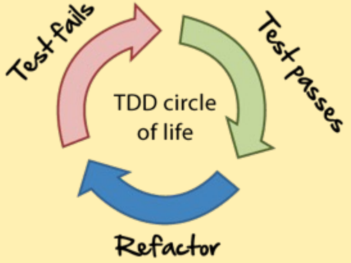
Technische Universität Braunschweig

Suresoft
SUSTAINABLE RESEARCH SOFTWARE



Introduction to Test Driven Development

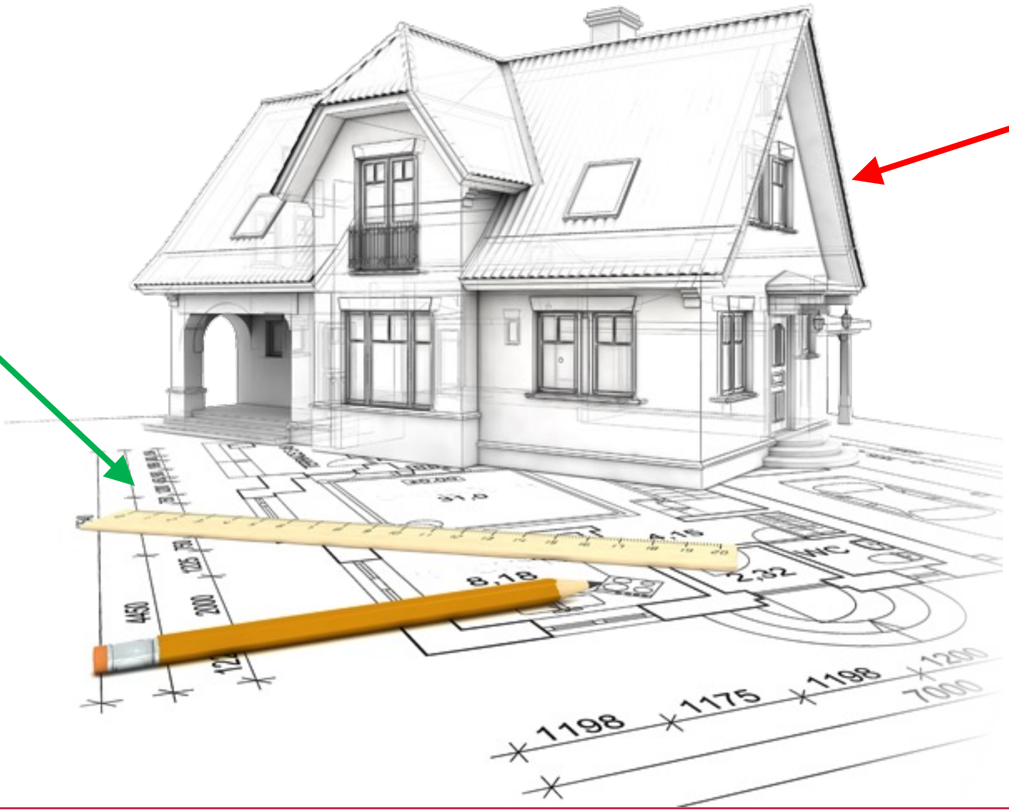
Agenda

- Motivation - Traditional vs. Agile Processes
- Benefits of TDD
- Process
- TDD in Action: HVAC - KATA
- Outlook

Traditional vs. Agile Processes

Design

cheap



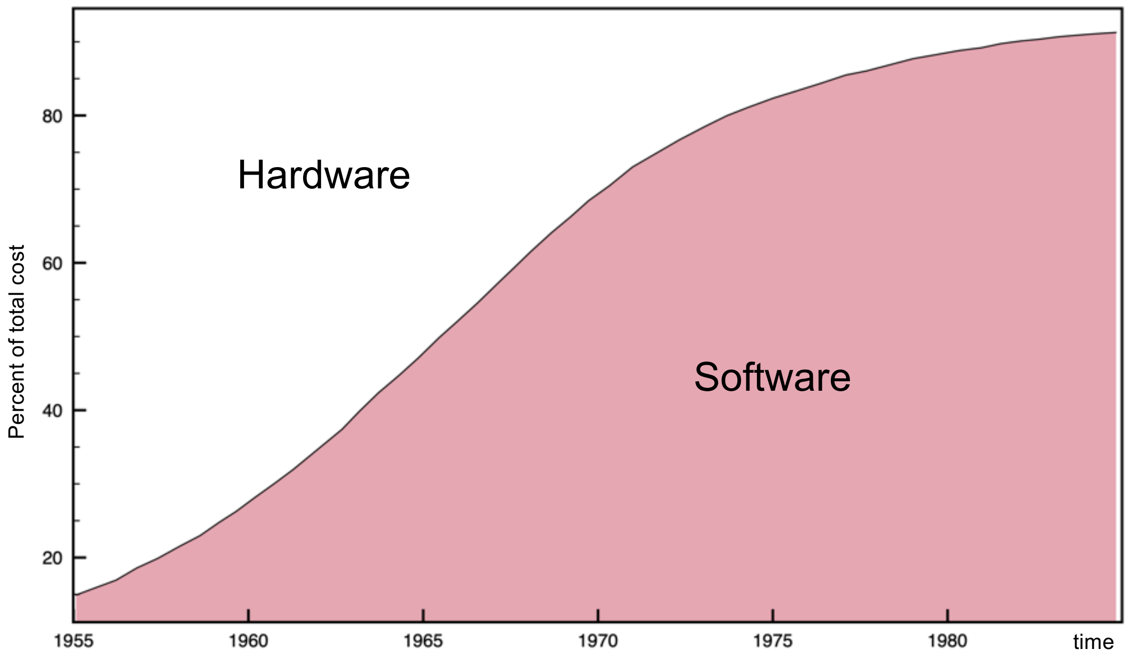
Product

expensive

Technische Universität Braunschweig

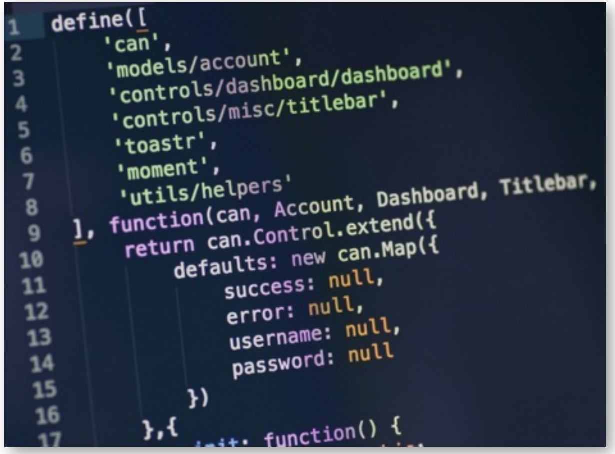
Suresoft
SUSTAINABLE RESEARCH SOFTWARE

Evolution of costs for Hardware vs. Software



[Barry W. Boehm, Software and its impact: A quantitative assessment, 1973]

