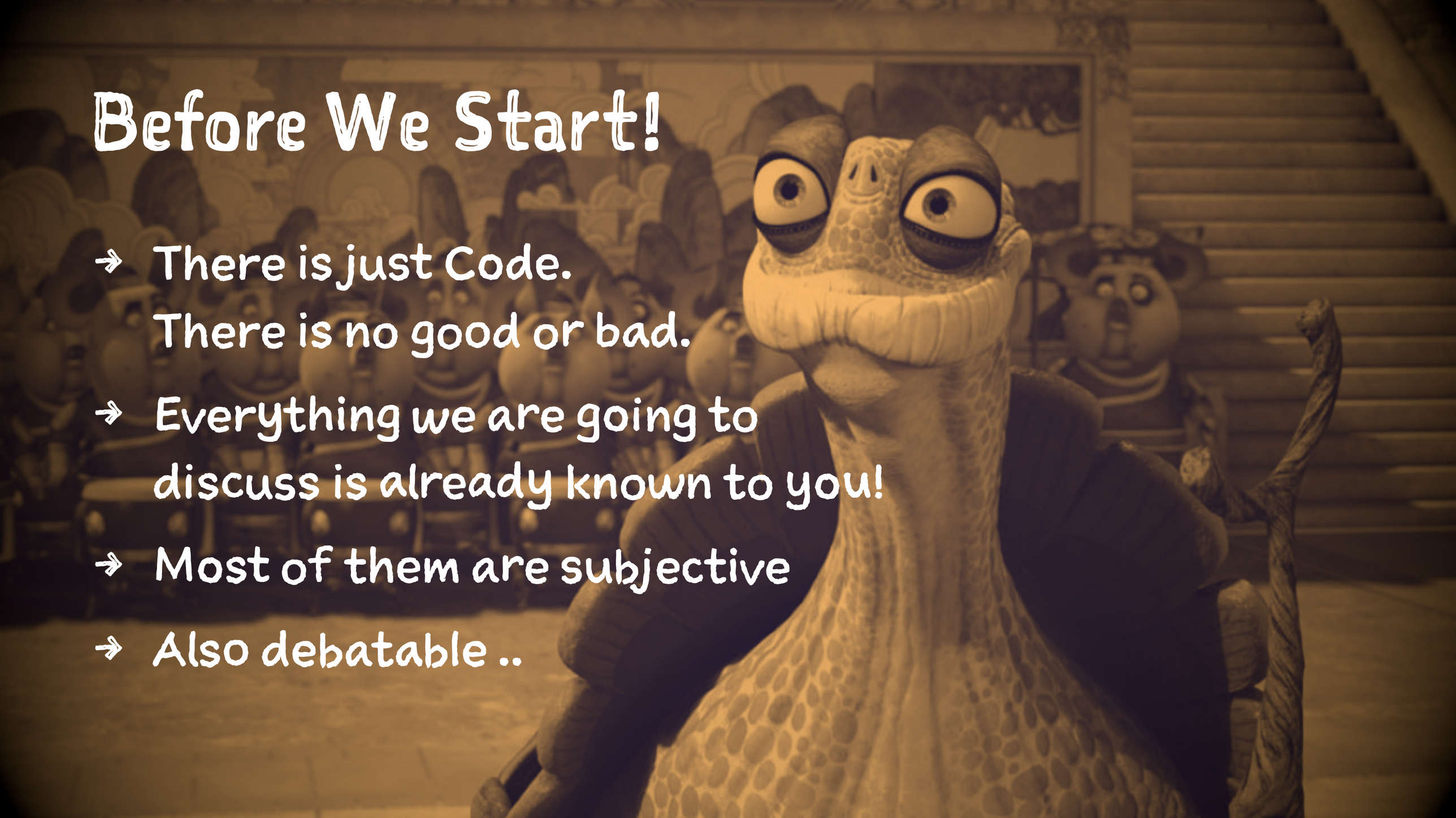
Test Driven Development

by Satya Sudheer



Before We Start!

- There is just Code.
  There is no good or bad.
- Everything we are going to discuss is already known to you!
- Most of them are subjective
- Also debatable ..

Agenda

- Basics
- Test Driven Development
- Coding Kata
- Q&A

Unit Test

A unit test is a piece of code that invokes a unit of work and checks one specific end result of that unit of work. If the assumptions on the end result turn out to be wrong, the unit test has failed. A unit test's scope can span as little as a method or as much as multiple classes

A unit is a method or function.

Why do we need Unit Tests?

- To find "Bugs" quickly.
- To produce "Maintainable Code".

