Introduction to Software Testing

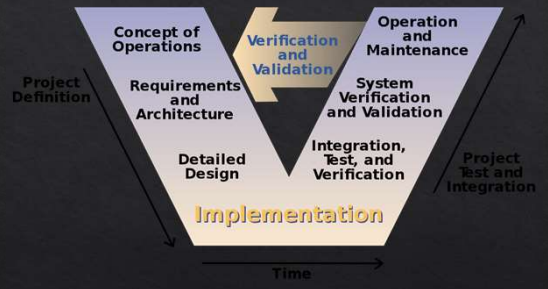
1

Agenda

1. Types of Testing
2. Unit Testing
3. Principles
4. Test Driven Development (TDD)
    1. xUnit Test Architecture
    2. Three laws of TDD
    3. Test first vs test last
5. Hands on Lab
6. Review

2

V Model



Wikipedia

3

Test Types

Unit test	Functional test	White box	Black box	Acceptance tests
Verification tests	Validation tests	Regression tests	Stress tests	Usability tests
Load tests	Performance tests	Integration tests	Smoke tests	

4

Linters

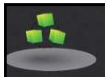
5

Linter

Locates and optionally fixes syntactical and stylistic errors defined in your coding standard

- JSLint
- JSHint
- ESLint (Most popular)
- JSCS

6



VS Code: Install and Configure ESLint

- Great example of ESLint installation for VSCode
- https://www.youtube.com/watch?v=01aO470ninM
- Suggest using AirBnB configuration

7

Unit Testing

8

9

Define: Unit Testing

...is a short program fragment written and maintained by the developers on the product team, which exercises some **narrow** part of the product's source code and checks the results. The outcome of a unit test is binary: either "pass" if the program's behaviour is consistent with the recorded expectations, or "fail" otherwise. Developers will typically write many unit tests (corresponding to many program behaviours of interest), called a "test suite". By common convention dating back at least to the JUnit family of tools, the colour red (as in "getting a red bar") represents the failure of one or more tests. The colour green ("a green bar") denotes successful execution of "all" unit tests associated with a program.

(Agile Alliance – Agile Glossary)

10

History: Unit Testing

1997: the testing tool **JUnit** is written by Beck and Gamma, inspired by Beck's earlier work on sUnit; its growing popularity over the next few years marks the end of the "capture and replay" era

(Agile Alliance – Agile Glossary)

sUnit – Smalltalk

jUnit - Java

11

Principles

12

Rule of 1

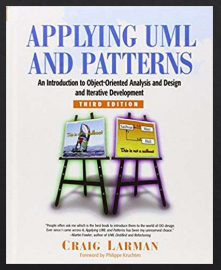
1 Component

1 Role

1 File

13

GRASP



14

GRASP – The Questions They Answer

Creator	Who should create these objects?
Controller	What first object beyond the UI controls the system (MVC)
Low Coupling	How to reduce the impact of change
High Cohesion	How to keep objects focused, understandable and manageable

15

GRASP – The Questions They Answer

Protected Variation	How to assign responsibilities to elements so that variation / instability in these elements do not have an undesirable impact on other elements
Indirection	How to assign responsibilities to avoid direct coupling
Polymorphism	Who is responsible when behaviour varies by type

16

GRASP – The Questions They Answer

Information Expert	Who has the knowledge to become responsible
Pure Fabrication	Who is responsible when you are desperate

17

18

High Cohesion

Measures how functionally related classes and methods are and how much work a class is doing. In other words how focused the class is

- ***Problem***: Fuzzy class responsibilities
- ***Solution***: Increase cohesion. Break into and subclasses
- ***Benefits***: Makes code easier to read
  Easier to change
  Easier to refactor
  Easier to bring under test
- ***Example***: Model View Controller
- ***Caveat***: Performance in distributed systems

19

20

Low Coupling

Measures how strong one class is **connected** to, has **knowledge** of, or **depends** on other classes

***Problem*:** System resistant to change
Difficult to re-use

***Solution*:** Reduce coupling

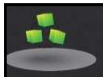
***Benefits*:** Lower dependencies
Easier to change & refactor
Easier to bring under test
Low impact when changing other classes
Higher reuse potential

21

SOLID Principles

Single Responsibility Principle
Open/Closed Principle
Liskov's Substitution Principle
Interface Segregation Principle
Dependency Inversion Principle

22



What is SRP?

23

SRP - Single Responsibility Principle

SRP is a good place to start with refactoring

A class should exist for one and only one reason. It should do one thing and do it well

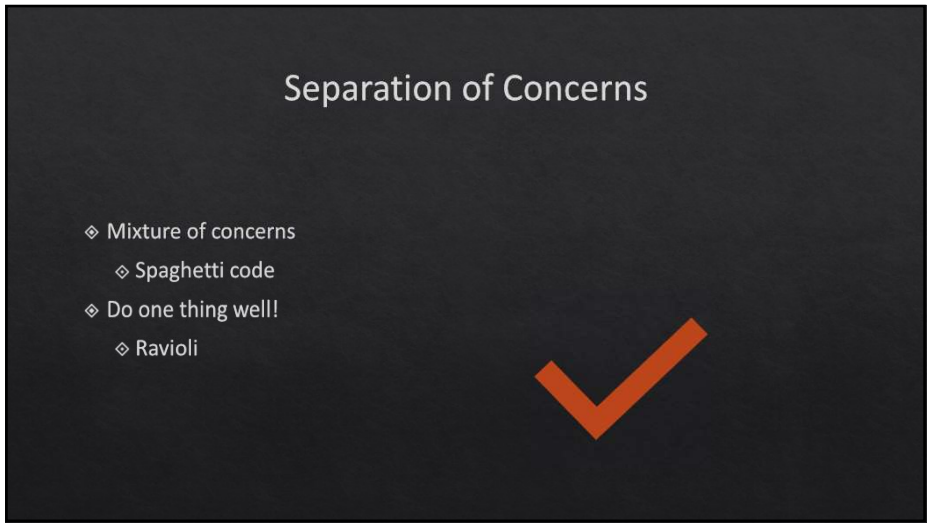
Separation of concerns

Bad smell: overcomplex

An example would be moving utility code out into another class

- Logging code
- Streaming
- Storage management

24


Separation of Concerns
Mixture of concerns
Spaghetti code
Do one thing well!
Ravioli

25