Traditional vs. Agile Processes

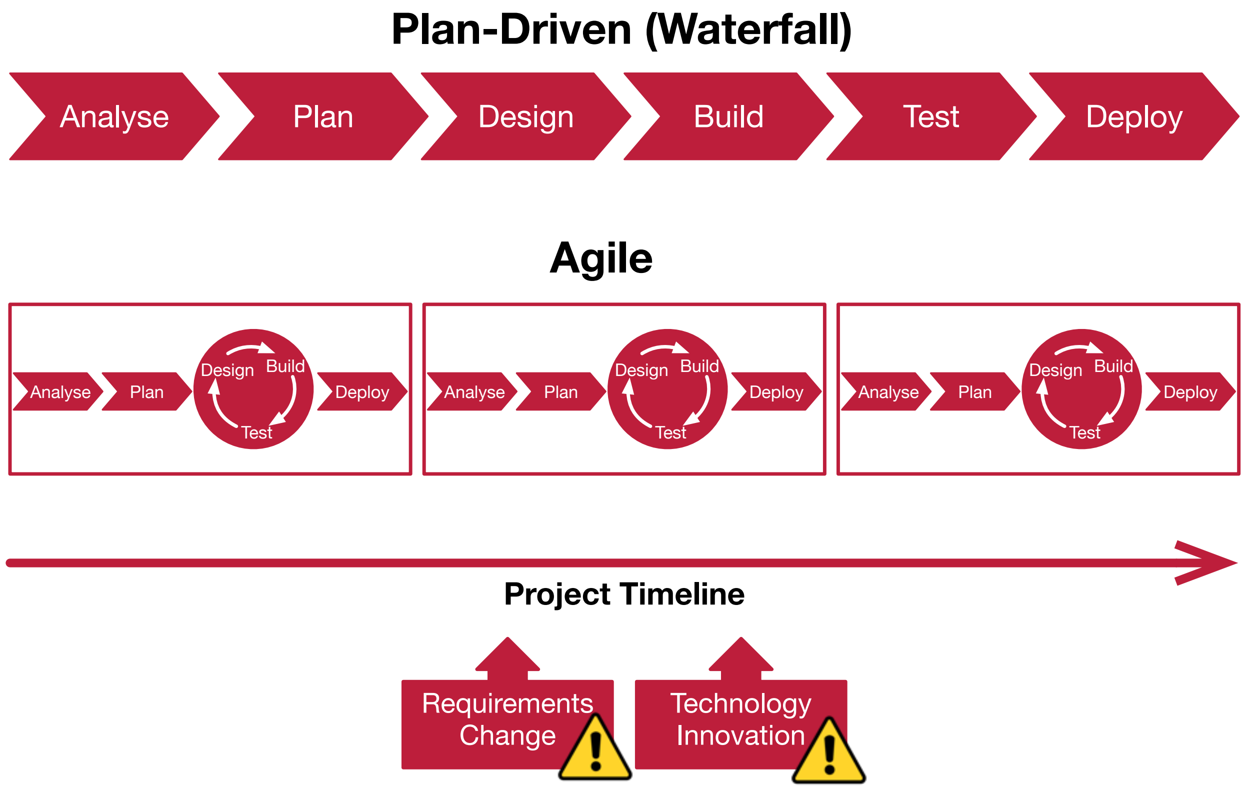


```
define([
    'can',
    'models/account',
    'controls/dashboard/dashboard',
    'controls/misc/titlebar',
    'toastr',
    'moment',
    'utils/helpers'
], function(can, Account, Dashboard, Titlebar,
    return can.Control.extend({
        defaults: new can.Map({
            success: null,
            error: null,
            username: null,
            password: null
        })
    },{
        init: function() {
```

Design
expensive

Product
cheap

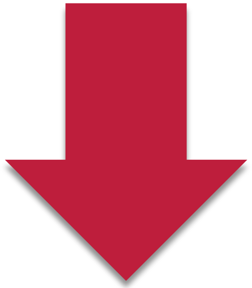
Traditional vs. Agile Processes



Traditional vs. Agile Processes

Design for Change

- Software Engineering is about designing for change.
- Fast feedback loop
- Low coupling & high cohesion (SOLID Principles)

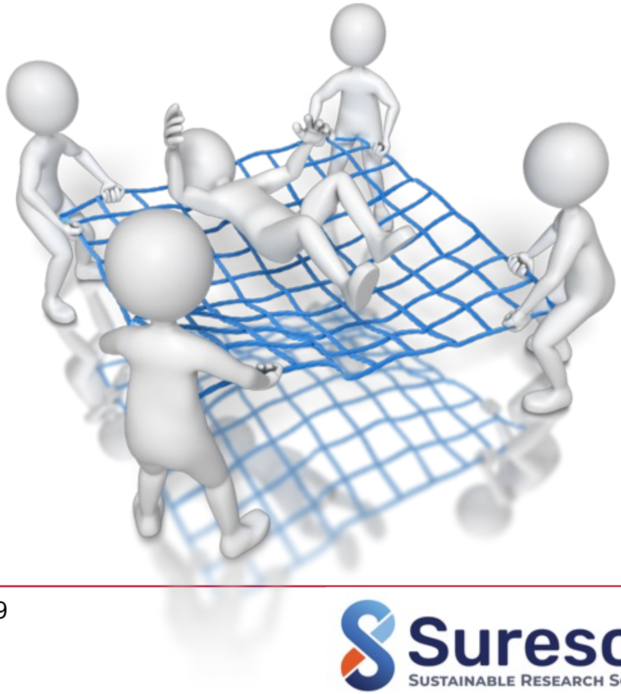


TDD

Software Development is not a Jenga game!!

Benefits of TDD

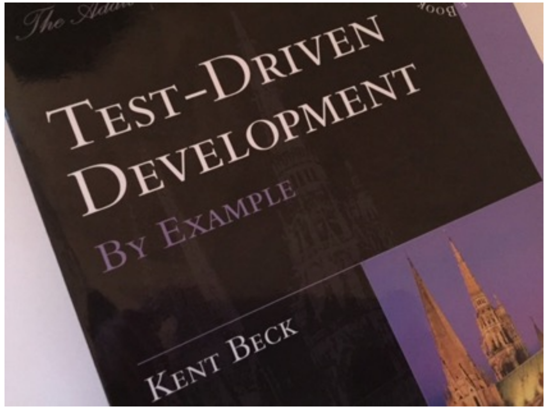
- Supports **low coupling & cohesion** otherwise testing is hard
- Safety Net – Rapid Response -> eliminates the **fear of change**
- Reduces **debugging time**
- Reliable low level **documentation**
- Shift of **perspective**: developer to user



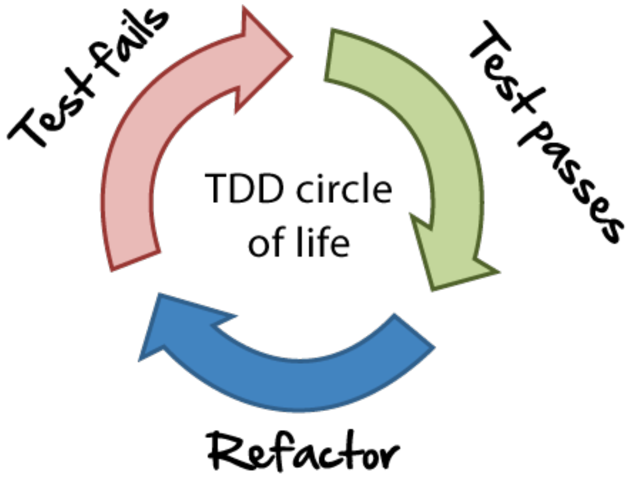
Roots of TDD

- Kent Beck is credited with having developed TDD in 2003
- Has been proven to work (IBM, Microsoft, Sabre..)

  defect reduction rate of 2x, 5x even 10x
- Part of all agile software dev. processes (XP, Scrum, etc.)



The TDD Waltz …



- Red - write a test – Write a single test that doesn‘t work, and perhaps even compile at first.
- Green – make it work – Do the simplest thing to make the test work.
- *Refactor* – make it right – Eliminate all duplication and clean up your code.

