

Issues with unit tests after production code:

Hard to deduce the requirements

Intrinsic bias in testing of the self-written code

Can't know whether all requirements were tested

Boring

Test Driven Development

All tests up-front:

Create an empty class for SUT

Write all the required tests

You can expand SUT's public API to make the tests compile

You can't write internal implementation of SUT

Write the internal implementation of the SUT

Run the tests

Debug until all tests pass

All tests up-front benefits and limitations:

- + Hard to deduce the requirements
- + Intrinsic bias in testing of the self-written code
- Can't know whether all requirements were tested
- + - Boring



Three Rules of TDD by Uncle Bob



1. You are not allowed to write any production code unless it is to make a failing unit test pass
2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures
3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test

You are not allowed to write any production code
unless it is to make a failing unit test pass

You are not allowed to write any more of a unit test
than is sufficient to fail; and compilation failures
are failures

You are not allowed to write any more production code than is sufficient to pass the one failing unit test

Uncle Bob's technique benefits and limitations:

- + Hard to deduce the requirements
- + Intrinsic bias in testing of the self-written code
- + Can't know whether all requirements were tested
- + Boring

Uncle Bob's technique additional benefits:

Design requires less mental effort

Sometimes you have a bad day and can't do design, but you can always write a simple test and make it pass [probably Kent Beck]

Uncle Bob's technique special requirement:

Fast cycle of incr. build + test

Synchronous execution:

All effects of method call happen before the method returns

Return value contains flow execution result

Asynchronous execution:

Method call can have effects after the method returns

Flow execution result will be known after the method returns

How to get the results of asynchronous execution?

Observer!

Test Driven Development

Scope of TDD:

Writing unit tests after the production code is hard

For me Unit Testing = TDD

TDD is a skill that needs to be practiced

All test up-front:

Good fit for small classes (three test cases or less)

Can be used on projects with long build times

Uncle Bob's technique:

Granular

Dynamic

Guaranteed functional coverage

Reduces mental complexity of software development

Requires reasonable build times

Async execution:

Observer design pattern

Be pragmatic!