How many unit test do you write?

```
public bool IsLoginSuccessful(string user, string password)
{
// ...
}
```

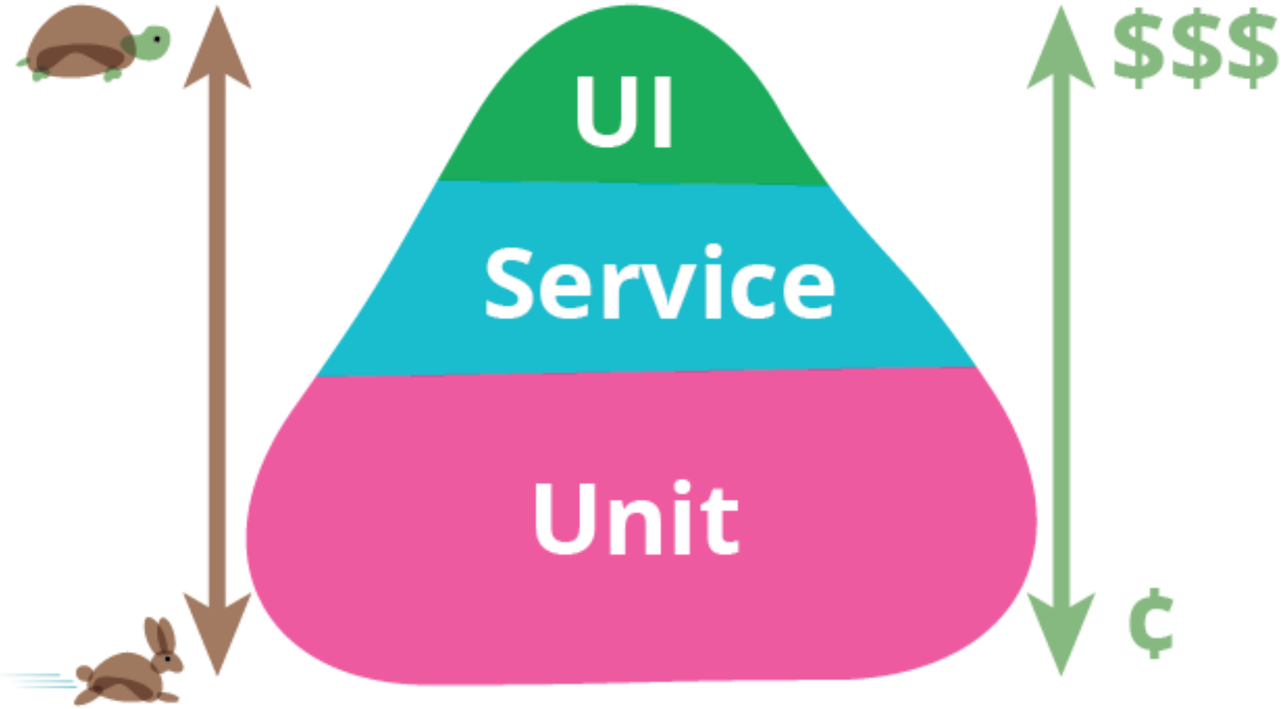
Unit tests isolate the unit of work from its real dependencies, such as time, network, database, threads, random logic, and so on.

Unit Tests runs in memory & quick!

What is an "Integration Test"?

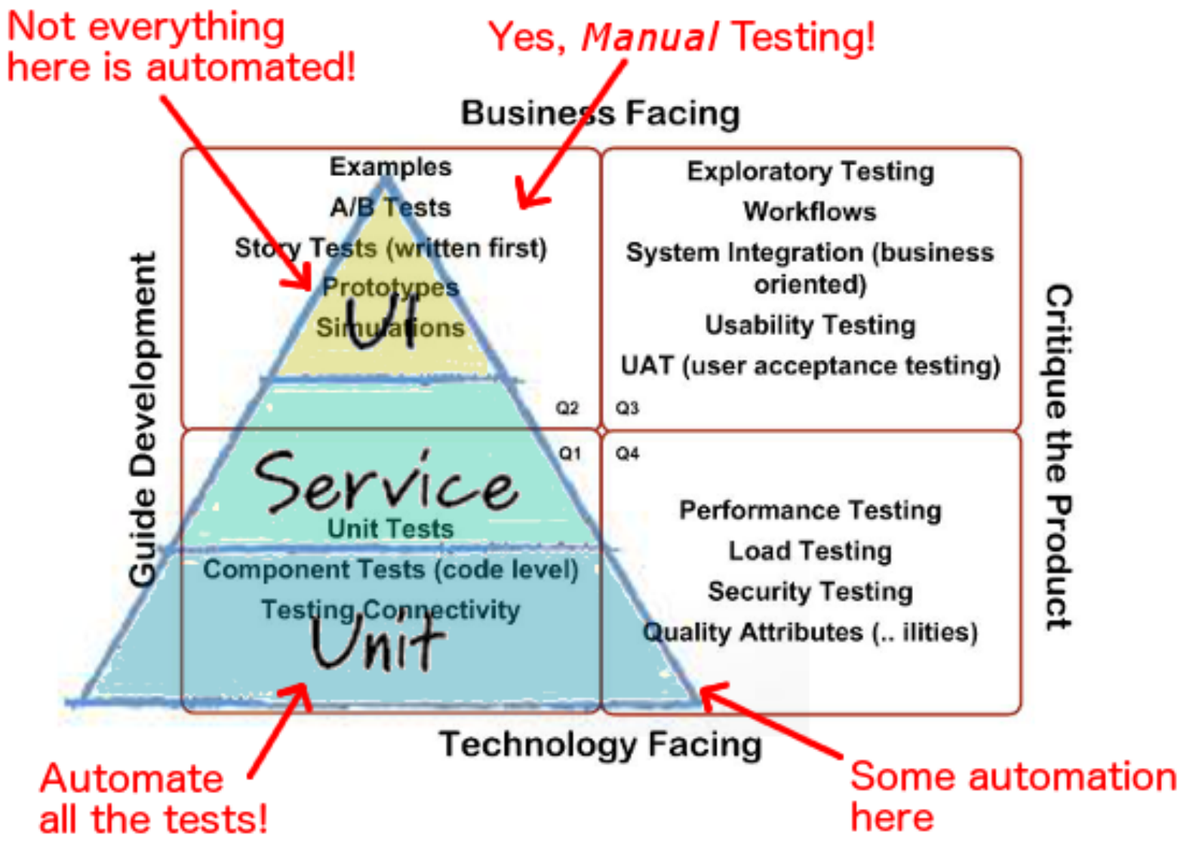
Integration testing is testing a unit of work without having full control over all of it and using one or more of its real dependencies, such as time, network, database, threads, random logic, and so on.