Uncle Bob‘s Three Rules of TDD

Three rules of test driven development:

1. You are not allowed to write any production code until you have first written a failing test.
2. You are not allowed to write more of a unit test that it is sufficient to fail – and not compiling is failing.
3. Your are not allowed to write more production code that is sufficient to pass the current failing test.

Definition of a Unit Test

A *unit test* is an automated piece of code that invokes the **unit of work** being tested, and then checks some **assumptions** about **a single end result** of that unit. A unit test is almost always written using a unit testing framework. It can be written easily and runs quickly. It’s trustworthy, readable, and maintainable. It’s consistent in its results as long as production code hasn‘t changed.

Structure of a Unit Tests – 4 As

- **Arrange -** sets up system state (Test Fixture) ready to be tested
- **Act** - does the thing you are testing / acts on the test fixture
- **Assert** - does the test / asserts the state of the test fixture
- **Annihilate** - tears everything down

unittest – Unit Testing Framework in Python

***unittest** is a testing framework form the Python Standard Library that is suitable for automated testing of single units (mostly classes or methods).*

- `assertTrue(x)`
- `assertFalse(x)`
- `assertNotEqual(a, b)`
- `assertNotEqual(a, b)`
- `assertRaises()`

```
import unittest

class MathTest(unittest.TestCase):
  def test_multiplication():
      self.assertEqual(3*3, 9, “3*3 should be 9”)
```

https://docs.python.org/3/library/unittest.html

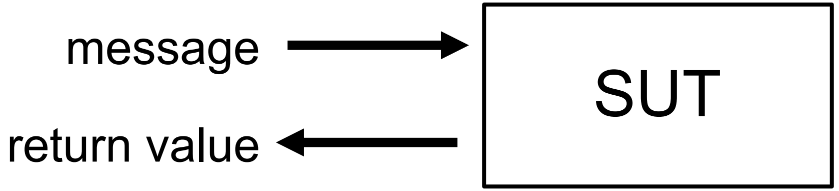
pytest – Unit Testing Framework

pytest is a unit testing framework for python that is suitable for automated testing of single units (mostly classes or methods).

```
•    assert
```

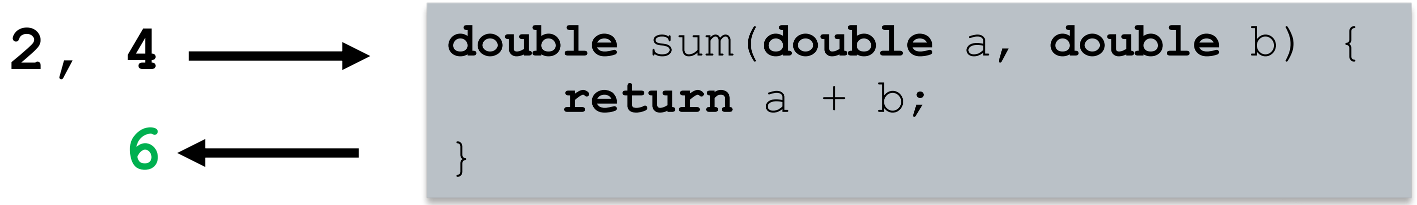
```
def test_multiplication():

 assert 3 * 3 == 9
```

Three Types of Tests



Return Value Verification

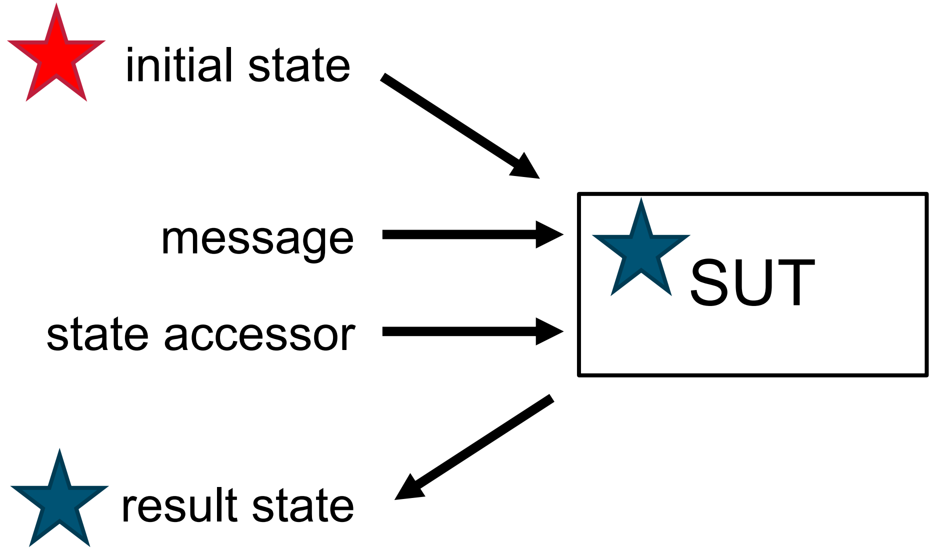
Three Types of Tests



```
double sum(double a, double b) {
    return a + b;
}
```

