Agile Scrum & Test Driven Development

Dr. Julie Greensmith

Overview

- Agile Scrum Practices
- Test Driven Development in Context
- Contrasting TDD with traditional development approaches
- A TDD example
- The advantages of TDD in practice

Agile Development

- Condensed to 5 Main Principles:
  - Deliver Early and Often to Satisfy Customer
  - Welcome Changing Requirements
  - Face to Face Communication is Best
  - Measure Progress against Working Software
  - Simplicity is Essential

eXtreme Programming (XP)

- XP is centred around agile principles
  - also including minimal documentation and paris programming and everybody
- Test Driven Development is a technique which can be paired nicely with XP
  - make the coding process more efficient
  - focus is on failing and passing tests during the implementation part of a project

Alternative to XP is Scrum

- An agile team is termed a ‘scrum’
- Progress is charted using a scrum board
- Using iterative development over short periods known as ‘sprints’
- Daily meetings of the scrum occur known as stand ups and are 5-10 meetings long
- The team leader is termed the scrum master

A Scrum

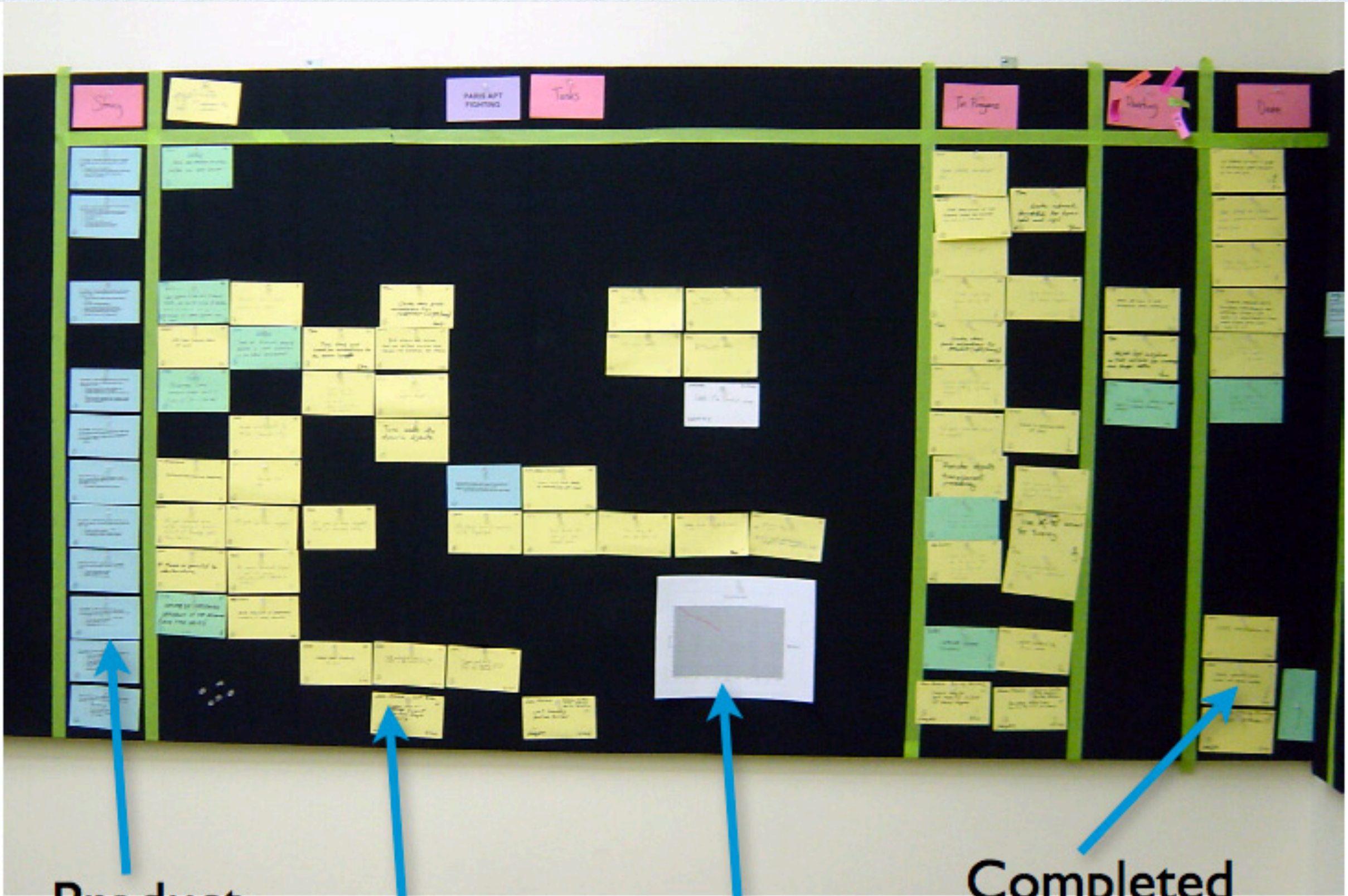
- An agile team of between 6 and 10 developers
- All members work with each others
- The team is co-located to enhance communication
- Described like passing a rugby ball effortlessly to achieve a goal
- The leader is the Scrum Master

