TEST-DRIVEN DEVELOPMENT RECOGNITION AND EVALUATION

A THESIS PROPOSAL SUBMITTED TO MY THESIS COMMITTEE

DOCTOR OF PHILOSOPHY

IN

INFORMATION AND COMPUTER SCIENCES

By
Hongbing Kou

Thesis Committee:

Philip M. Johnson, Chairperson
...

September 21, 2004
Version 1.1.0

Abstract

“Test-Driven Development (TDD), also called Test-First Design (TFD), is a software development practice in which test cases are incremently written prior to code implementation[2]”. The rational of TDD is to “Analyze a little, test a little, code a little and test a little, repeat.” This work is to recognize TDD process constructed by many Red/Green/Refactoring iterations and evaluate TDD by analyzing the development activity stream. Upon successful this research will provide the inspection support and help TDD practitioners continuously improve the development process.

Chapter 1

Related Work

"Test-Driven Development (TDD) is a software development practice in which test cases are incrementally written prior to code implementation."[2] Here test case is the unit test, which is a piece of code written by a developer that exercises a very small, specific area functionality of the code being tested. The rational of TDD

is to "Analyze a little, test a little, code a little, and test a little,repeat." The goal of TDD is to write "clean code that works"[1] and it has two basic rules. [1]:

1. Write new code only if an automated test has failed.
2. Eliminate duplication (Refactoring).

They imply an order to do programming task [1]:

1. *Red*
   Write a little test that does not work, and perhaps does not even compile at first.
2. *Green*
   Make the test work quickly, committing whatever sins(for example, constant, fake implementation) necessary in the process.
3. *Refactor*
   Eliminate all the duplications created in merely getting test work.

In book "Test-Driven Development by Example", Ken Beck claimed that red/green/refactor rhythm is the mantra of TDD, once you have an automated suite of tests you will never go back. It gives incredible confidence in your code. He also said TDD will naturally generate 100% code coverage and a coverage tool is not necessary at all if you do TDD perfectly. Unit tests are very important to software development because it makes it easy to check whether the system functions properly or not. It allows the programmer to refactor code at a later date, and be sure the model is still functioning properly[6].Also the automatic unit test suite created in the development process could be used as regression tests. And “a unit test behaves as executable documentation, showing how you expect code to behave under the variation condition you’ve considered.”[5]

In common software development unit test is not a must and is usually done as an afterthought. Occasionally no tests are created at all, especially with tight schedules. With TDD, before writing implementation,the automated unit tests are written first and they are created on the ground of requirement analysis, which leads to better and effective test suites.

However, in reality, software developers are educated to program first and walk though the program to make sure the system works well to the desired scenarios. As a new and counter-intuitive approach TDD requires discipline, training and good tool support. Kent Beck suggested the “xUnit” framework. “xUnit” has many variances and it has been ported to more than 30 languages’ support[7]. xUnit is a very light-weight tool and it can be mastered by a novice programmer in several minutes. But to applications like graphic user interface, socket programming, parallel computing, database querying etc. It is hard to write small unit test to deploy them or not practical. TDD practitioners suggest mocking technique to mock up the real operations for test purpose. There are two widely accepted mock tools called MockObject and EasyMock. Mock technique is still under development and is welcomed by many practitioners in daily programming work.

Boby George’s analysis and quantification of test-driven development concluded that both students and professional TDD developers appear to have higher code quality[4]. E. Michael Maximilien and Laurie A. Williams found that defect rate of project IBM Retail Store Solution was reduced by 50% compared to another system built with ad-hoc unit test approach [3], and TDD developers passed 18% more functional black-box tests than non-TDD developers.

Chapter 2

Hypothesis, Claim and Experimental Validation

Bibliography

[1] Kent Beck. *Test-Driven Development by Example*. Addison Wesley, Massachusetts, 2003.

[2] Laurie Williams Boby George. An Initial Design of Test-Driven Development in Industry. *ACM Sympoium on Applied Computing*, 3(1):23, 2003.

[3] Laurie Williams E. Michael Maximilien. Accessing Test-Driven Development at IBM, 2003.

[4] Boby George. Master's thesis.

[5] Andy Hunt and Dave Thomas. Pragmatic Unit Testing in Java with JUnit.

[6] Unit Test. `<http://en.wikipedia.org/wiki/Unit_test/>`.

[7] XP Software download. `<http://wwww.xprogramming.com/software.htm/>`.