Integration tests are usually much slower!



Agile Testing Quadrants

- Quadrant 1: Technology-facing tests that support the team
- Quadrant 2: Business-facing tests that support the team
- Quadrant 3: Business-facing tests that critique the product
- Quadrant 4: Technology-facing tests that critique the product



Why do we need Unit Tests?

[Revisting]

To find "Bugs" quickly.

What if your "Unit Test" has bugs!

Test Driven Development

Red - Green - Refactor

Test-driven development (TDD), is a rapid cycle of testing, coding, and refactoring.

TDD Cycle

Red - Green - Refactor

- Write a fialing test for the next bit of functionality you want to add.
- Write the functional code until the test passes.
- Refactor both new and old code to make it well structured.

Prepare to Experince Awesomeness!

FizzBuzz Kata (TDD)

FizzBuzz

- Write a program that prints the numbers from 1 to 100.
  - For multiples of three print “Fizz” instead of the number
  - For the multiples of five print “Buzz”.
  - For numbers which are multiples of both three and five print “FizzBuzz”.

Noodle Break, Not Exactly! Git Help

→ Fork & Getting Started

```
git clone <url>
cd kata
```

→ Sync with remote repo

```
git pull origin master
```

→ Adding files:

```
git add .
git commit -m "appropriate message"
git push origin master
```

"Unit Test" Frameworks

- To Write Tests: Provides you an easy way to create & organize Tests.
- To Run Tests: Allows you to run all, or a group of tests or a single test.
- Feedback & Intergration: Immediate "Pass"/"Fail" feedback.

Examples: MSUnit, JUnit, etc..

TDD: Step 1

Write a "Failing Test"

How to Write Super Awesome "Unit Tests"

- Name - 3 Parts
  - Unit Of Work
  - Scenario
  - Expected behaviour
- Structure - 3 "A"s
  - Arrange
  - Act
  - Assert

Few More!

- No Logic in "Unit Tests"
- One Assert per Test.
- Should Run Fast
- Independent
- Repeatable
- Immutable

TDD: Step 2

Just code enough to "Pass the Test"

Super Simple "Tip"

- Take Baby Steps:
  - Rule 1: The next step to take should always be as small as possible.
  - Rule 2: Read "Rule 1" - Yes, as small as possible.

Why Baby Steps?

We will produce well-designed, well-tested, and well-factored code in small, verifiable baby steps.

TDD: Step 3

Improve the code by "Refactoring".

Oh, Yeah Refactoring

- No "Logic Changes", Just moving the code around!
- Explore performant code options.
- Keep watching your "Unit Tests"
- Remember "Baby Steps" Rule.

Refactoring is Important!

The most common way that I hear to screw up TDD is neglecting the third step. Refactoring the code to keep it clean is a key part of the process, otherwise you just end up with a messy aggregation of code fragments.

In Short

Unit Tests & Integration Tests are as important as production code because they allow you to make changes confidently and correctly, improving your flexibility and maintainability over time.

Questions

(if any)

The basic steps of TDD are easy to learn, but the mindset takes a while to sink in.

Until it does, TDD will likely seem clumsy, slow, and awkward. Give yourself two or three months of full-time TDD use to adjust.


Coding Dojo
To Become Awsome Developer

How to make sure, your code is understandable, readable? Clean Code!

Next Session ..

Thank You :)