Use a Scrumboard

- Work is managed and charted using a scrumboard
- Sticky notes are used to represent tasks
- Derivative of the Kanban JIT manufacturing process
- Items represent *product backlog*
- Online versions also available

Scrumboard

Story	To Do	In Process	To Verify	Done
As a user, I... 8 points	Code the... 9 Test the... 8 Code the... 2 Code the... 8 Test the... 8 Test the... 4	Code the... DC 4 Test the... SC 8	Test the... SC 6	Code the... D Test the... SC 8 Test the... SC Test the... SC Test the... SC 6
As a user, I... 5 points	Code the... 8 Test the... 8 Code the... 4 Code the... 6	Code the... DC 8		Test the... SC Test the... SC Test the... SC 6


PARIS APT FIGHTING
Tasks
Product backlog
Tasks to do
Burndown chart
Completed tasks

Scrum Sprints

- Sprint planning meeting - decide on the items in the product backlog to be developed during the sprint
- Focus exclusively on developing those features
- All members meet daily during a sprint - *stand up*
- No changes are made while the team is in a sprint
- At the end of a sprint is a *retrospective*

The Scrum Master

- Takes care of the team during the sprint
- Ensures the team is working at maximum efficiency
- Makes sure the team is not interrupted during a sprint

Stand Up Meetings

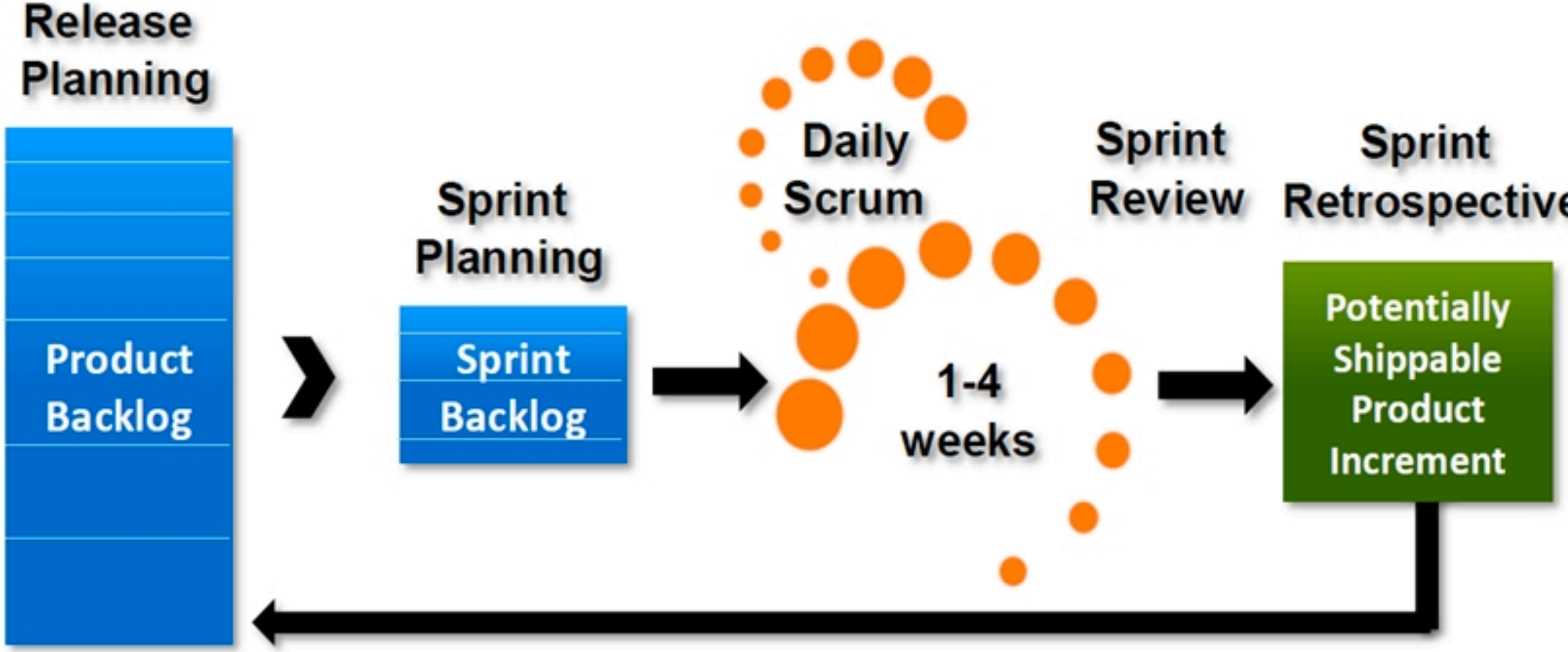
- Literally stand up in these meetings for 10-15 minutes every day
- Strict protocol for each meeting
- Only team members can speak
- A physical token is used to show who is allowed to speak
- Each member states:
  - what they did yesterday
  - what they will do today
  - anything that is impeding their progress

Why is this effective?

- Flexible management structure but with a high review frequency
- Allows for close working relationships which builds inherent trust in teams
- Allows developers to focus on what they do best which is developing software
- Decomposes work into short chunks
- Allows for rapid delivery

Combining Agile Practices

- Agile is very powerful if using sprints to manage the development process
- Used in conjunction with **pairs programming** and with **test driven development** as scrum does not dictate how the implementation is performed
- Most organisations 'mix and match' the different agile practices


Release Planning
Product Backlog
Sprint Planning
Sprint Backlog
Daily Scrum
1-4 weeks
Sprint Review
Sprint Retrospective
Potentially Shippable Product Increment

Test Driven Development

What is TDD?

- Concerned with the *unit testing, design and implementation* components of a project
- You write your test code **before** writing your functional code
- Tests designed to check very small bits of code
  - e.g. this loop functions correctly
  - e.g. **Not** this class functions correctly (too much to test!)

Kent Beck (2003)