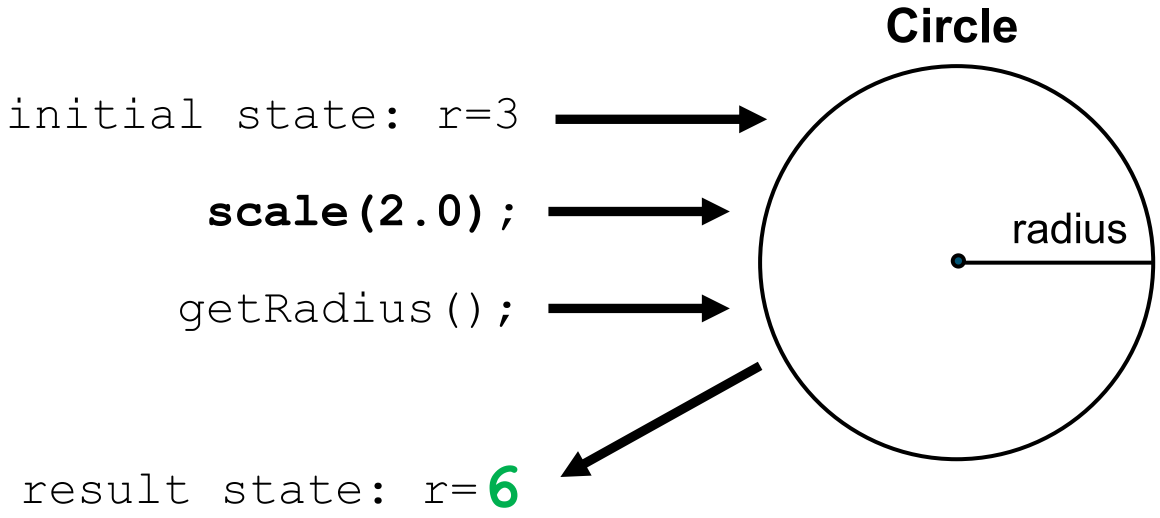
Return Value Verification

Three Types of Tests



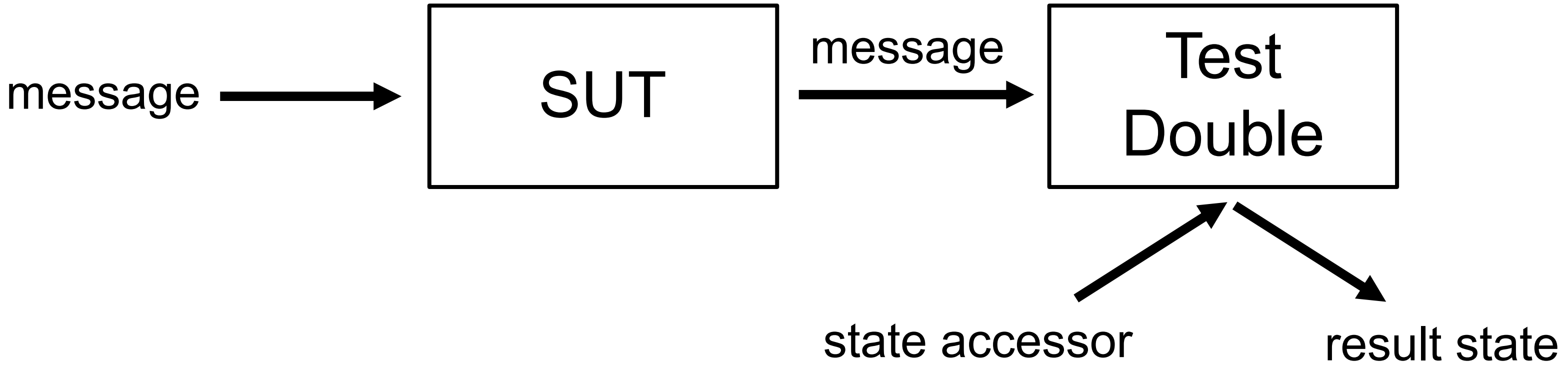
State Verification

Three Types of Tests



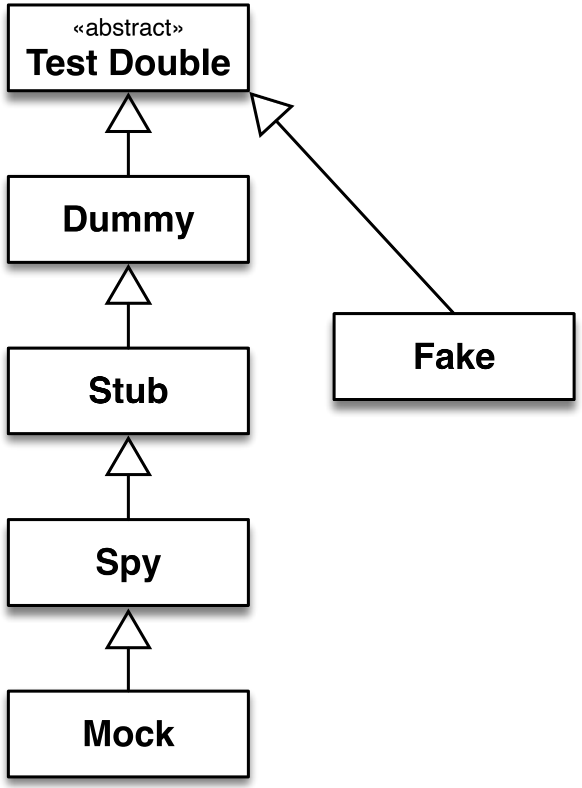
State Verification

Three Types of Tests



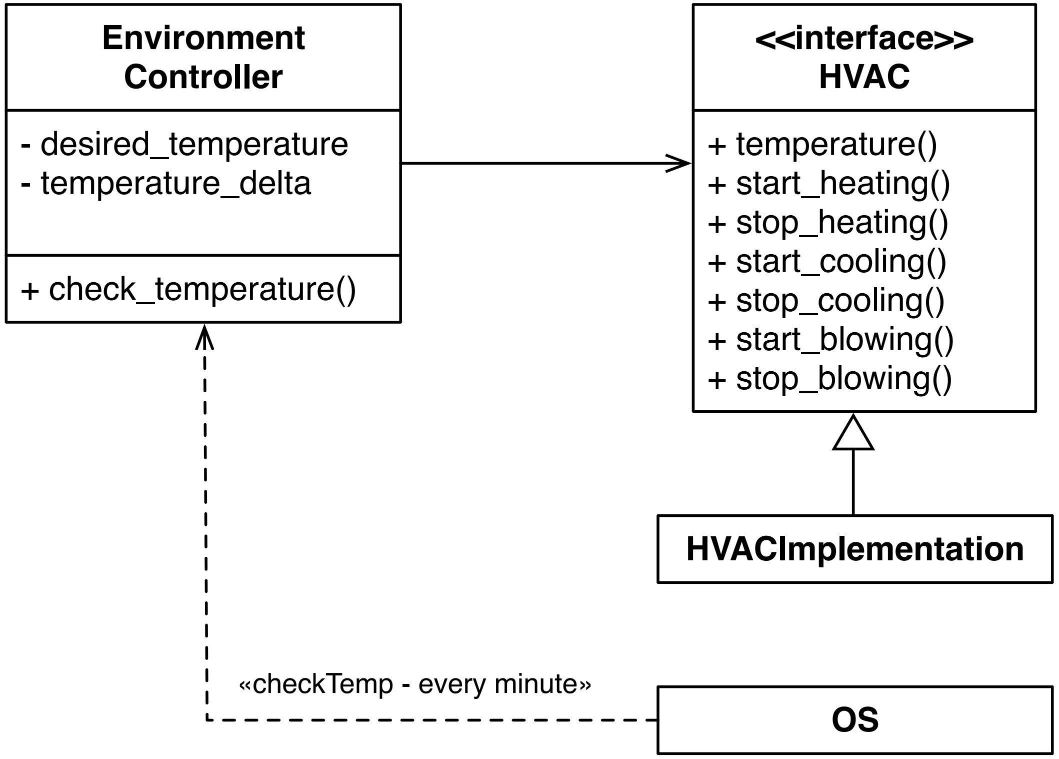
Behavior Verification

The Ontology of Mock Objects

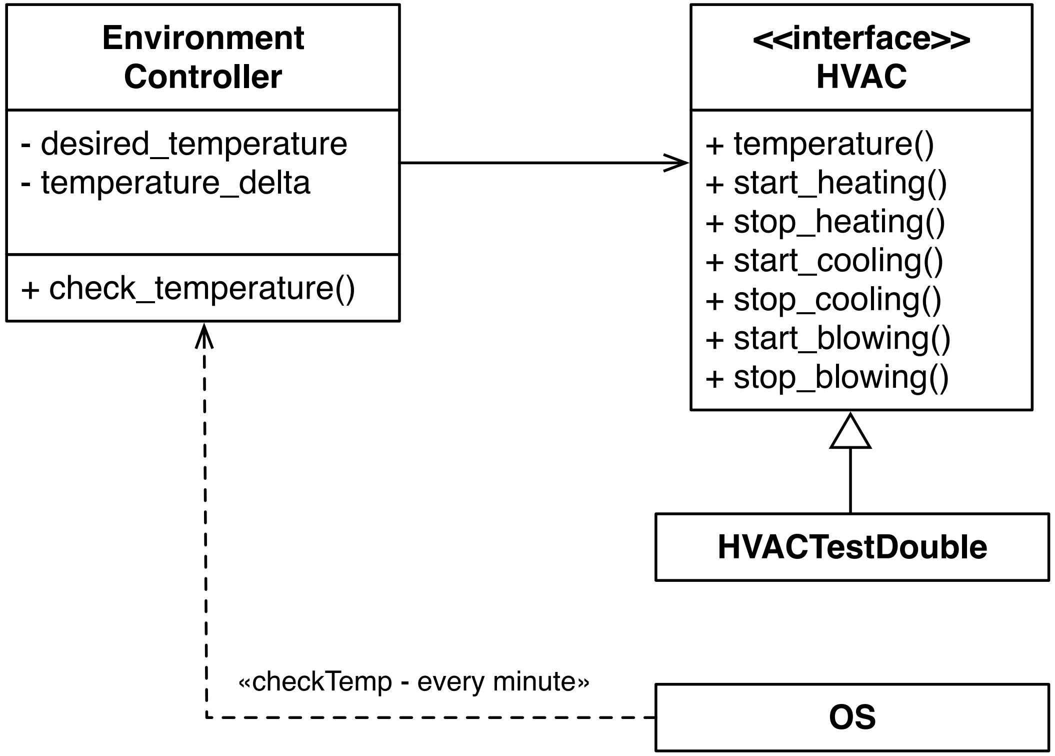


Gerard Meszaros, 2007,
XUnit Test Patterns: Refactoring Test Code

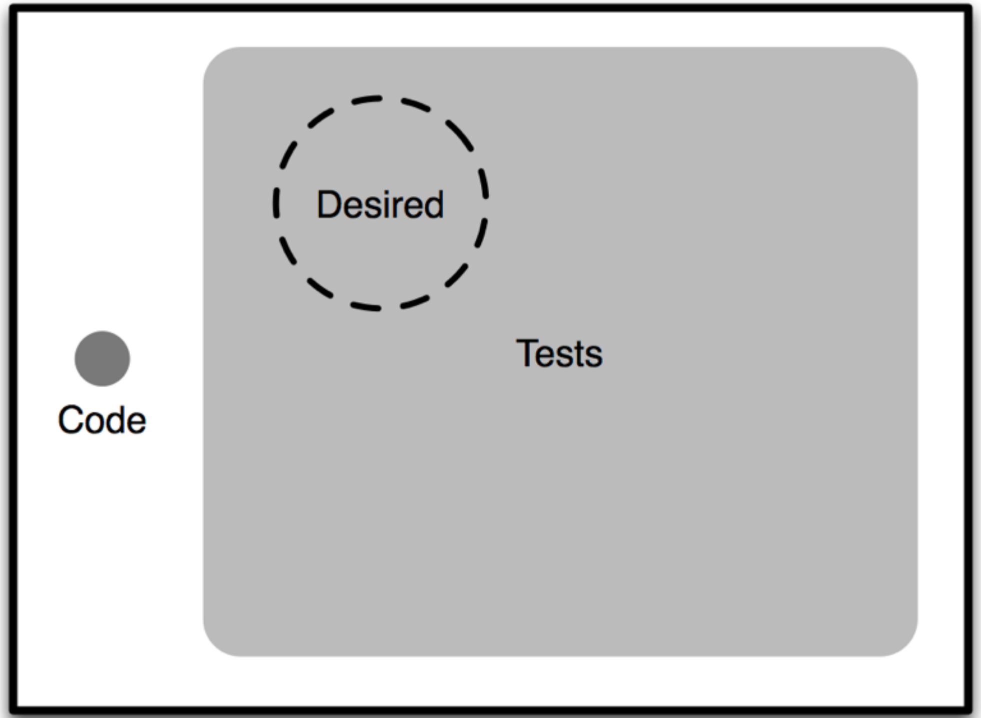
Environment Controller Kata



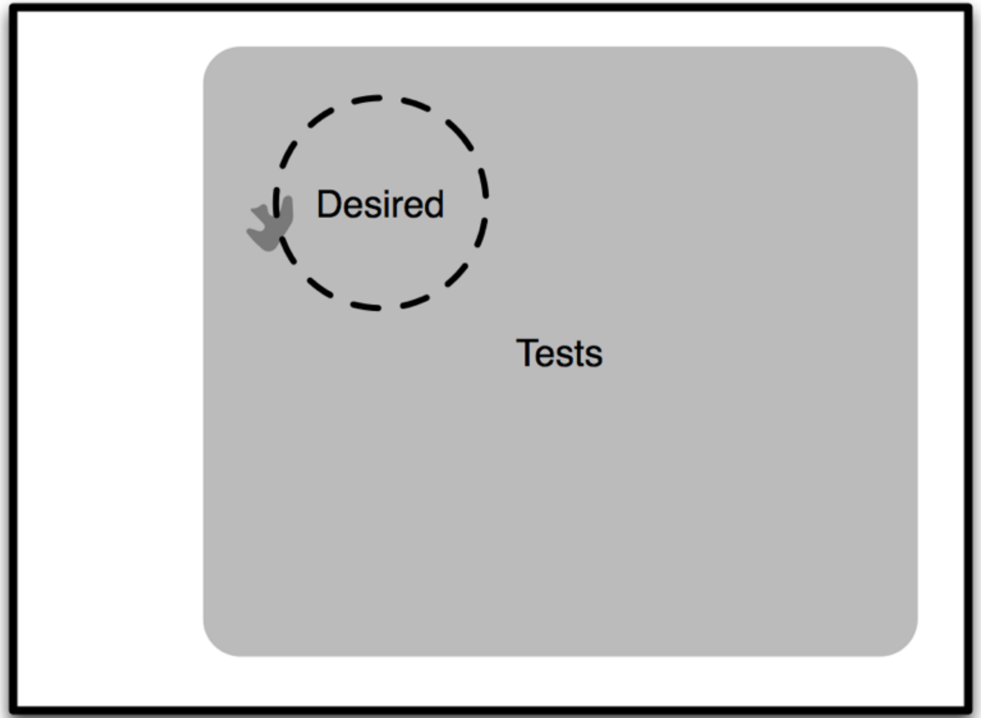
Environment Controller Kata



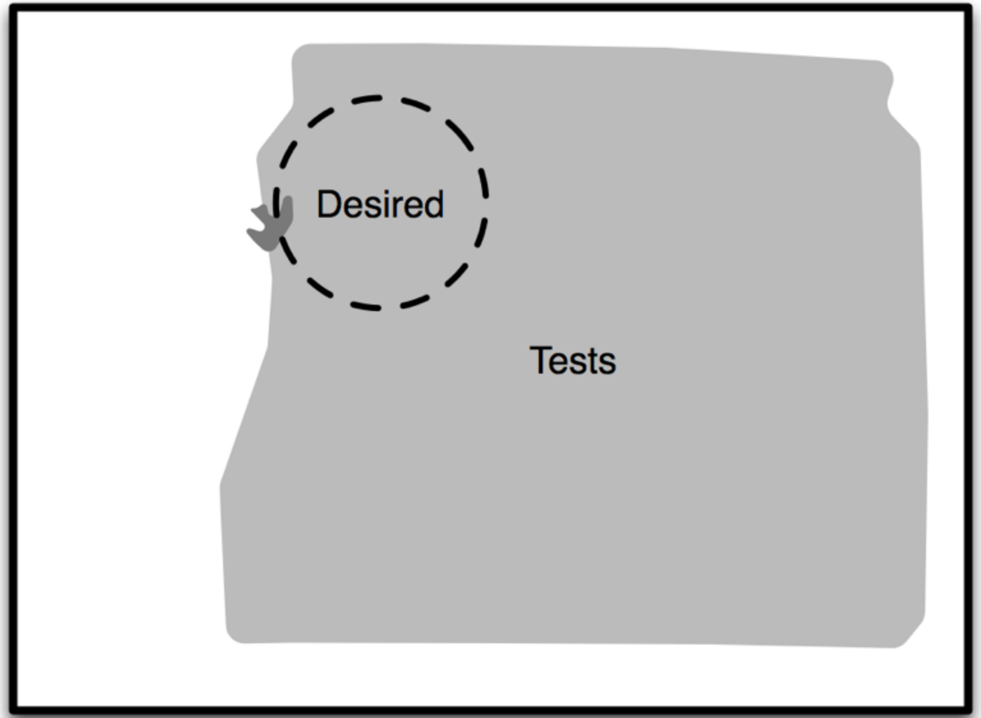
Tests = Constrains



