COMP 4110 Software Testing and Verification Group Project

Step 4: Unit Testing

Hasan Malik 104610462
Bilal Malik 104435995
Clyde Rempillo
Mohammad Farhat

Abstract

In this report, we have decided to utilize unit testing as our main topic, as this plays towards our strengths and knowledge as a group. We will be able to precisely walk different avenues of the source code and have successful test cases for a full safe and complete software environment. With the use of the 10 articles, we are going to provide a detailed summary and analysis with its relation to the materials discussed in class. We begin with discussing how software plays a crucial role in software quality assurance even though it is a time and resource consuming process. Moreover, we continue our discussion with unit testing in global software development environment, such as the use of Parameterized Unit Testing (PUT). This testing method can be used to enable economical use of resources by reducing manual effort and assist in generating specific inputs to optimize and efficiently create results. Finally, important practices in unit testing and the related concept of test-driven development are discussed in this report in regards to the important applications and implementations in the software testing industry.

Introduction

We begin by discussing the basics of unit testing as discussed in class and then we move onto the implementations as it relates to the articles we provided. In Chapters 1 and 2, we mainly focused on the psychological and economic part of software testing. It has proven to be impossible to test every permutation of a program, hence, creating test cases for all possibilities would not be ideal. Moreover, the articles we have picked as a group provide various types of implementations economically. From Chapters 3 through 4, program inspection methods such as walkthroughs and test-case designs were discussed as it is a critical part of software testing. Finally, in Chapter 5, Module/Unit Testing was discussed in class, to provide a deeper understanding and knowledge of this important practice when it comes to software testing. We will provide summaries of our 10 articles and their relation to the materials discussed in class.

We look at many different articles using unit testing. From attempting utilize unit testing on SQL through building a new language, to utilizing it in the code learning environment. Unit testing is used everywhere, hopping on hackerrank and creating programs to resolve many difficult problems in a safe vanilla environment where it can be tested properly is all made the smoother through unit testing frameworks. He we can see unit testing being used on the front lines with UI, instead of its usual environment working behind the scenes with modules. When coding in the eclipse ide, it fetches its data from the Java Enterprise Edition api through control flows built on unit testing. Whether it's for academia, pushing the knowledge and cusps of computer science, or for application in the industry, unit testing plays a big role. We will walk through many different avenues in computer science where unit testing is applied

Reading List:

1. Advances in Unit Testing: Theory and Practice

The Java Enterprise Edition and its powerful API are used to create and enhance applications that interact with Database Systems. Control flows of these applications firmly rely upon a particular database state. An automatic test-case approach is displayed for applications in the current environment. The methodology produces both test information for the application's contribution, just as entity objects for the distinctive database states that are needed to cover a separate control flow. The task is to integrate constraints and limitations from symbolically executing a control stream with constraints on a necessary database state. The authors also supported Java EE functionalities in the symbolic execution, reliance infusion is an example of this. An exploratory assessment shows an expansion in control-flow coverage by the authors methodology.

Java EE is utilized in many different places in the computer world including the eclipse integrated development environment. Eclipse can access these api's offered by JavaEE at a smooth and accessible rate. This access occurs through unit testing to ensure that control flow communication with these api is working at its finest. We can see the direct application of this paper and unit testing in our own development environment. This is an example of one of the many different uses of unit testing.

[1] A. Fuchs and H. Kuchen, "Unit Testing of Database-Driven Java Enterprise Edition Applications," *SpringerLink*, 19-Jul-2017. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-61467-0_4. [Accessed: 04-Feb-2020].

2. CUT: automatic unit testing in the cloud

Unit tests can take major strides in performance by being run at the same time across distributed execution environments. The largest form of this practice can be witnessed in cloud testing. Since designing and building these environments requires a great amount of effort, time and expertise that developers may not have the skill or resources for, Cloud Unit Testing (CUT) was introduced. CUT is a tool that allows you to automatically execute unit tests in distributed execution environments. CUT will handle everything by allocating computation resources and granting access to virtual machines. Developers won't be required to change any unit test code that has already been written, and will be able to control all aspects of test execution. In addition to all this CUT will also provide resources that will generate analytics and give you insight into the unit tests that have been executed.

