Deliver OUTCOME, early and often

Unit Testing

with Test-First Development and Test-Driven Development in Action Workshop

Deliver experiences by Siam Chamnankit Company Limited

Year 2025

Circular Buffer

© 2013 - 2023 Siam Chamnankit Company Limited

Tennis

Tennis has a rather quirky scoring system, and to newcomers it can be a little difficult to keep track of. The tennis society has contracted you to build a scoreboard to display the current score during tennis games.

You can read more about Tennis scores on wikipedia which is summarized below:

1. A game is won by the first player to have won at least four points in total and at least two points more than the opponent.
2. The running score of each game is described in a manner peculiar to tennis: scores from zero to three points are described as “Love”, “Fifteen”, “Thirty”, and “Forty” respectively.
3. If at least three points have been scored by each player, and the scores are equal, the score is “Deuce”.
4. If at least three points have been scored by each side and a player has one more point than his opponent, the score of the game is “Advantage” for the player in the lead.

You need only report the score for the current game. Sets and Matches are out of scope.

https://en.wikipedia.org/wiki/Circular_buffer

© 2013 - 2022 Siam Chamnankit Company Limited

TDD

© 2013 - 2023 Siam Chamnankit Company Limited

What is TDD?

Test-driven development (TDD) is a software development process relying on software requirements being converted to test cases before software is fully developed, and tracking all software development by repeatedly testing the software against all test cases. This is as opposed to software being developed first and test cases created later.

Test-driven development is related to the test-first programming concepts of extreme programming, begun in 1999, but more recently has created more general interest in its own right.

https://en.wikipedia.org/wiki/Test-driven_development

What is TDD?

Test-Driven Development (TDD) is a methodology in software development that focuses on an iterative development cycle where the emphasis is placed on writing test cases before the actual feature or function is written.

TDD **utilizes repetition of short development cycles**. It combines building and testing. This process not only helps **ensure correctness of the code** -- but also helps **to indirectly evolve the design and architecture of the project** at hand.

Kent Beck, Test-Driven Development By Example, Addison-Wesley, 2002

© 2013 - 2022 Siam Chamnankit Company Limited

Red-Green-Refactor

© 2013 - 2023 Siam Chamnankit Company Limited

Red-Green-Refactor cycle

(Red)	Add a test to the test suite. Run all the tests to ensure the new test fails
(Green)	Write just enough code to get that single test to pass. Run all tests
(Refactor)	Improve the initial code while keeping the tests green.

Repeat

© 2013 - 2022 Siam Chamnankit Company Limited

FizzBuzz

© 2013 - 2023 Siam Chamnankit Company Limited

FizzBuzz

Fizz buzz is a group word game for children to teach them about division. Players take turns to count incrementally, replacing any number divisible by three with the word "fizz", and any number divisible by five with the word "buzz", and any number divisible by both 3 and 5 with the word "fizzbuzz"

https://en.wikipedia.org/wiki/Fizz_buzz

xUnit

© 2013 - 2023 Siam Chamnankit Company Limited

xUnit: Assertions

- Equal
- NotEqual
- NotSame
- Same
- Contains
- DoesNotContain
- InRange
- NotInRange
- Empty
- NotEmpty
- True
- False
- IsAssignableFrom
- IsType<T>
- IsNotType<T>
- NotNull
- Null
- Throws<T>

https://xunit.net/docs/comparisons

© 2013 - 2022 Siam Chamnankit Company Limited

xUnit

- Unit Test Report
  - HTML
  - JUnit
- Code Coverage Report
  - HTML
  - Cobertura

- Parameterize
  - Theory and InlineData
- Capturing Output
- Life Cycle
  - Constructor and Dispose
  - Class Fixture
  - Collection Fixtures

F.I.R.S.T

© 2013 - 2023 Siam Chamnankit Company Limited

F.I.R.S.T

Fast: The faster your tests run, the more often you’ll run them.
Isolated: Each unit test should have a single reason to fail. You must design your tests to be independent not only of external factors but of each other as well.
Repeatable: You should obtain the same results every time you run a test.
Self-Verifying: A good unit test fails or passes unambiguously.
Timely: You should always know what you’re trying to build before you build it. Tests written first specify the behavior that you’re about to build into the code.