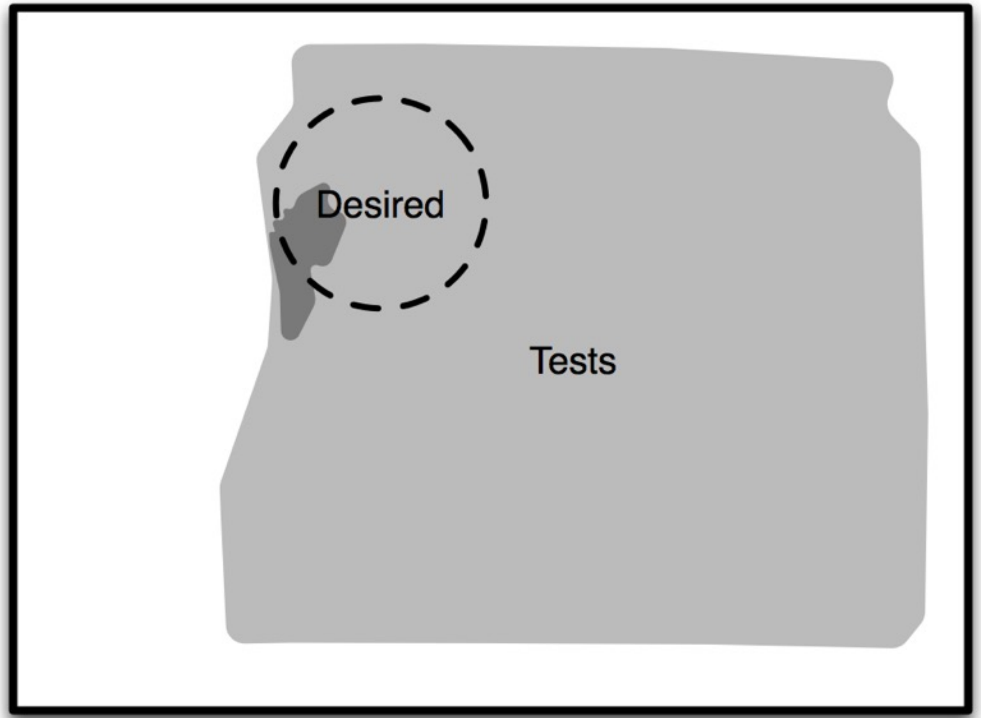
Tests = Constrains



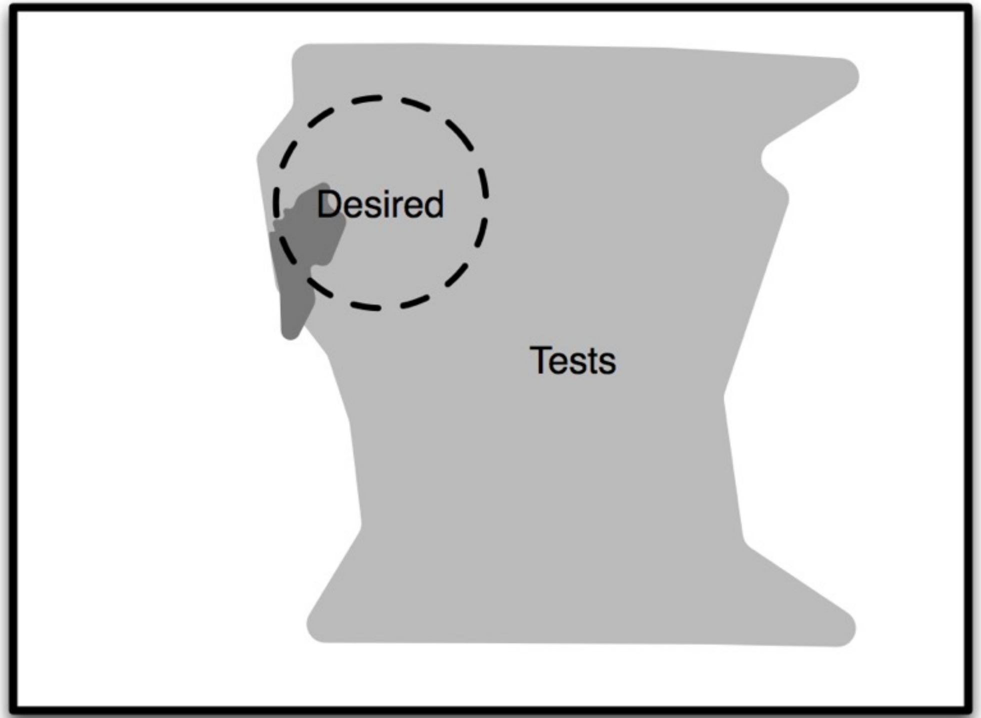
Tests = Constrains



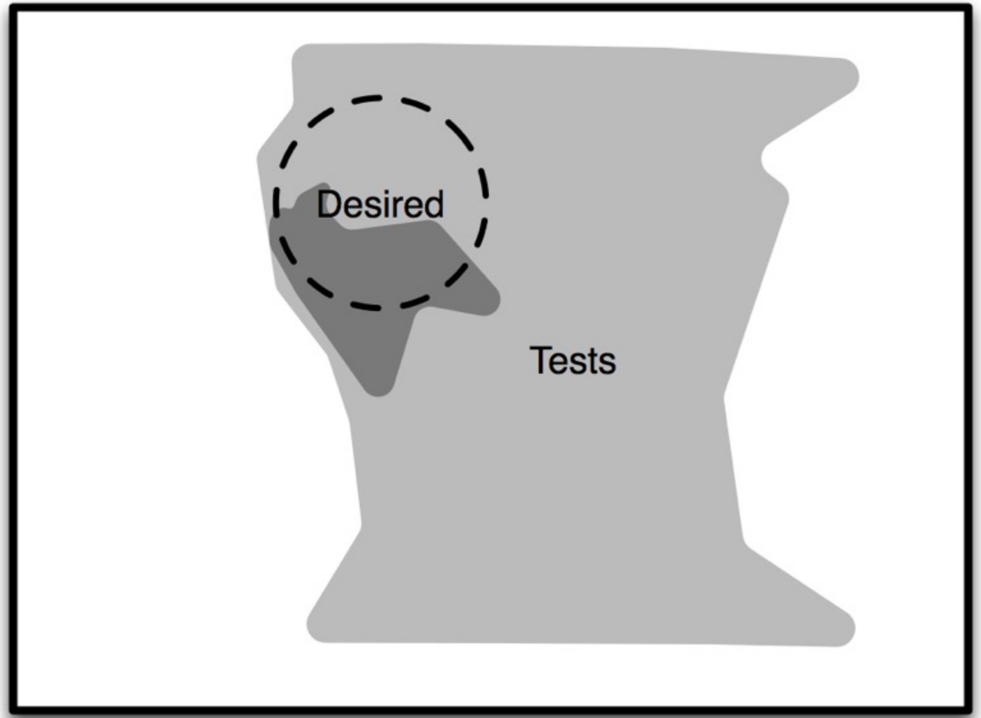
Tests = Constrains



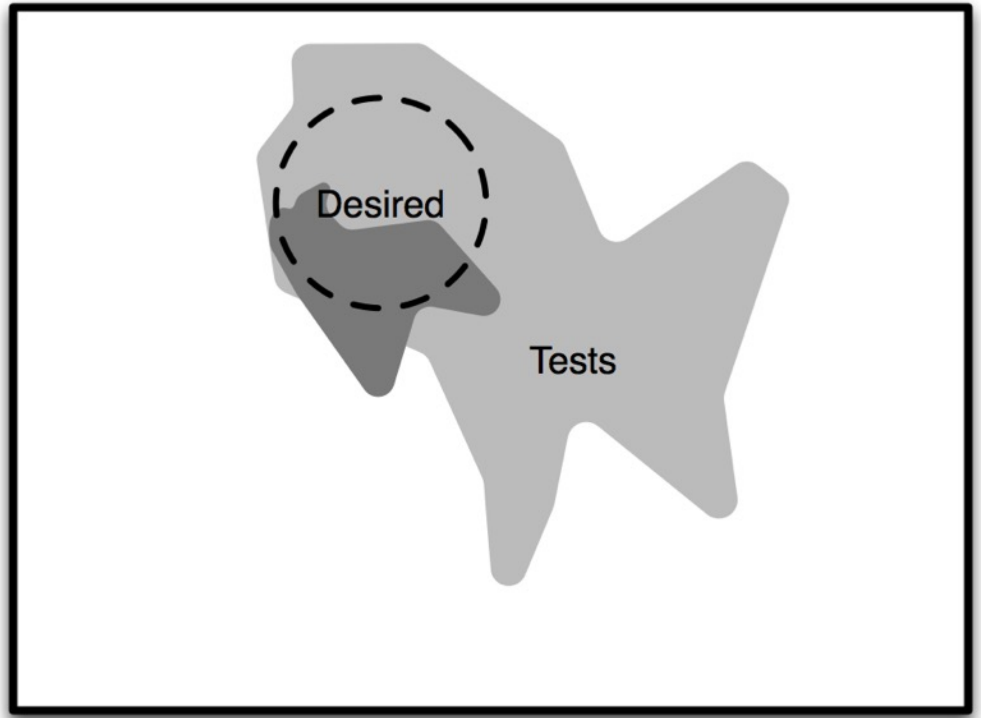
Tests = Constrains



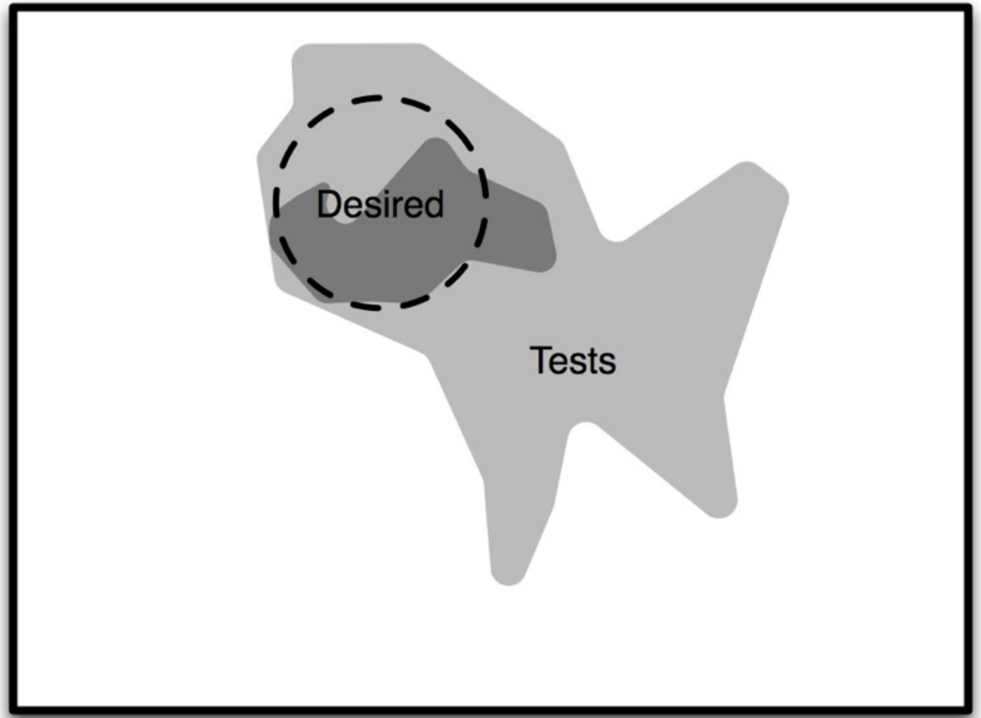
Tests = Constrains



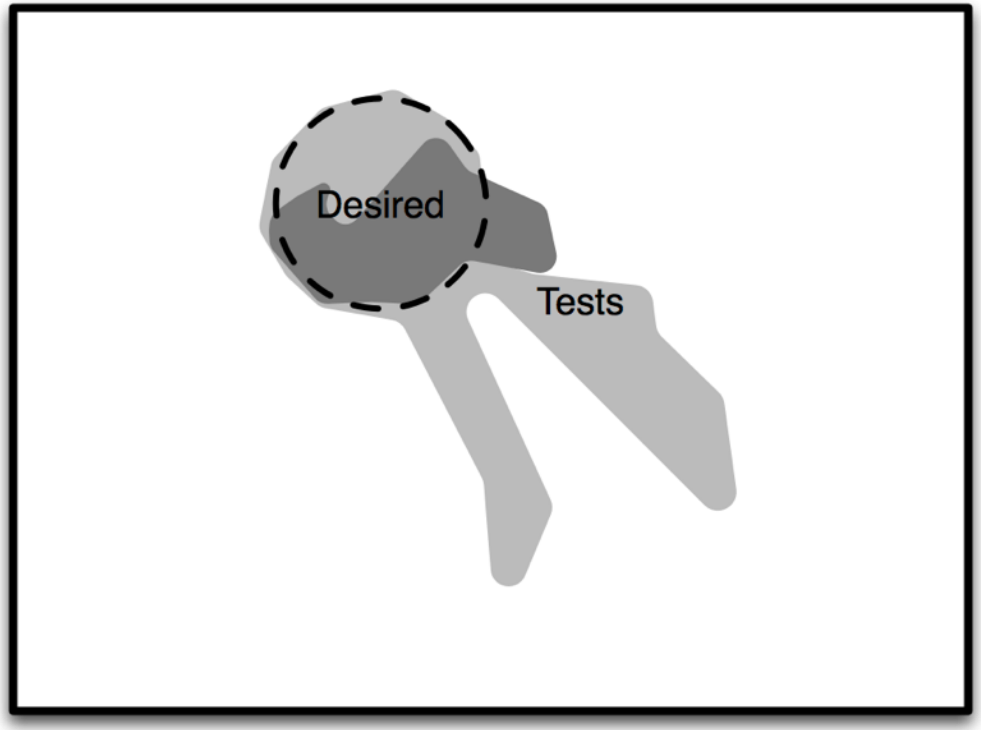
Tests = Constrains



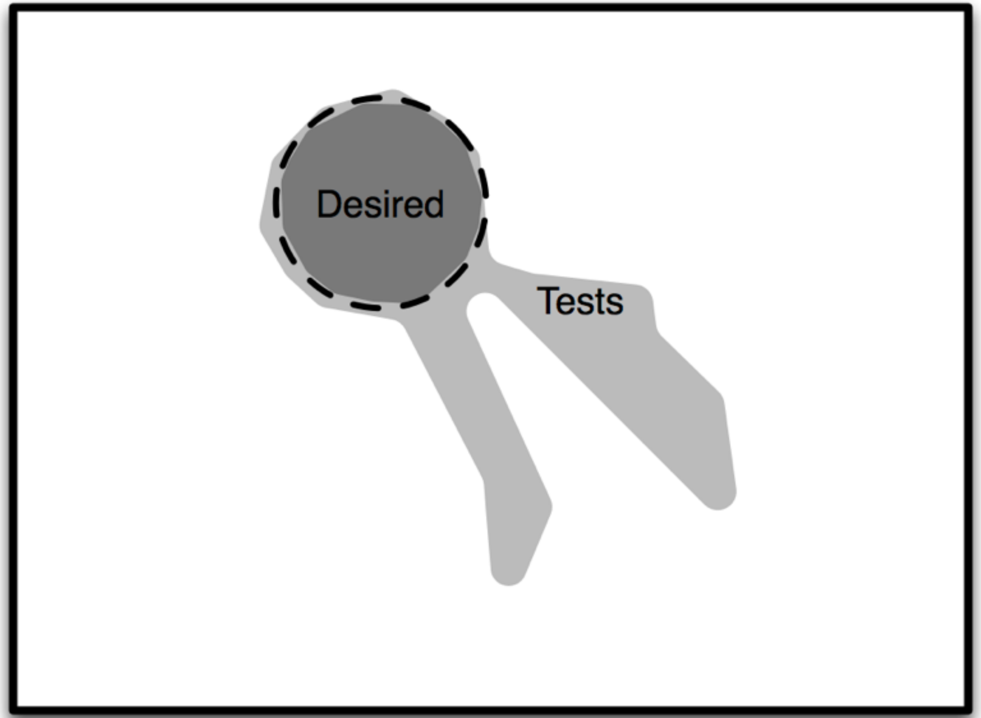
Tests = Constrains



Tests = Constrains



Tests = Constrains

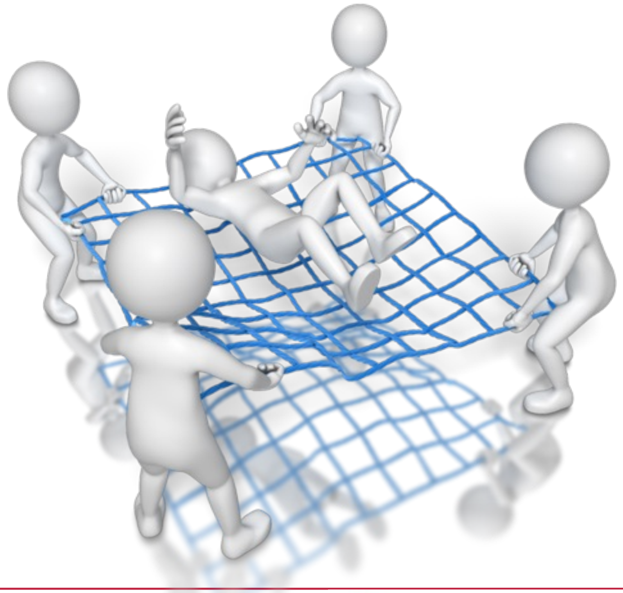


Tests = Constrains

- Tests can constrain the behavior but they can’t specify it.
- Test can only proof you program being wrong they can not proof it being right.