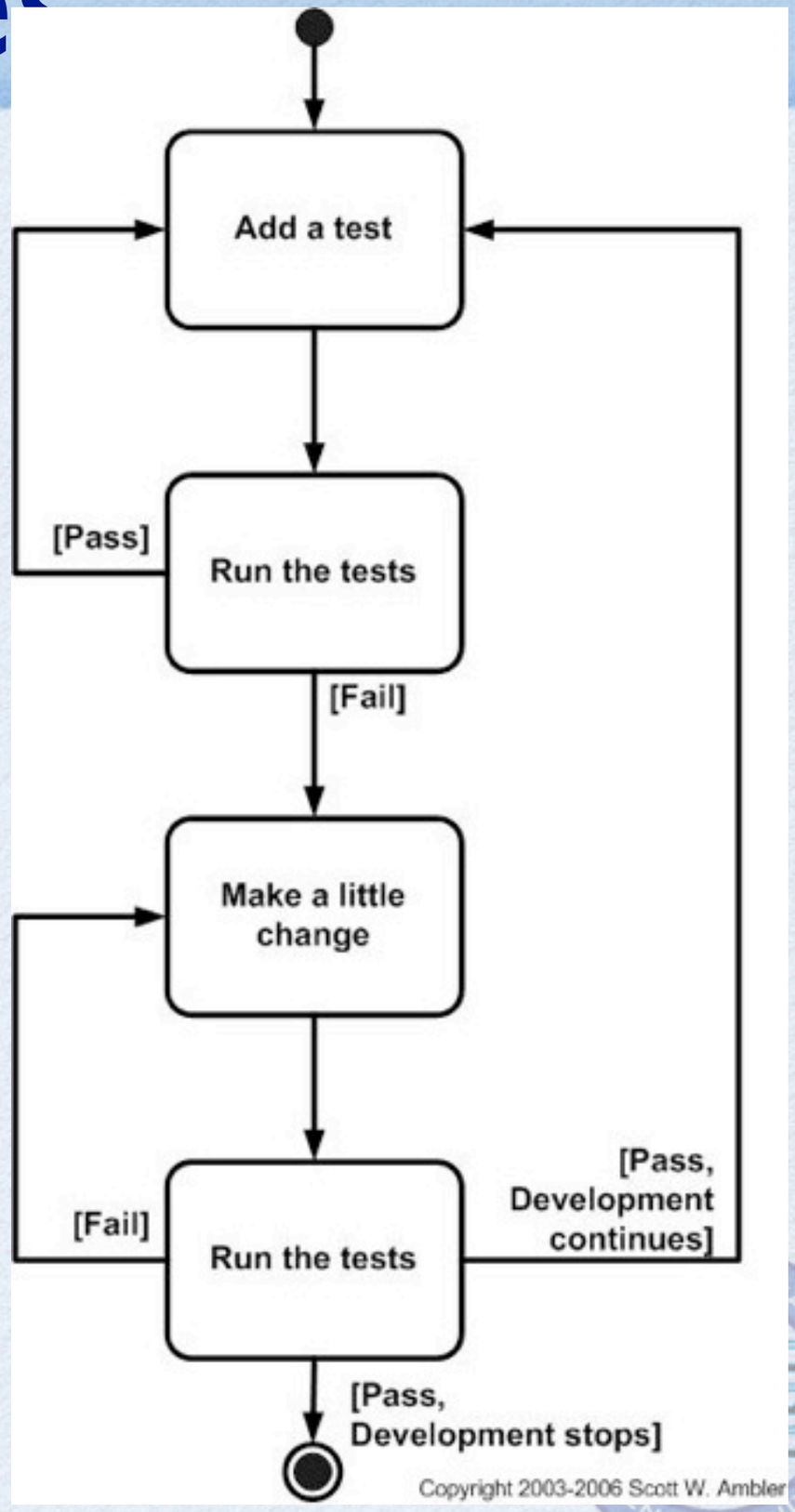
It requires a bit of Zen

- It can be tricky to do as you need to be disciplined
- Spend time writing test cases first from your design which can seem a bit laborious
- Have to avoid the temptation to rush ahead and just code
- Pairs programming in XP can help with this
  - one writes the code while the other writes and runs the tests

The Basic Principles

- TDD is a method which is based on *Test First Design*
- Once you pass the test then you must *refactor*

TDD = TFD +Refactoring



TDD in Contrast

- A test framework or test harness program is developed first before any code is written
  - there are different frameworks which already exist for most standard programming languages
- Allows for a more thorough code base to be developed than with standard unit testing
- Designed to run in tandem with other testing
  - this include system, acceptance and integration
  - With TDD every line of code should be tested, giving maximal testing coverage unlike a development first approach

Manifesto: Why Use TDD?

- Kent Beck says :

"If it is worth building, it is worth testing"

"If its not worth testing, why are you wasting time working on it?"

