
>
> (2) A document that describes the technical and management approach to be followed for testing a system or component. Typical contents identify the items to be tested, tasks to be performed, responsibilities, schedules, and required resources for the testing activity [4].

In other words, testers do not rely on serendipity to find defects. Instead, the test plan outlines a systematic approach to evaluating the quality of a software product. It should provide convincing evidence to the reader that, if the outlined procedures are followed with exactness, the important defects of the product will be found in a timely manner.

Implementation

The implementation phase begins when the test plan is executed. This means that the tester fully describes all the individual test cases identified in the test plan. According to IEEE, a **test case** is:

> (1) A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.
>
> (2) Documentation specifying inputs, predicted results, and a set of execution conditions for a test item [4].

Each test case should be designed to provide feedback on the quality of one aspect of the software system. These are referred to as metrics by McCall, where there may be more than one test case for a given factor [6].

Test

When the tester executes the test plan, he or she conducts all the individual test cases. This could mean manually performing the test as if the tester were the user. It could mean writing software called automation that conducts the test case for the tester. It could mean creating tools that interface with the system in such a way that manual testing or automation can conduct the test case. This, of course, is the activity one normally associates with testing a software system.

Installation & Checkout

The installation and checkout phase of the testing process is perhaps the phase with the most misleading name. The purpose here is to generate anomaly reports and summary reports to the members of the engineering team and the stake holders.

The first and perhaps most important deliverable is a **test report**. According to IEEE, a test report is:

> A document that describes the conduct and results of the testing carried out for a system or component [4].

In other words, a test report is a list of defects or bugs found during the execution of the test plan. These are typically sent to the software engineer or to the program manager as soon as they are found.

It is also necessary to generate a summary report detailing the overall quality of the system. Here the tester "signs off" on the project, indicating that the system is at the accepted level of quality and thus fit for delivery to the customer.

Operation & Maintenance

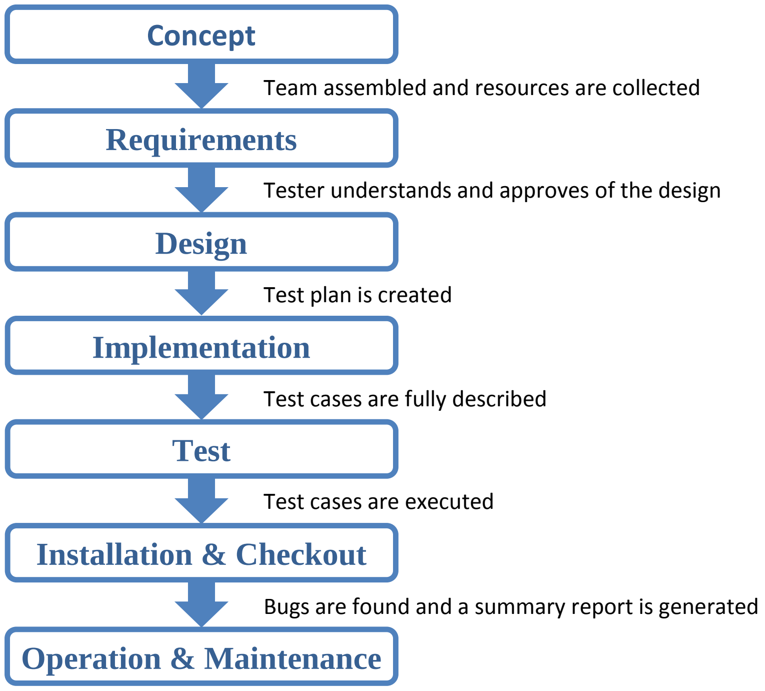
It is seldom the case that the development effort of a software system ceases when the system is delivered to the customer. The process of fixing defects, changing functionality, and adding features after delivery is called maintenance. With each of these changes, the testing team needs to re-evaluate the resulting quality. Sometimes this requires writing or performing test cases on small parts of the system, sometime this requires the entire system to be re-evaluated.

To facilitate the re-testing of a system, it is desirable that the test plan have the following properties:

- **Understandable**: If a tester other than the author needs to implement or modify a test case, this tester needs to understand the test case. Documentation, unambiguous instructions, and standard use of terminology or procedures help with the understandability of test cases.
- **Valid**: Each test case should map directly to a component of the requirements. Thus, if a test case were to fail, it would be easy to see how the failure is tied to a customer or user need.
- **Reliable**: If a test case has a large number of false-positives (instances when errors are reported when none exist), false-negatives (an error exists that is not reported), or the automation fails due to errors in the automation software itself, then the cost of re-implementing a test case can be high.
- **Efficient**: A well-written test case should execute with little effort or, better yet, automatically. The longer it takes to execute a test plan, the greater the maintenance cost.