The development of CUT can greatly increase the speed and efficiency of regular unit testing. It also will create greater testing opportunities for projects that may not have had the resources to effectively perform tests and maintain a desired quality of code. The use of virtual machines will also open new doors with stronger machines that are capable of performing complex tests. The ability to have analytics also creates new avenues for testers to better refine and understand their testing skills. CUT is an automated cloud unit testing experience that can change how unit testing is practiced today. I believe CUT will play a major role in the future of unit testing.

[2] A. Gambi, S. Kappler, J. Lampel, and A. Zeller, "CUT: automatic unit testing in the cloud," *CUT: automatic unit testing in the cloud | Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 01-Jul-2017. [Online]. Available: https://dl.acm.org/doi/10.1145/3092703.3098222. [Accessed: 03-Feb-2020].

3. Unit testing performance with Stochastic Performance Logic

Unit testing works as a quality management tool in software development. Unfortunately, practical constraints make unit testing difficult to be utilized for performance testing. Therefore, Stochastic Performance logic was developed, "a formalism for expressing performance requirements, together with interpretations that facilitate performance evaluation in the unit test context." The small scope of unit testing makes it flexible, where little installation is necessary. This makes unit testing an attractive take for performance testing. There are various hurdles that need to be jumped through, including test implementation, and design which can lead to many misleading results. A framework is designed to overcome these hurdles as well as succeed as a performance unit test framework. Central to the framework is Stochastic Performance Logic. This mathematical language is used to express requirements to better work alongside the unit test parameters.

This SPL based performance testing has been integrated into an automated testing framework for Java, akin to JUnit. An example of the successful result can be seen by the 58 tests in which 6 cases were utilized to catch many bugs in the performance, bugs not caught through the process testing. This new testing software has seen success and revolutionary work though its research to design a formalized math language to help create the process unit test frameworks. I believe this, of the test cases I've worked on, has the potential to make for many greater findings. The utilization of math and science to build this framework not only helps people in the industry, but I believe people in academia can look at this tool for further understanding and greater knowledge.

[3] Bulej, L., Bureš, T., Horký, V. *et al.* Unit testing performance with Stochastic Performance Logic. *Autom Softw Eng* 24, 139–187 (2017). https://link.springer.com/article/10.1007/s10515-015-0188-0

4. A Constraint-Based Framework for Test Case Generation in Method-Level Black-Box Unit Testing

In this article, the authors outlined the importance of unit testing in software engineering by introducing a new package aimed to make it easier for developers to add units to test their own package. The authors did this by drawing from examples in the ".rd" documentation. Importantly, the authors outlined unit tests using the test that package, a standard unit testing package. These features make examplestr consistent with best practices for R package development.

The authors used examplestr::make_tests_shells_pkg() on an old deprecated package that they were renovating. They encountered an error at first, however, the package provided precise instructions on how to resolve the error. Otherwise, the examplestr package works exactly as advertised. In my personal opinion, examplestr would have worked just as well in combination with the Rd2roxygen package to help bring antiquated R packages up to a more rigorous and maintainable software standard.

[4] C.K. Chang, N.W. Lin. "A Constraint-Based Framework for Test Case Generation in Method-Level Black-Box Unit Testing". *Journal of Information Science and Engineering.* 2016. [Online]. Available:
https://www.iis.sinica.edu.tw/page/jise/2016/201603_07.pdf . [Accessed: 04-Feb-2020].

5. Unit Testing, Model Validation, Biological Simulation

In this article, the discussion was about experience in applying a number of basic practices of industrial software engineering, which is also more broadly known as *unit testing* and the related concept of *test-driven development* in the context of the OpenWorm project. The authors showed how these industrial practices can be used and extended for computational modelling in biological sciences. The manuscript was well-written and provided a great way to understand the basic concepts.

I personally do believe that establishing a specific culture of test-driven development in biological sciences is of great importance, however, I also found that the application for software engineering practices are not as easy as how the authors portrayed in their article. Therefore, I have provided my personal critiques and suggestions upon improving their manuscript:

a. The judgement of the quality and validity of a computational model is a matter of scientific discussion and often cannot be easily reduced to a pass or fail in a validation test. I believe that there should be more detail on the transformation of the numeric score to a Boolean value used in the given test sample, but also on the statistical concepts behind the decisions.

b. I believe that to iteratively improve a computational model, it is crucial to know not only "if", thus, also why a certain model passes or fails the model validation test. A discussion on the limits of the current continuous integration tools for biological modelling would be very helpful.