Spot the Difference

Design | Implement | Test

Traditional approach

Design | Test | Implement

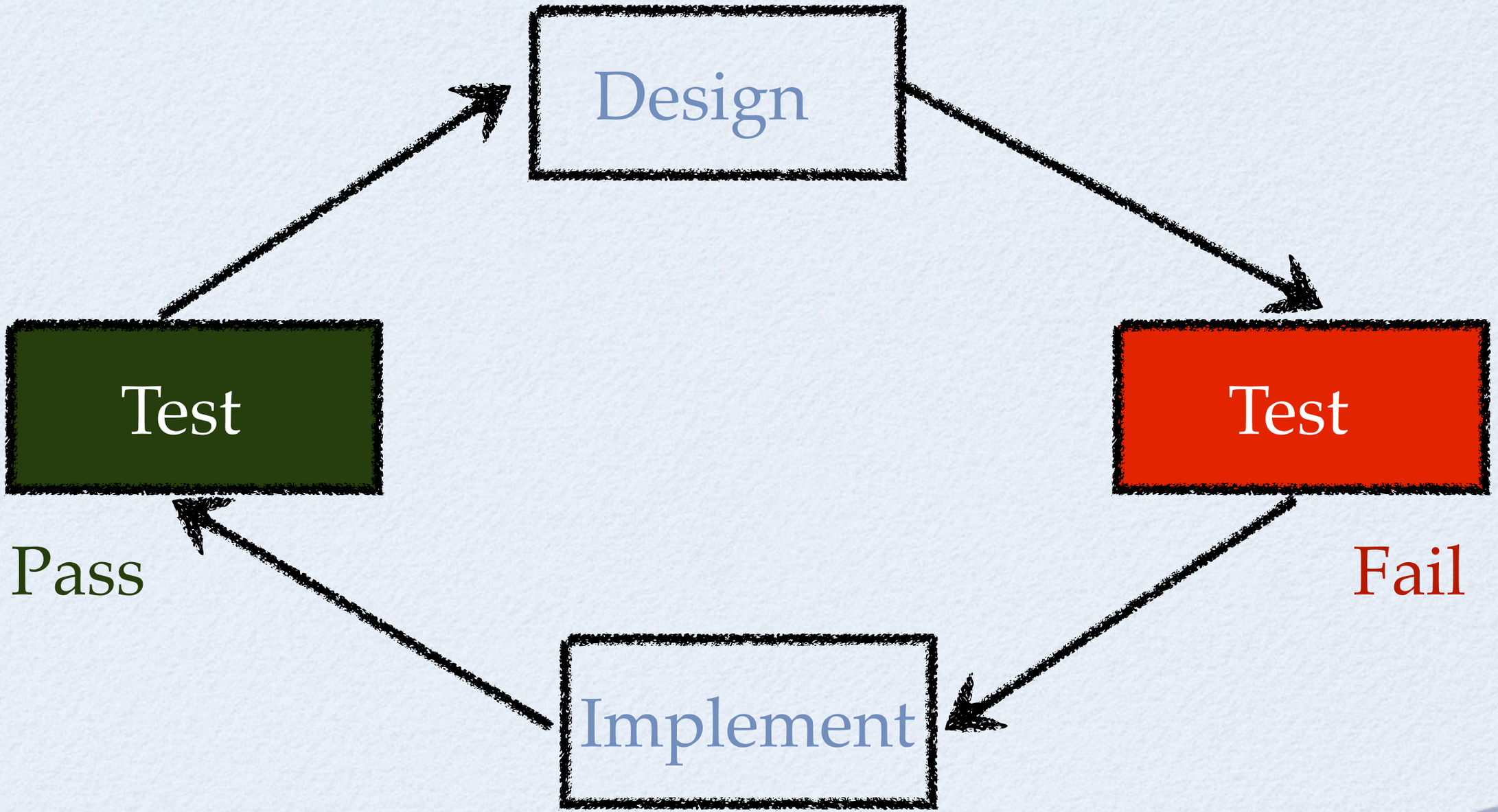
Test First Design

Design | Test | Implement | Test

TDD is Iterative

- *Design*: decide what you want to develop
- *Test:* write a test to express the design in *testing code or test code framework (not on paper!!)*
  - as you have not written the code yet it should *fail*
- *Implement:* write the code
- *Test:* repeat until you pass the test
  - go back to your design and write more tests, carry on the testing cycle until all tests are passed

Red Green Refactor


Design
Test
Test
Pass
Fail
Implement

How it actually works

- An addition method for a class in a calculator program
- Our design for this function is as follows:
  - "The method *add( )* takes two arguments and adds them together. The result is returned.
- *How do we use TDD to develop this simple function?*

Step 1: Write your test code

- Decide what you are testing and formalise it in test code
  - want to test that 2+2 returns a correct value of 4
- Write your test code in a testing framework
  - e.g. JUnit for Java or CUnit for C

```
use Test::More tests => 1;

is(add(2, 2), 4), ok;
```

(method (test case(s)), return value), passed = ok;

Step 2: Run the Test

- The first time you run this test it will fail
- If it does not fail then you are not doing TDD.....
- This is as you have not written any code yet
- This might seem a bit 'numpty' but there is a point to doing this.....

