Test-Driven Development
Reference

About Seapine Agile Services

Seapine Agile Services provides transformation and training solutions to maximize your organization's performance. Our Agile consultants address your unique development and training needs with a pragmatic, collaborative approach that is tool- and methodology-neutral. Whether your needs are technology- or process-related, Seapine Agile Services helps your organization become more innovative, while improving quality and lowering costs.

Find out more at
www.seapine.com/agileservices

Also, check out the Seapine Agile expedition at
www.seapine.com/exploreagile

The Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions *over processes and tools*
Working software *over comprehensive documentation*
Customer collaboration *over contract negotiation*
Responding to change *over following a plan*

That is, while there is value in the items on the right, we value the items on the left more.

Agile Manifesto, © 2001
www.agilemanifesto.org

Seapine Software™

"A journey of a thousand miles begins with a single step, and an application of a thousand lines should begin with a single test."
Katie Dwyer

What is Test-Driven Development (TDD)?

TDD is the practice of writing a unit test before writing any code. This can be done relatively quickly, with the developer writing the test, then writing the code, and then running the test, all in small increments. TDD ensures the code is consistently refactored for a better design. However, TDD isn't a replacement for regression suites, design specifications, QA, or code reviews.

TDD = TFD (Test First Design) + Refactor

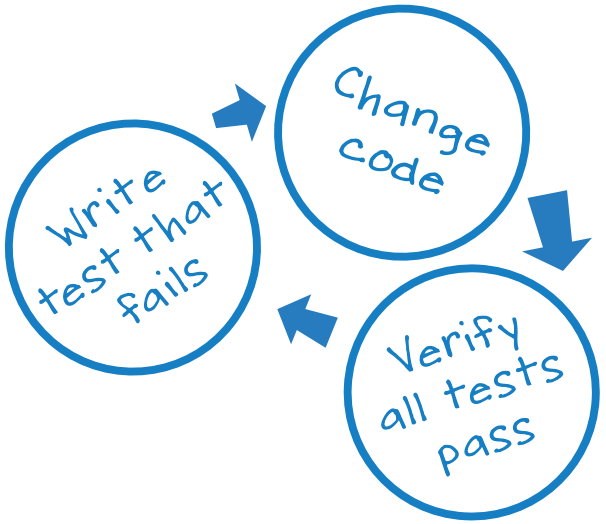
Contrary to popular opinion, Agile is not cowboy coding. If there's no BDUF (Big Design Up Front), how do you ensure your coding process has some structure? TDD is one answer.

What's the goal of TDD?

To write "clean code that works". (According to Ron Jeffries, **http://xprogramming.com/index.php.**)

What are the benefits of TDD?

- Better code design
- Earlier defect detection
- Fewer defects in production releases
- Significant drop in defect density
- Disciplined, quality-focused coding practices



TDD Activities

Step	Activity
1	Write a test to verify new or changed functionality.
2	Run all tests and verify the new one fails (since it hasn't been implemented yet).
3	Change the code just enough so that all tests pass. Your goal at this stage is to pass the test, so the code may not be elegant.
4	Refactor as needed to clean up the code. Verify all tests still pass after the refactor.
5	Repeat the process for the next change.

Working with Legacy Code

You can use TDD with your existing legacy code, but it can be overwhelming if the code wasn't designed to be testable. Following are a few tips to help you get started:

1. Refactor the code you are changing so it is testable.
2. Look for "code smells"—symptoms in the source code that could indicate a deeper problem.
3. Don't worry about code coverage in old code.

To learn more, check out *Working Effectively with Legacy Code*, by Michael Feathers.

Writing a Good Unit Test

Good unit tests are the backbone of successful TDD. Keep the following in mind as you write unit tests:

- Each unit test should be independent.
- Each unit test should test one aspect or behavior and document the expected behavior.
- Each unit test should not verify too much functionality.
- Each unit test should not be dependent on interface.

Sample Unit Test: Constructor Default Values

```
/* The purpose of this test is to verify the default values
   branchid should be INVALID_RECORD_ID, recursive should be set to true
*/
TEST(SCMWorkingDirTestSuite_Constructor_DefaultValues)
   {
      CSCMWorkingDirListObj workDir;
      CHECK_EQUAL(INVALID_RECORD_ID, workDir.GetBranchID());
      CHECK_EQUAL(static_cast<Boolean>(TRUE), workDir.GetRecursive());
   };
```


Seapine Software™