University of Sheffield

COM3529 Software Testing and Analysis

Test Automation

Professor Phil McMinn

A Testing Problem

```
To: p.mcminn@sheffield.ac.uk
From: student3529@sheffield.ac.uk
Subject: A Problem with Testing – Please help!!!

Dear Phil

You asked last lecture whether we liked testing. To be honest, no I don’t, I find manually
trying out my software with inputs really dull!

Is this the right module for me? Please help!


Yours,
Stu
```

A Testing Solution

To: student3529@sheffield.ac.uk
From: p.mcminn@sheffield.ac.uk
Subject: Re: A Problem with Testing – Please help!!!

Dear Stu,

Have no fear.

I agree, manual testing is really dull! Automated testing however, is much more interesting, and is more like development. In fact, we should be writing tests while developing! Finding problems while developing is much more fun than finding them once the software is deployed. No more late nights spent debugging!

I’m going to be covering **writing automated tests** in the next lecture. Be sure to be there!

Best,
Phil

What do you understand by the phrase *automated test*?

JUnit example

```java
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class TriangleTest {

    @Test
    public void shouldClassifyEquilateral() {
        Triangle.Type result = Triangle.classify(10, 10, 10);
        assertEquals(Triangle.Type.EQUILATERAL, result);
    }

    // ...
```

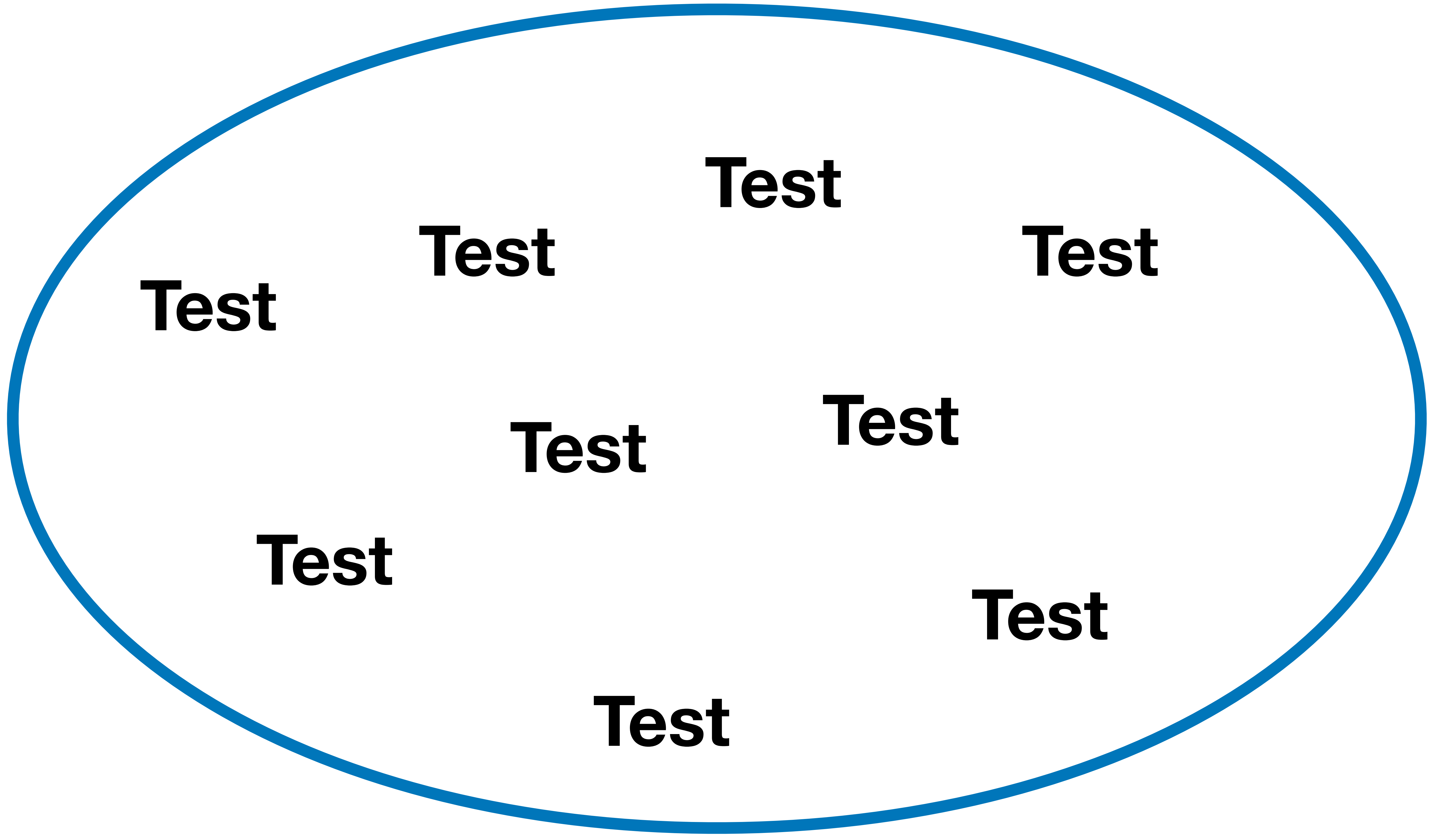
RSpec example

```ruby
require_relative "../spec_helper"

describe "the add page" do
  it "is accessible from the search page" do
    visit "/search"
    click_link "Add a new player to the database"
    expect(page).to have_content "Add Player"
  end

  it "will not add a player with no details" do
    visit "/add"
    click_button "Submit"
    expect(page).to have_content "Please correct the errors below"
  end

  it "adds a player when all details are entered" do
    add_test_player
    expect(page).to have_content "George Test"
    clear_database
  end
end
```