Step 3: Run the test for each line

- Proceed with your implementation until the test has passed
- For each line or component unit you add, run the test again to get maximum testing coverage

```
float add( float a, float b) {
   float sum;
   sum = a + b;
   return sum;
}
```

Step 4: Expand on tests

- From the design we are given there are many more tests that we can think of including:
  - what if we get the incorrect number of arguments?
  - what if the arguments are not numbers
  - how do we deal with NULL values?
- Repeat steps 2 and 3 until you can not think of any more tests
  - add new testing code to check for these things
  - repeat until all tests are passed

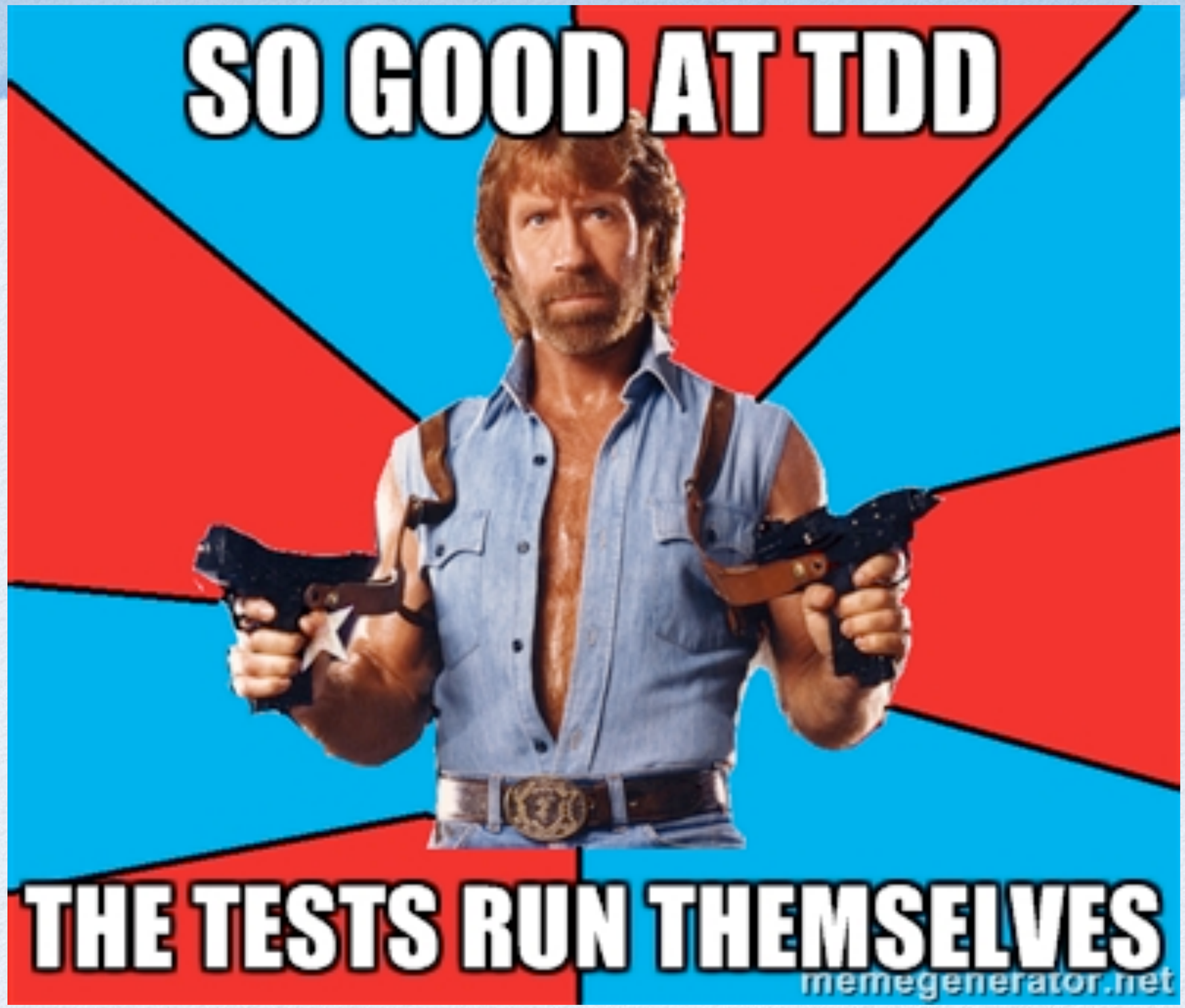
Refactoring

- Striving to develop code to pass tests can get messy
  - you're not really thinking ahead with your code
- This is where refactoring is used once you have all your tests "green"

- Refactoring:
  - "does not change the behaviour of the code [optimization] but makes it more readable, reduces redundancy and makes it easier to use in future
- *With every line of code you refactor do you know what you must do?*

Run your tests again
until the code passes!!
(thats the point!)

http://www.youtube.com/watch?v=uGaNkTahrIw


SO GOOD AT TDD
THE TESTS RUN THEMSELVES
memegenerator.net

Automation of the process

- If you were to do this testing with any kind of human input, analysis or intervention it would take forever
- Use an automated framework, many exist (*xUnit)*:
  - JUnit for Java
  - CUnit for C, cppUnit for C++
  - VBUnit for Visual Basic etc
- Kent Beck develops JUnit (http://www.junit.org)

JUnit Test Results

Statistics | Output

- test.net.sourceforge.pmd.cpd.MatchTest passed
- test.net.sourceforge.pmd.cpd.SourceCodeTest passed
- test.net.sourceforge.pmd.cpd.TokenEntryTest passed
- test.net.sourceforge.pmd.cpd.XMLRendererTest FAILED
  - test_no_dupes passed (0.01 s)
  - test_one_dupe FAILED (0.01 s)
    - expected:< code fragment ...> but was:< code fragment...>
    - junit.framework.ComparisonFailure
