UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

University of Limerick

CS4157 Software Quality

The Role & Importance of Testing in Facilitating the Development of Quality Software

Niall Dillane

CSIS

Submitted to

Dr. Valentine Casey
CSIS

1 May 2020

Abstract

Testing is an often neglected stage of the software development process, garnering less praise and compensation for its workers. Throughout this paper, we will research and demonstrate the various aspects of testing, as well as its importance in developing high quality systems.

Contents

1 Introduction

Software Testing is the investigation of software for the purpose of finding defects and providing feedback on the quality of the product (Kaner 2006). This typically involves running the program — either manually or automatically with some kind of script — and providing input with the intent of finding defects (colloquially known as bugs), or rather proving that there are no bugs. Of course, it is impossible to prove that a piece of software is entirely bug free (Pan 1999), at least if it has any degree of complexity, but testing aims to provide a reasonably complete picture of the state of the software and an avenue to reducing defects.

Testing has become more prominent in recent years (Tuteja et al. 2012), but I believe it still does not get the credit it deserves. Testing is a very broad area, which is key at each stage of the software development process: from specification, to implementation, to maintenance. As it consumes 40-50% of development efforts (Tuteja et al. 2012), it warrants significant consideration and planning from the beginning, so as to minimise costs and time needed later on.

2 Dynamic Analysis in Verification and Validation

Research and discuss the role and importance of dynamic analysis in the verification and validation of software and the essential part it plays in facilitating quality.

There are two different types of analysis in software testing: those are static and dynamic analysis.

Static analysis simply involves analysing a piece of software without actually running it (Wichmann et al. 1995). This can entail human review of code in a text editor, tools used to provide statistics e.g. lines of code per file, or the use of a compiler. This can be useful in certain circumstances, such as finding security vulnerabilities in a codebase (Livshits 2006) and is naturally quicker than walking through a program, but generally it is limited from a "user experience" point of view.

Dynamic analysis is based on system execution, often using tools to automate the process (Ghahrai 2018). There are various stages, from individual unit tests of small portions of the system, to integrating these units and the system overall. Dynamic testing provides a more complete picture of the state of the system, and is a more "real life use" approach. This can be time consuming, and so more and more automation is sought out, but one must be wary of the false sense of security these tools can provide.

Verification and Validation are related concepts in software testing. In whole, they describe the process of: identifying if the system has satisfied the specification provided (verification); and determining if that specification is what the customer really wants or needs (validation). In short, verification answers the question: "Are we building the system right?", and validation answers "Are we building the right system?" (Boehm 1984).

Dynamic verification involves, as outlined previously, executing the code. At this point, developers wish to identify if the system has been built to specification, conducting a review of the program in various ways. These can be testing in the small — checking if inidividual functions perform as expected — up to testing the system as a whole, by executing modules and checking that they interact as expected. Non-functional requirements (discussed more in Section 3) may also be tested, such as stress-testing a server to see how many users it can handle.

Dynamic validation is an even higher level of testing (acceptance testing), wherein the development team and external stakeholders walk through the product and use it as a user would. This is black box testing (see Section 6), as the testers are performing these actions without looking at the internal workings of the system. A purely surface view.

In both verification and validation, dynamic analysis is very important, as it provides real-world feedback on the system and any defects found. Runtime execution is very different from compilation, so I believe static analysis alone is not enough in verifying that the system is built correctly. This is even more critical in the case of validation, since it is recommended to have external stakeholders take part in the testing. These may be non-technical people, so simply showing a series of code metrics will likely be insufficient. Much more preferable is having a dynamic experience, acting out pre-defined user stories and ensuring that the customer is satisfied with the experience.

Some have argued in favour of pure static analysis, pointing out that dynamic analysis is unreliable and cannot ensure complete coverage (Zhioua et al. 2014). However, even here the arguments are generally limited to certain aspects of the software, such as security.

3 Functional and Non-Functional Requirements

Clearly outline and define what Functional and Non-Functional Software Requirements are and in your discussion compare and contrast the differences between them

Functional requirements define behaviour that a system must include, some kind of function that takes input and/or provides output (Stellman and Greene 2005). These are generally found in use cases, e.g. "a user logs in to their profile". These are definite pieces of functionality that the system will fulfill, which may perform calculations, communicate with outher services or utilise data.

Non-functional requirements (NFRs) place constraints on the development of the system and its functionality, and may include: performance, reliability, security, etc, (Stellman and Greene 2005). These are not exactly behaviours that the system must allow, but rather quality attributes which define how the system exhibits these behaviours, e.g. "90% of pages must load within 1 second".

3.1 How to Test

Research and discuss how Functional Software Requirements and Non- Functional requirements can be successfully tested and define the differences in approach required by each

Functional testing is a black-box (Section 6) approach to testing, wherein functions are passed input and have their output verified for correctness (Kaner et al. 1999). This is not the same as input testing, where we test a single method of the program, but rather a function of the system which could interact with many different parts. For example, in the case of testing "login" functionality, the input data prepared may be a username and password, while the output expected may be a success message and redirection. These are prepared, the test carried out and then verified, either manually or with an automated tool.