[5] G.P Sarma, T.W Jacobs, M.D. Watts, S.V. Ghayoomie, S.D. Larson, R.C. Gerkin. "Unit testing, model validation, and biological simulation", *F1000Research,* 2016. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5007758/ [Accessed: 04-Feb-2020].

6. Unit Testing in Global Software Development Environment

Global Software Development is an increasing trend in the software world. One of the many challenges Global Software Development faces is maintaining the quality in the developed code in distributed sites. This study proposes an effective way of controlling this challenge which is through module/unit testing in object oriented systems. It has become a common trend that developers often look passed unit testing however this must change when dealing with Global Software Development as managers tend to lose control on the quality aspect of the development. This trend exists because developers have a difficult time finding the correct metrics [6]. The metric under consideration is the Weighted class complexity is developed on the form of expression related between the data and function and class can be assumed as a set of data and set of method accessing them [6]. As a result of the Weighted class complex formula, there is a substantial amount of data which displays the effectiveness of the test in terms of an object oriented Class.

[6] Misra S., Adewumi A., Maskeliūnas R., Damaševičius R., Cafer F. (2018) Unit Testing in Global Software Development Environment. In: Panda B., Sharma S., Roy N. (eds) Data Science and Analytics. REDSET 2017. Communications in Computer and Information Science, vol 799. Springer, Singapore. Available: https://doi.org/10.1007/978-981-10-8527-7_25

7. Predicting Unit Testing Effort Levels of Classes: An Exploratory Study based on Multinomial Logistic Regression Modelling

In this article, the authors discussed how software testing plays a crucial role in software quality assurance even though it is a time and resource consuming process. They included several different factors that affect the overall effort spent on testing, such as human factors, process issues, testing techniques, tools used, and more. Software testability is an important software quality attribute, and is in fact a complex notion that is affected by many different factors. The authors indicated that their focus was on unit testability of classes and particularly on the effort involved to write the code of unit test cases. They also included a brief presentation of the Quality Assurance Indicator (Qi) metric that is based on the concept of Control Call Graphs (CCG). They were able to successfully demonstrate the CCG of a method can be seen as a set of paths that the control flow can pass through.

The authors provided well-documented and reasonable demonstration, however, one of the concerns is why some of their results are significantly discerned by the metrics Qi and CBO only in the case of ANT. Also, I believe that this paper should be replicated in the future using several other case studies in order to draw more general conclusions and validation. In order to improve this paper, I have the following suggestions for the authors to possibly implement if possible:

a. They should investigate deeper on the performance of the prediction models based on the distribution of the unit testing effort.

b. They should also extend the study by exploring other classification techniques to better demonstrate the unit testing effort.

c. I also believe that they should add other OO metrics, and explore the use of the Qi metric during the testing process to better guide the iterative distribution of the testing effort.

[7] M. Badri, F. Toure, L. Lamontagne. "Predicting Unit Testing Effort Levels of Classes: An Exploratory Study based on Multinomial Logistic Regression Modelling", *Procedia Computer Science*, 2015. [Online].
Available: https://www.sciencedirect.com/science/article/pii/S1877050915026630 [Accessed: 04-Feb-2020].

8. Relational Symbolic Execution of SQL code for Unit Testing of Database Programs

Symbolic Execution executes the unit test over symbolic input values instead of concrete ones. It is utilized as a white box testing method, to ensure a large portion of the program is executed. In simple terms, Symbolic execution works to make test cases that can encompass large amounts of the programs, specifically if there are logical branches that may not be utilized in a single run. This utilized under the weight of unit testing, means to apply runs on different cases on specific modules to span a large amount of that module. As well, classical code mixed with SQL, like PHP, has always found a need for testing to prevent data corruption. Thus, a marriage between Symbolic Execution of unit testing in the presence of SQL statement seems like a natural mix. Unfortunately, the reality of coding constraints makes this task very complex. Deciding the satisfiability of an SQL query is not computationally possible, why use this specific query instead of another if they both output the same result. Transferring these test cases into the SQL becomes a difficult endeavour. "SQL is a declarative language…They do not make explicit the sequence of operations necessary to compute this action." To overcome this obstacle many researchers have introduced native variables in the classical code to represent the database content and replace the SQL queries and modifications by native code acting alongside the new variables. Let's say we have java for example, instead of including the embedded SQL commands, they will be Java. This way, the auto generation of test scripts alongside SQL can be of much ease while still ensuring the test walks through all key paths and reduces the chance of bugs significantly in the module.