Summary

The entire testing process can be summarized as follows:


Concept
Team assembled and resources are collected
Requirements
Tester understands and approves of the design
Design
Test plan is created
Implementation
Test cases are fully described
Test
Test cases are executed
Installation & Checkout
Bugs are found and a summary report is generated
Operation & Maintenance

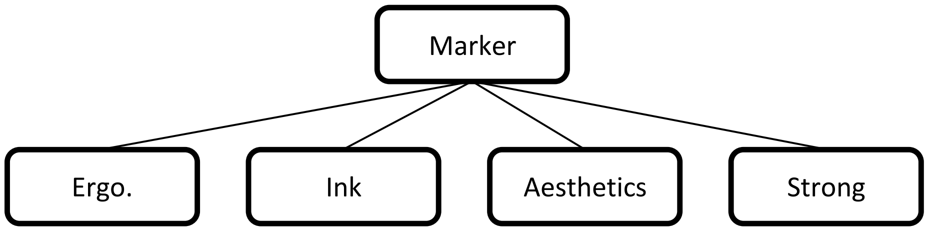
Creating a Test Plan

The core activity of the entire software testing process is the creation of the test plan. Without a good plan, it is unlikely that the tester will be able to provide good data on the quality of the system. How, then, does one go about creating a test plan? This will be answered two ways: first by example then with a detailed explanation.

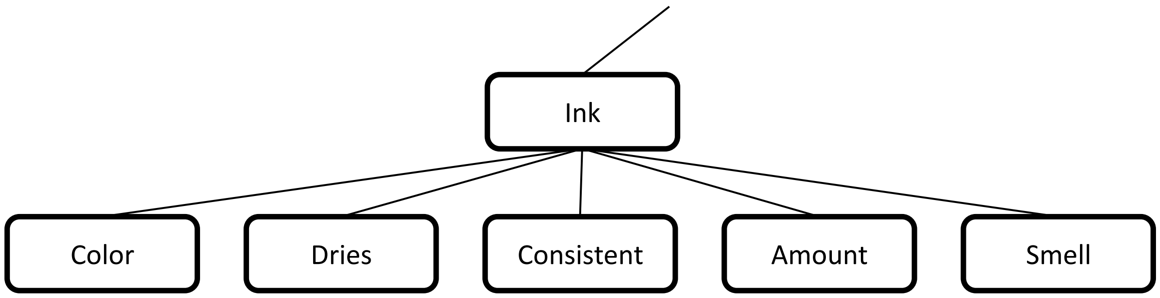
Example: Testing a Marker

Imagine you were asked to create a test plan for a standard dry-erase marker. The expectation is that "all" the defects in the marker will be found once your test plan is implemented. Where do you start?

Of course, you start with the requirements. In the requirements document, you discover that the pen has to be ergonomic (fits nicely in the hand), have high-quality ink, be aesthetically pleasing (look nice), and be strong. Start by making a graph of these components:



With all the bases covered, it is necessary to dive into more detail. How does one test the ink? With more digging into the requirements document and interviewing the stake holder, we discover that the ink has several properties: the color, how fast it dries, the consistency of the ink when writing, how much ink is in the marker, and whether the ink smells foul. We then augment this leaf of the graph:



From this point, we still need more detail. What are the aspects of the amount of color of the ink? There may be several more sub-categories of this. After several more iterations this graph might be three or four levels deep with several dozen leaf nodes. Let's say one of the leaf nodes is "the length of a line you can draw with a fresh marker before the ink runs out." From here we will need to build a test case.

A test case must have three properties: valid, reliable, and efficient. **Valid** means that the data resulting from the test case pertains to the actual property it purports to be measuring. **Reliable** means that multiple executions of the test case will result similar data. **Efficient** means that it will not take too much time or resources to gather the data. So what experiments can I conduct that will be valid, reliable, and efficient?

First attempt: Purchase a very long white board and draw a single line. This test will be valid (it does in fact measure how long a line can be), it will have reliability problems (how do I know the exact point when the ink ran out?), and not very efficient (there will be a great deal of walking involved and will require purchasing a very long whiteboard).

