Week08 – Short Paper Assignment – Test Driven Development

SWEN 603 9041

Debashis Jena

University of Maryland Global Campus

# Introduction

As soon as the analysis and design is complete, usually the developers start creating their modules and components looking at the requirements. As the development is complete, typically the definition of “done” has an additional task to create unit tests. Therefore the unit tests become part of the development tasks. In some platforms they have a requirement of certain percentage of the code base to be covered by the tests. That way most of the developers gain a lot of confidence in their code. The test-driven development (TDD) process is a software development method that is driven by the tests. The unit tests are created before the working code is written and let the unit tests fail first. As the code is written and takes form, the tests start passing. As the new tests are written, they are also expected to fail first instead of a false positive.

A close up of a logo

Description automatically generatedTest driven development is a practice followed in agile software development. As the agile method has ever-changing requirements, TDD is the software development practice that can help developers produce working software. It will keep the process simple and provide them quick feedback in case something goes wrong. This process forces the team to work in short incremental development cycles, and provides quick feedback in the development progress. The complete TDD process looks as follows:

1. Write tests for the new features.
2. Let the test fail.
3. Write enough code to make the test pass.
4. Make sure all the previous tests pass as well.
5. Refactor the code.

# Level of Testing

A picture containing drawing

Description automatically generatedAs the development phases go, the level testing varies from one phase to the other. So it is very important to write proper tests for each of these levels. As the development begins the tests are written before the start of the phases, executed continuously and refactored as required. The figure displays how the testing levels differ by the time, effort and cost.

1. Unit tests – The architecture document shows how the components are distributed and decoupled from each other. These components are also called as units. Typically these are written by the developers. Writing unit tests follow the respective framework for the programming language that is used. For example, in Java the Junit and TestNg frameworks are available to write and run the unit tests. For the dependent piece of code, the developers write stubs and drivers to replace the connected components. This testing is also called white-box testing or component testing, where the developers have sufficient knowledge about the entire code base and the architecture.
2. Integration tests – As the components are built they are integrated with each other or with the external applications via an API. Typically, external API functionalities are not tested, since they are assumed to have been tested by another team or the third party vendor. In this testing phase, the integration points and data transaction between the connected applications are tested, not the applications itself. There are integration tests written to validate the input and output of the connections. There are many integration test tools available. For API testing, REST Api testing clients (like Postman) can be used. Some of the frameworks also support “mocking”. Mocking is defined as mimicking the API response, without making the real API call to the external system. In Java, the Mockito framework library can be used to mock the external API responses. There are many such frameworks available that can be utilized to validate the integration between systems.
3. Acceptance test – This is the full system tests performed using automation testing tools like selenium or QTP. Typically these tests are written only when the application GUI is ready to be accessed. They are meant to validate the system functionality from end to end. These tests may follow a keyword driven or data-driven framework. The framework setup may be time consuming process initially.

**Conclusion**

TDD process has many benefits and some shortcomings too. Here are a few of each.

*Benefits*:

* Confidence in code
* Less testing required upon any major change in the other parts
* Helps setting a code quality caliberation
* Early defect discovery
* Can be used for go / no-go decision

*Shortcomings*:

* Not same as manual ene-to-end testing
* Lacks a different perspective
* Moderate chance of having false positive based on the unit tests

**Reference**

Gulati, Shekhar & Sharma, Rahul. ( © 2017). *Java unit testing with junit 5: test driven development with junit 5*. [Books24x7 version] Available from *http://library.books24x7.com.ezproxy.umuc.edu/toc.aspx?bookid=135434*.

Row, Jason. ( © 2015). *Software testing fundamentals: test-driven development*. Available from *http://library.books24x7.com.ezproxy.umuc.edu/toc.aspx?bookid=71789*.