Non-functional testing is manifold, with different techniques required for performance, security, portability, etc. Each of these aspects will have their own methods, and are selected according to the NFRs of the specific system. Within performance, a common NFR is load testing: determining a system's performance and behaviour under normal circumstances as well as peak load (Eriksson 2019). Generally, there are tools available for testing these different aspects, but other more abstract NFRs, like usability, may require manual testing and surveying.

4 Software Testing Process

Outline and discuss the key stages in the software testing process

The software testing process can vary depending on the organisation and the software development lifecycle they employ. The latest developments in this space are Agile[1] and Extreme Programming[2], which often utilise Test Driven Development in writing tests first, then the code required to pass them (Alliance 2005). These tests are added to as the system develops, as well as additional code to satify them, hence the term "continuous integration".

In more traditional software development models, such as the Waterfall model[3] testing is carried out at various stages:

- Requirements are analysed and testable aspects are identified

[1]`https://agilemanifesto.org/`
[2]`http://www.extremeprogramming.org/`
[3]`https://en.wikipedia.org/wiki/Waterfall_model`

- Tests are plannned and scripts developed if necessary
- Tests are executed and results logged. Errors are reported to the development team
- Metrics and reports are generated on the state of the software, whether it is ready for release etc.
- Regression testing is carried out to check if defects have been fixed, or if new features pass the tests

(Pan 1999)

Note also that there are different levels of testing, as touched on previously.

4.1 Unit Testing

4.2 Integration Testing

4.3 Regression Testing

4.4 System Testing

4.5 Acceptance Testing

5 Effective Test Strategy

Research and define the principles of an effective test strategy.

5.1 Documented Test Procedures

Consider and discuss the role and importance of having documented test procedures in place to facilitate effective testing

5.2 Test Plan

Research and outline the purpose, importance and content of a test plan

6 White Box and Black Box Testing

Define and discuss the importance of utilizing both white box and black box testing

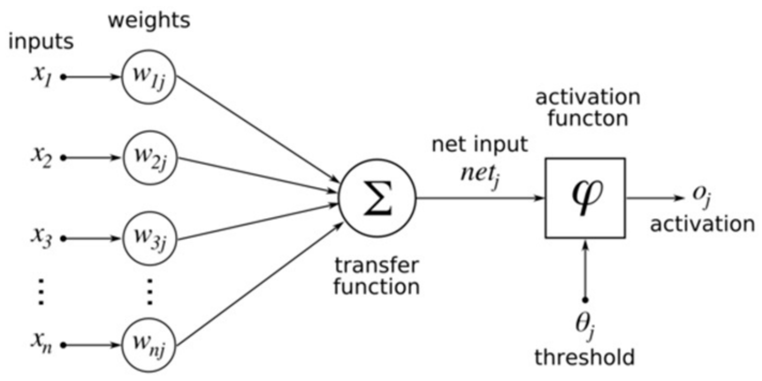
7 Sample Table

Table 1: Comparison p values

Attribute	***p-value***	**Significant**
Model A	0.0521	N
Model B	0.6171	N
Model C	<0.00001	Y

7.0.1 Sample Figure

Figure 1: Perceptron (Artificial Neural Network)



References

Alliance, A. (2005), ‘Tdd’, *agilealliance. org [Online]. Available: http://guide. agilealliance. org/guide/tdd. html.[Accessed: Apr. 27, 2015]* .

Boehm, B. W. (1984), ‘Verifying and validating software requirements and design specifications’, *IEEE software* (1), 75–88.

Eriksson, U. (2019), ‘All about load testing, understand the load tests’.
URL: *https://reqtest.com/testing-blog/all-about-load-testing/*

Ghahrai, A. (2018), ‘Static analysis vs dynamic analysis in software testing’, *Testing Excellence* .

Kaner, C. (2006), Exploratory testing, *in* ‘Quality assurance institute worldwide annual software testing conference’.

Kaner, C., Falk, J. and Nguyen, H. Q. (1999), *Testing computer software*, John Wiley & Sons.

Livshits, B. (2006), *Improving software security with precise static and runtime analysis*, Vol. 67.

Pan, J. (1999), ‘Software testing’, *Dependable Embedded Systems* **5**, 2006.

Stellman, A. and Greene, J. (2005), *Applied software project management*, " O’Reilly Media, Inc.".

Tuteja, M., Dubey, G. et al. (2012), ‘A research study on importance of testing and quality assurance in software development life cycle (sdlc) models’, *International Journal of Soft Computing and Engineering (IJSCE)* **2**(3), 251–257.

Wichmann, B., Canning, A., Clutterbuck, D., Winsborrow, L., Ward, N. and Marsh, D. (1995), ‘Industrial perspective on static analysis’, *Software Engineering Journal* **10**(2), 69–75.

Zhioua, Z., Short, S. and Roudier, Y. (2014), ‘Towards the verification and validation of software security properties using static code analysis’, *International Journal of Computer Science: Theory and Application* **2**.