Benefits of TDD

- Safety Net – Rapid Response -> eliminates the **fear of change**
- Supports **low coupling & cohesion** otherwise testing is hard
- Reduces **debugging time**
- Reliable low level **documentation**
- Shift of **perspective**: developer to user



Question

Why should you write your tests first?

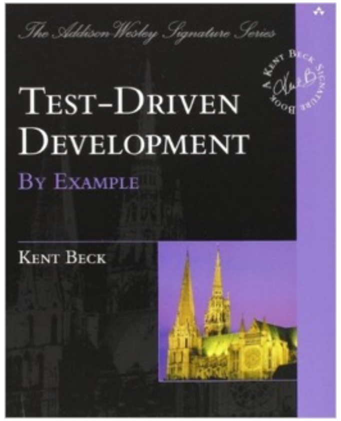
Question

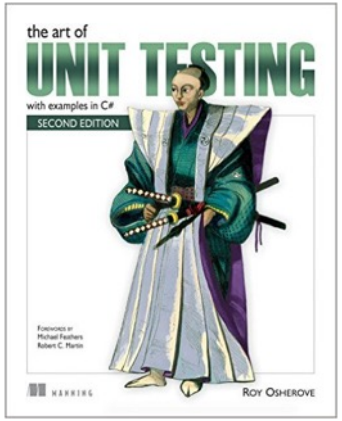
Why should you write your tests first?

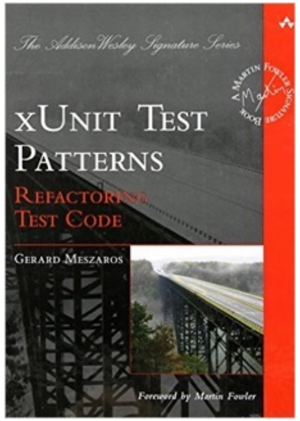
- To achieve 100% test coverage (goal)
- The production code tests your tests
- In case they are optional you probably won’t write them