Second attempt: Purchase one large white board and color in the entire board. This will be somewhat valid (coloring an area is not exactly the same thing as drawing a line), relatively reliable (it should be somewhat clear when the marker ran out of ink), and more efficient than the first test (no walking is required and the white board is much easier to procure, but it will still take a lot of effort).

Third attempt: Purchase a coffee can and attach it to a motor spinning at roughly the speed the average person writes on a white board. Cover the outside of the coffee can with white board material so it is seamless. On one side of the can, place a marker holder. On the other side, put an eraser. Station a digital camera between the marker and the eraser that measures the quality of the line. Conduct the test by spinning the drum and putting the marker in contact with the surface. Record the amount of revolutions when the first break in the line occurred and when the line is only 50% covered. This experiment is valid (it measures the length of the line), reliable (the two measurements paint an accurate and reproducible measure of the length of the line), and efficient (after the initial cost of the experiment is created, the cost of re-testing a given marker is extremely low).

The Process

The above example illustrates the three steps in the process of creating a test plan: creating the test graph (often called the test matrix), identifying the test cases, and creating a test procedure.

Test Graph: The process of creating a test graph starts with the requirements and drill down to smaller and more specific components. With each iteration, ask yourself two questions: "do these sub-components completely cover all aspects of the component" and "is this component specific enough to create a test case?" It is useful to go back to the requirements document periodically to make sure that all aspects of the design are checked off.

It is often difficult to know how to start a test graph and to know when it is complete. Often a good place to start is your quality model. Recall from earlier in the semester the various quality models presented by McCall [6], Boehm [7], Alvaro [8], and others. These models often provide not only the first-level leafs of the test graph (the criteria of software quality), but also often present a hierarchy of how they are related. In other words, there is a relationship between your quality model and your test graph.

Test Cases: When the graph is complete, all the leaf nodes represent test cases. For each test case, create at least one experiment that has the properties of validity, reliability, and efficiency [6]. It might take a few tries to come up with the best experiment. Note that if only two of the properties are met (say valid and efficient but not reliable), then the test case is not yet ready and needs work.

Each test case should be easily traced to an aspect of the software quality model. When the test case is attached to the test graph, this relationship should be fairly obvious. The test case usually has a title, a test procedure, and another other notes that would help the tester carry out the test or understand the results.

Test Procedure: When a given test case is fully identified, it is necessary to create a **test procedure**:

> Detailed instructions for set-up, execution, and evaluation of results for a given test case [4].

Observe that there are three parts to a procedure: the set-up, execution, and evaluation:

1. **Set-up:** It is necessary to create unambiguous instructions how to perform the test case. What material or software is needed? How do you configure the testing software? What state must the system be in before the test is conducted? This is all part of the set-up.
2. **Execution**: A step-by-step specification of how to conduct or execute the test must be given. If the test is conducted manually, it needs to include a set of steps in enough detail that any competent tester can perform it the same as the test case author. If the test is conducted automatically, then the software needs to be well-commented and easy to use.
3. **Evaluation**: It needs to be unambiguous how to interpret the results. How will the tester implementing the test case know if the test succeeded or failed? If the test is conducted manually, then it should be clear what success looks like and what failure looks like. If the test is conducted with automation, then there needs to be instructions how to interpret the resulting test logs.

Works Cited

[1] Navy Mathematical Computing Advisory Panel, "Symposium on Advanced Programming Methods for Digital Computers," Office of Naval Research, Department of the Navy, Washington, D.C., 1956.

[2] T. Bell and T. Thayer, "Software Requirements: Are They Really a Problem?," in *Proceedings of the 2nd international conference on Software engineering*, 1976.

[3] IEEE, "IEEE Standard for Software Verification and Validation Phases," 1986. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=2487. [Accessed 24 April 2015].

[4] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," December 1990. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=159342. [Accessed 23 April 2015].

[5] Y. Chernak, "Validating and Improving Test-Case Effectiveness," *IEEE Software,* vol. 18, no. 1, pp. 81-86, 2001.

[6] J. Cavano and J. McCall, "A Framework for the Measurement of Software Quality," in *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues*, 1978.

[7] B. Boehm, H. Brown and M. Lipow, "Quantitative Evaluation of Software Quality," *Proceedings of the 2nd international conference on Software engineering,* pp. 592-605, October 1978.

[8] A. Alvaro, E. Almeida and S. Meira, "Towards a software component quality model," *5th International Conference on Quality Software,* 2005.