The Iteration Skeleton

Each TDD iteration follows the Rhythm

The TDD Rhythm:

1. Quickly add a test
2. Run all tests and see the new one fail
3. Make a little change
4. Run all tests and see them all succeed
5. Refactor to remove duplication

(6. All tests pass again after refactoring!)

The Rhythm: Red-Green-Refactor

The Rhythm

Improve code quality

Clean part

Implement delta-feature that does not break any existing code

Works part

Introduce test of delta-feature

Principle Summary

TDD Principle: **One Step Test**

Which test should you pick next from the test list? Pick a test that will teach you something and that you are confident you can implement.

TDD Principle: **Fake It ('Til You Make It)**

What is your first implementation once you have a broken test? Return a constant. Once you have your tests running, gradually transform it.

TDD Principle: **Triangulation**

How do you most conservatively drive abstraction with tests? Abstract only when you have two or more examples.

TDD Principle: **Obvious Implementation**

How do you implement simple operations? Just implement them.

Principle Summary

TDD Principle: **Isolated Test**

How should the running of tests affect one another? Not at all.

TDD Principle: **Evident Tests**

How do we avoid writing defective tests? By keeping the testing code evident, readable, and as simple as possible.

TDD Principle: **Evident Data**

How do you represent the intent of the data? Include expected and actual results in the test itself, and make their relationship apparent. You are writing tests for the reader, not just for the computer.

TDD Principle: **Representative Data**

What data do you use for your tests? Select a small set of data where each element represents a conceptual aspect or a special computational processing.