Ingredients of an Automated Test Case

1. The inputs needed to put the software into the right state for the test
2. The actual test case inputs
3. The expected results of the test
4. Reset of the system state

A Test Suite – A Set of Tests



Ideally, the tests can be executed in any order

The Dawn of Test Automation

Testing has always been part of programming

- … when you wrote your first program, you almost certainly tried it out with some sample data

- **For a long time, this was the state of the art in industrial practice!**

In the early 2000s, software development practices started to change

- Software systems got too big and too complex for manual testing to remain an effective and efficient way to ensure they were working and **remained working**

Testing at the Speed of Modern Software Development

Software systems are growing larger and evermore complex.

A typical application or service at Google, for example, is made up of thousands or millions of lines of code.

The ability for humans to **manually validate every behaviour in a system** has been **unable to keep pace with the explosion of features and platforms in most software**.

Testing at the Speed of Modern Software Development

Imagine what it would take to manually test the functionality of Google search – every time the code was changed.

> … not just web search, but images, flights, movie times etc.
>
> Then multiply that for every language, country, and device that must be supported.
>
> Then add in factors like accessibility and security.

Manual testing does not scale. We need automation.

Developer-Driven Automated Testing

The idea of coding automated tests (e.g., in JUnit) as a means of improving productivity and velocity may seem antithetical.

After all, the act of writing tests can take just as long (if not longer!) than implementing a feature in the first place … right?

On the contrary!

> In industry, investing in software tests provides several key benefits to developer productivity.

Less Debugging

Tested code has fewer defects when it is submitted.

Crucially, it also has fewer defects throughout its existence – since code tends to be updated during its lifetime.

> … it will be changed by other teams and even automated code maintenance systems.
>
> Changes to code, or its dependencies, can be quickly detected by an automated test and rolled back before the problem reaches production.

Increased Confidence in Changes

Projects with good tests can be modified with confidence since all the important behaviours of their projects are continuously being verified.

These projects *encourage* refactoring.

After a change, we can re-run the automated tests to ensure we didn’t break any of the existing functionality.

Improved Documentation

Software documentation is notoriously unreliable!

Clear, focused tests that exercise one behaviour at a time function as executable documentation.

Thoughtful Design

Writing tests for new code is a practical means of exercising the API design of the code itself.

If new code is difficult to test, it is often because the code being tested has too many responsibilities or difficult-to-manage dependencies.

Well-designed code should be modular, avoiding tight coupling and focusing on specific responsibilities.

Fixing design issues early means less rework later.

Fast, High Quality Releases

With a healthy automated test suite, teams can release new versions of their application with confidence.

Many large projects, involving hundreds of engineers and thousands of code changes submitted every day, involve very short release cycles – often every day.

This would not be possible without automated testing.

Benefits of an Automated Test Suite

1. Less Debugging
2. Increased Confidence in Changes
3. Improved Documentation
4. Thoughtful Design
5. Allows for Fast, High Quality Software Releases