Test Driven Design (TDD)

## What is TDD?

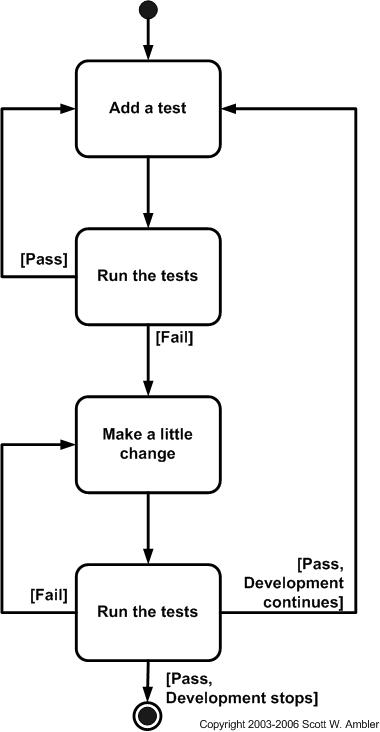


Figure : The Steps of test-first design (TFD).

The first step is to quickly add a test, basically just enough code to fail.  Next you run your tests, often the complete test suite although for sake of speed you may decide to run only a subset, to ensure that the new test does in fact fail.  You then update your functional code to make it pass the new tests.  The fourth step is to run your tests again.  If they fail you need to update your functional code and retest.  Once the tests pass the next step is to start over (you may first need to refactor any duplication out of your design as needed, turning TFD into TDD).

**TDD = Refactoring + TFD.**

TDD completely turns traditional development around. When you first go to implement a new feature, the first question that you ask is whether the existing design is the best design possible that enables you to implement that functionality.  If so, you proceed via a TFD approach.  If not, you refactor it locally to change the portion of the design affected by the new feature, enabling you to add that feature as easy as possible.  As a result you will always be improving the quality of your design, thereby making it easier to work with in the future.

Two simple rules for TDD:

* First, you should write new business code only when an automated test has failed.
* Second, you should eliminate any duplication that you find.

### TDD and Documentation

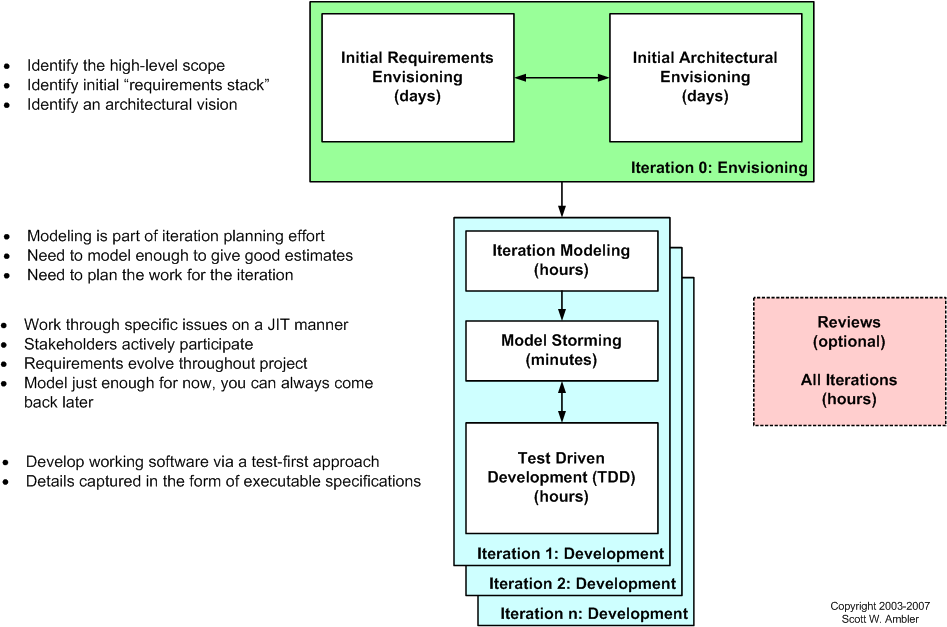
Like it or not most programmers don’t read the written documentation for a system, instead they prefer to work with the code.  And there’s nothing wrong with this.  When trying to understand a class or operation most programmers will first look for sample code that already invokes it.  Well-written unit tests do exactly this –provide a working specification of your functional code – and as a result unit tests effectively become a significant portion of your technical documentation. The implication is that the expectations of the pro-documentation crowd need to reflect this reality.  Similarly, acceptance tests can form an important part of your requirements documentation.  This makes a lot of sense when you stop and think about it.  Your acceptance tests define exactly what your stakeholders expect of your system. Therefore they specify your critical requirements.  Your regression test suite, particularly with a test-first approach, effectively becomes detailed executable specifications.

Are tests sufficient documentation?  Very likely not, but they do form an important part of it.

### Why TDD?

A significant advantage of TDD is that it enables you to take small steps when writing software. It is much easier to find, and then fix, those defects if you've written two new lines of code than two thousand. The implication is that the faster your compiler and regression test suite, the more attractive it is to proceed in smaller and smaller steps.

### Agile Model-Driven Development (AMDD)



Comparing TDD and AMDD:

* TDD shortens the programming feedback loop whereas AMDD shortens the modeling feedback loop.
* TDD provides detailed specification (tests) whereas AMDD is better for thinking through bigger issues.
* TDD promotes the development of high-quality code whereas AMDD promotes high-quality communication with your stakeholders and other developers.
* TDD provides concrete evidence that your software works whereas AMDD supports your team, including stakeholders, in working toward a common understanding.
* TDD “speaks” to programmers whereas AMDD speaks to business analysts, stakeholders, and data professionals.
* TDD is provides very finely grained concrete feedback on the order of minutes whereas AMDD enables verbal feedback on the order minutes (concrete feedback requires developers to follow the practice Prove It With Code and thus becomes dependent on non-AM techniques).
* TDD helps to ensure that your design is clean by focusing on creation of operations that are callable and testable whereas AMDD provides an opportunity to think through larger design/architectural issues before you code.
* TDD is non-visually oriented whereas AMDD is visually oriented.
* Both techniques are new to traditional developers and therefore may be threatening to them.
* Both techniques support evolutionary development.