An Exploration into the Current State of Test-First vs. Test-Last Testing

Christopher Morris Thomas

Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

2013 Fall Senior Seminar

Outline

Outline

Software Testing Defined

Software testing is a branch of software engineering that uses various practices to:

- identify potential malfunctions in code
- demonstrate functionality of software

Testing can be done manually or done using test code.

Why Do We Care About Software Testing?

In 2007, a study found that businesses spent at least 50% of their production costs on product verification and testing.

Because of this, software businesses are constantly looking to optimize their testing practices in an attempt to reduce costs.

Test-Last Testing

Test-last testing is:

- writing tests after the code has been written
- used in waterfall or any plan based development
- considered the “conventional testing method"

Test-First Testing

Test-first testing is:

- writing repeatable tests before the code is written
- often used to define code functionality
- Often used in agile programming practices
- usually used as part of a development model

Test Driven Development

Test driven development (TDD) is the most well known and used test-first development model.

The original TDD method consists of three steps that are repeated until the program is complete:

- write a new failing test
- write code to make tests pass
- optimize the new code

Currently, TDD refers to a wide variety of different test-first methods that are similar to the original TDD method.

Outline

Comparing Test Methods

When it comes to comparing two test methods there is no one attribute that proves one method is superior to another.

While there are many attributes that can be used to compare testing methods, I focused on three attributes:

- code coverage
- total development time
- code correctness

Kollanus Article Review Setup

In 2010, Kollanus reviewed 40 different articles that provided empirical evidence comparing TDD to test-last development methods.

The articles were split up into three types:

- controlled experiments
- case studies
- other

In this review Kollanus explores:

- code correctness
- productivity (which is similar to total development time)

Kollanus Code Correctness Results

In this review Kollanus found that 16 of the 22 studies stated that TDD methods increased code correctness.

This result prompted Kollanus to conclude that TDD may increase code correctness.

Kollanus was uncomfortable with this conclusion because 5 of the 7 controlled experiments did not find that TDD increased code quality.

Kollanus Productivity Results

In terms of productivity Kollanus found:

- 11 articles that found TDD increased development time
- 7 articles that found TDD caused no increase in development time
- 5 articles that found TDD decreased development time

Kollanus concludes in this study that TDD potentially increases development time. Though again this conclusion was uncomfortable to Kollanus who noted that more studies overall did not find that TDD increased development time.

Lemos et al Study Setup

In 2012, researchers Lemos et al from the Univ. of Sao Paulo preformed an experiment comparing test-first and test-last testing with auxiliary functions (10-200 lines of code).

In the study, Lemos had third year computer science students with basic test-first and test-last knowledge solve coding problems using test-first and test-last development methodologies.

Lemos et al Results

After the study was completed Lemos found that test-first testing:

- produced 40% more code coverage than test-last testing
- took 12% longer to implement than test-last testing
- did not increase code correctness

Contradictory Studies

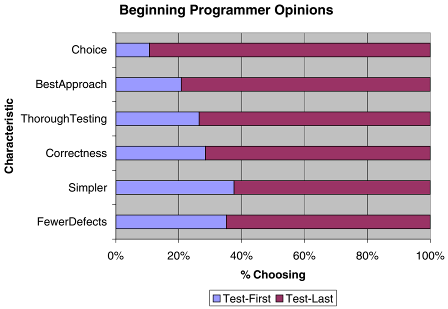
One large problem currently in the test-first vs test-last debate is that many studies contradict each other.

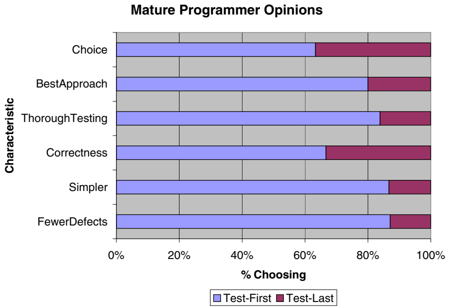
Although no one currently knows exactly why these contradictions occur, a few theories exist. Three potential problems in current test-first research that could produce contradictions are:

- lack of TDD method documentation
- test-first conformance issues
- Method difficulty

Method Diffuculty

Results from a Univ. of Kansas opinion study by Janzen et al:





Discussion of Results

Due to current issues in contradictory data it is hard to make solid conclusions.

That being said some strong trends were noticed:

- test-first testing takes at least as much time as test-last testing
- test-first testing tends to increase code coverage
- test-first testing has at least the same code correctness as test-last testing
- test-first testing is more difficult to implement then test-last testing

Outline

Difficulty of TDD

Most the data for test-first testing comes from studies using TDD. This means that the results have a potential to reflect properties of TDD and not test-first testing.

Although all the trends found in the previous slide have this potential issue, the most worrisome data point is method difficulty. This is because many articles noted that TDD can be a hard to implement for reasons other then test-first testing.

Difficulty of TDD Cont.

In a survey given to participants of a series of TDD experiments it was noted that:

- 56% of participants said they had difficulty adapting to the TDD mindset
- 23% of participants said that lack of upfront design was found to be a hindrance

Another survey, given online, found that **25% of TDD programmers admitted to frequently or always making mistakes in following the steps of TDD.** Some of these mistakes include:

- writing tests that are too complex for effective TDD
- forgetting to optimize their new code

Discussion

these studies suggest that TDD may increase method difficulty for test-first data.

Because of this, TDD may not be the best way to implement test-first testing.

Things To Take Away From This Presentation

- currently testing can be divided into test-first and test-last testing
- current data on comparing test-first to test-last testing tends to produce contradictory results over multiple studies, though a few main trends in the data exist
- TDD may not be the optimal way to perform test-first testing

Questions?

References

O. A. L. Lemos, F. C. Ferrari, F. F. Silveira and A. Gracia.
Development of auxiliary functions: should you be agile? An empirical assessment of pair programming and test-first programming.
In Proceedings of the 2012 International Conferance on Software Engineering. ICSE 2012, pages 529-539, Piscataway, NJ, USA, 2012. IEEE Press.

S. Kollanus.
Test-Driven development - still a promising approach?
In Proceeds of the 2010 Seventh International Conference on the Quality of Information and Communications Technology. QUANTIC 10, pages 403-408, Washington, DC, USA, 2010. IEEE Press.

D. S. Janzen and H. Saiedian.
A leveled examination of test-driven development acceptantance.
In Proceedings of 29th international conference on Software Engineering, ICSE 07, pages 719-722, Washington, DC, USA, 2007. IEEE Computer Society.