This task is a very interesting problem. On one hand, you have a very formidable software in the symbolic execution. This meeting the SQL head on, a language that doesn't have a proper way to follow its structure and spit out test cases. The work around, is to create new variables within the original code that work as directives, building and querying a database without leaving the comfortable confinements of the native language. This way the symbolic execution can continue to create test cases, by looking at the different paths inside of the original source code. It can continue to build solid test scripts to ensure these modules are tested properly and ready for integrating with the rest of the code. This form of unit testing is implemented through Whitebox methodologies and can work well with backend frameworks in web and app development, that utilize databases to a heavy extent.

[8] M. Marcozzi, W. Vanhoof, J. Hainaut. (2015, July). Relational symbolic execution of SQL code for unit testing of database programs. ScienceDirect. [Online]. 105. Pp. 44-72. Available: https://www.sciencedirect.com/science/article/pii/S0167642315000660

9. Pythia reloaded: an intelligent unit testing-based code grader for education

The paper proposes a unique coupling between unit testing frameworks and automated competition software graders for automatic assessment of educational code. Seeing as assessment of code in educational programs, normally is provided in small predefined modules with set parameters. It creates an environment where unit testing can succeed. Currently, two major kinds of code assessment tools exist: unit testing frameworks and competition graders. Though each have their own issues working alone with automated assessment tools, Unit testing just outputs the test cases that failed and the ones that passed. This does not work well with learners, who need a more in-depth explanation to their results. The issue with competition graders is "that they are very specific to satisfy the constraints they have to meet. Therefore, they often only support a single programming language." Although both these tools have their perks, they both carry key issues as standalone automatic assessments. Therefore, Pythia was designed. "The platform combines a unit testing framework with a competition grader." This way the competition graders can be utilized for the isolated sandbox for the safe execution of code. All the while, the unit testing framework brings a structured way to test the learning code. To add, the platform adds a feedback system that can give accurate responses based on why the test case failed. A key fault with unit testing in the learning environment is resolved here.

This opens a new way of looking at testing code. As developers, we work with many unit testing frameworks. Now this testing framework needs to work at a higher level. Directly in the hands of the clients, this testing tool needs to work not too help make a working software, but as the working software the consumer utilizes. The way Pythia helps to learning environments, by creating an intelligent feedback system coupled with the two frameworks shows the work that goes into creating a strong learning environment.

[9] S. Combefis, A. Paques. (2015, July). Pythia reloaded: an intelligent unit testing-based code grader for education. ACM Digital Library. [Online] Available: https://dl.acm.org/doi/pdf/10.1145/2792404.2792407?download=true

10. Advances in Unit Testing: Theory and Practice

Parameterized Unit Testing (PUT) expands on previous industry practises which were based on traditional unit testing without parameters. A PUT takes parameters, calls the code to be tested then states the results of the test. PUT also allows for the separation of Black Box testing and White Box testing. PUTs are also supported by a wide variety of testing frameworks and there are now tools being developed specifically for generating test inputs. This testing method can be used to enable economical use of resources by reducing manual effort. [10]

Allowing the use of parameters with traditional closed unit tests will assist in generating specific inputs to optimize and efficiently create results. Creating specifics may also prevent the detection of unprecedented issues which the parameters may not accomodate. Even with the potential flaws PUT may cause the economic savings and reduced effort make it a viable solution. The wide variety of tools being developed specifically to support PUT will also give this practice longevity in the marketplace.

[10] T. Xie, N. Tillmann, and P. Lakshman, "Advances in Unit Testing: Theory and Practice," *Microsoft Research*,01-May-2016. [Online]. Available: https://www.microsoft.com/en-us/research/publication/advances-in-unit-testing-theory-and-practice/. [Accessed: 04-Feb-2020].