    - at test.net.sourceforge.pmd.cpd.XMLRendererTest.test_one_dupe(XMLRendererTest.java:68)
  - testRender_MultipleMatch passed (0.0 s)
- test.net.sourceforge.pmd.dfa.AcceptanceTest passed
- test.net.sourceforge.pmd.dfa.DAAPathFinderTest passed
- test.net.sourceforge.pmd.dfa.DataFlowNodeTest passed
- test.net.sourceforge.pmd.dfa.GeneralFiddlingTest passed
- test.net.sourceforge.pmd.dfa.StatementAndBraceFinderTest passed

Documentation and TDD

- Developers prefer to read code not documents
  - or even more so, prefer to write code and not documents
- Can use the tests as part of the documentation
  - replaces making implementation notes and huge comments blocks in files that no one can understand
- Can be used to replace implementation documentation provided that the tests are sufficiently detailed
- Does not replace all project documentation
  - still need user stories/requirements system architecture plans and support documentation

Advantages of TDD

- Development and unit testing are paired
  - reduces the overall development time and “technical debt”
- The unit testers and developers are the same people
  - reduces the manpower overheads
- Reduces the overall cognitive load on the developers
  - dont need to recall what you have done several stages later
- Produces code of higher quality
  - for every line of development you run your tests again
  - 100% of your code is tested == fewer bugs

TDD In Context

- TDD is there to enhance the testing process not to replace all forms of testing
- It is ideal for the more micro style of development
  - cannot replace acceptance, regression or system
- It is shown to minimise the amount of bugs which need to be fixed post release
- Also shown to reduce overall project costs
  - being adopted by forward thinking technology companies

Summary

- Scrum Agile
- Test Driven Development in Context
  - A recap of XP and Agile development
  - Test First Design
    - What it is and what it means in practice
- A TDD example
- Contrasting TDD with traditional development approaches
- The advantages of TDD in practice