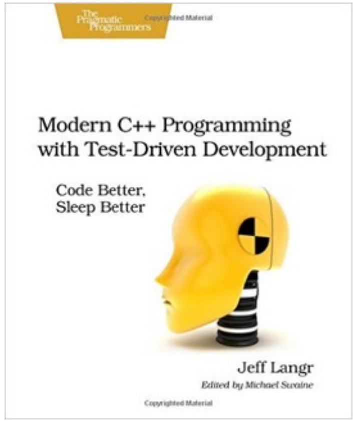
Outlook

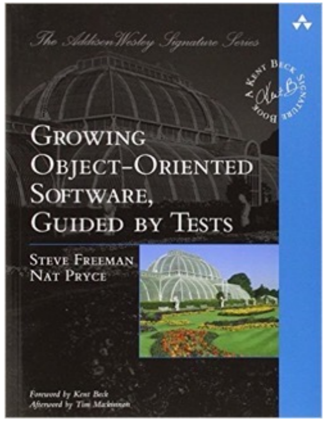
- Test Types (integration test, acceptance tests, etc.)
- Mocking Frameworks
- Google Mock & Google Test (C++)
- Magic Tricks of Testing (Sandi Metz)
- Versioning Systems (Git) / Continuous Integration

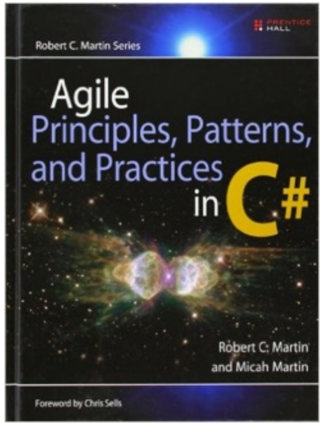

The Addison Wesley Signature Series
Test-Driven Development
By Example
Kent Beck


the art of
UNIT TESTING
with examples in C#
SECOND EDITION
Forewords by Michael Feathers Robert C. Martin
MANNING
Roy Osherove


The Addison Wesley Signature Series
xUnit Test Patterns
Refactoring Test Code
Gerard Meszaros
Foreword by Martin Fowler


The Pragmatic Programmers
Modern C++ Programming with Test-Driven Development
Code Better, Sleep Better
Jeff Langr
Edited by Michael Swaine


The Addison Wesley Signature Series
Growing Object-Oriented Software, Guided by Tests
Steve Freeman
Nat Pryce
Foreword by Kent Beck
Afterword by Tim Mackinnon


Robert C. Martin Series
Agile Principles, Patterns, and Practices in C#
Robert C. Martin and Micah Martin
Foreword by Chris Sells

Thanks!

The SURESOFT project is funded by the German Research Foundation (DFG) as part of the “e-Research Technologies” funding programme under grants: EG 404/1-1, JA 2329/7-1, KA 3171/12-1, KU 2333/17-1, LA 1403/12-1, LI 2970/1-1 and STU 530/6-1.