https://medium.com/pragmatic-programmers/unit-tests-are-first

© 2013 - 2022 Siam Chamnankit Company Limited

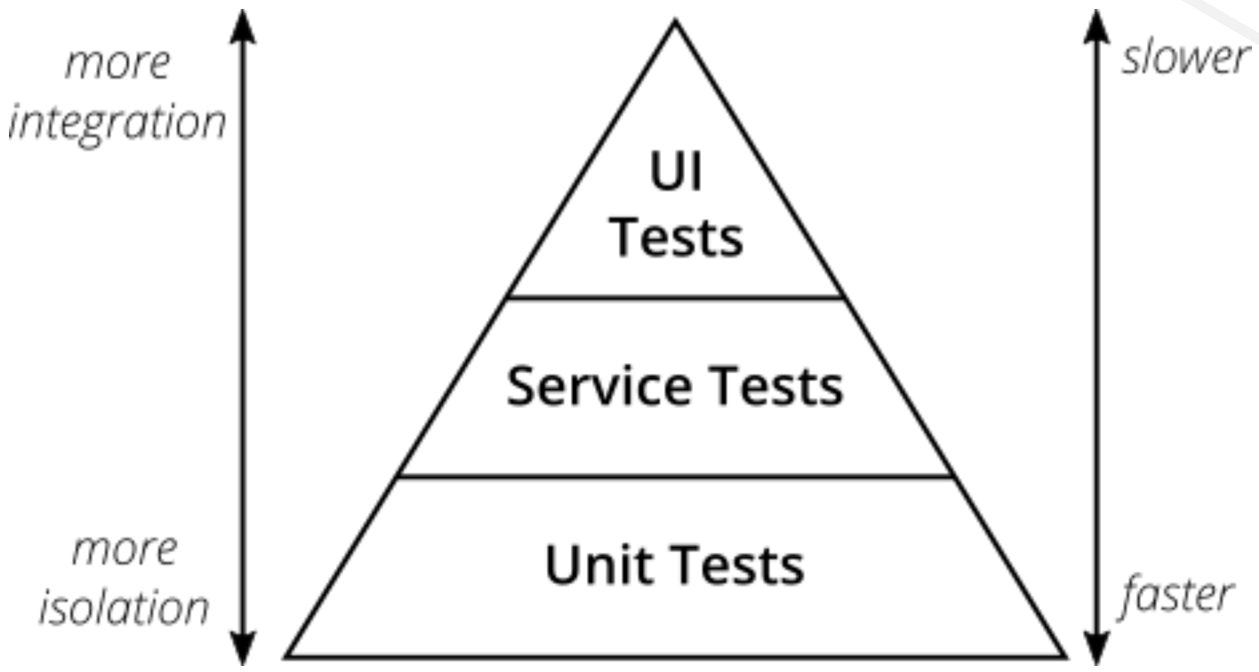
The Test Pyramid

© 2013 - 2023 Siam Chamnankit Company Limited

The Test Pyramid

Mike Cohn's original test pyramid consists of three layers that your test suite should consist of (bottom to top):

1. Unit Tests
2. Service Tests
3. User Interface Tests



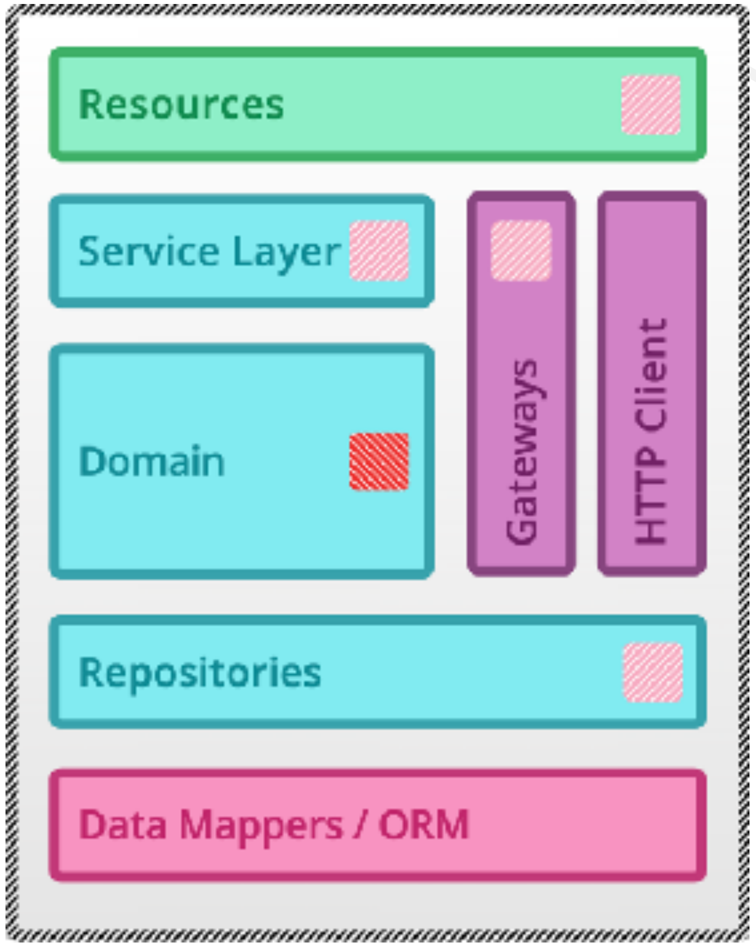
https://martinfowler.com/articles/practical-test-pyramid.html

Unit Test in Microservice Testing

© 2013 - 2023 Siam Chamnankit Company Limited

Unit Test in Microservice Testing

The size of the unit under test is not strictly defined, however unit tests are typically written at the class level or around a small group of related classes.



https://martinfowler.com/articles/microservice-testing/

Test Double

© 2013 - 2023 Siam Chamnankit Company Limited

Test Double

- **Dummy**: Some method signatures of the SUT may require objects as parameters. If neither the test nor the SUT care about these objects, we may choose to pass in a Dummy Object which may be as simple as a null object reference, an instance of the Object class or an instance of a Pseudo Object.
- **Stub**: We use a Test Stub **to replace a real component** on which the SUT depends so that the test has a control point for the **indirect inputs** of the SUT. This allows the test to force the SUT down paths it might not otherwise execute
- **Mock**: We can use a Mock Object as an observation point that is used **to verify** the **indirect outputs** of the SUT as it is exercised. Typically, the Mock Object also includes the functionality of a Test Stub.
- **Spy**: We can use **a more capable version of a Test Stub**, the Test Spy, as an observation point for the **indirect outputs** of the SUT. Like a Test Stub, the Test Spy may need **to provide values to the SUT in response to method calls** but the Test Spy also **captures the indirect outputs** of the SUT as it is exercised and saves them **for later verification** by the test.
- **Fake**: We use a Fake Object **to replace the functionality of a real DOC** in a test for reasons other than verification of indirect inputs and outputs of the SUT. Typically, **it implements the same functionality as the real DOC** but in a much simpler way. While a Fake Object is typically built specifically for testing, it is not used as either a control point or a observation point by the test.

http://xunitpatterns.com/Test%20Double.html

Moq

© 2013 - 2023 Siam Chamnankit Company Limited

Moq

- Method

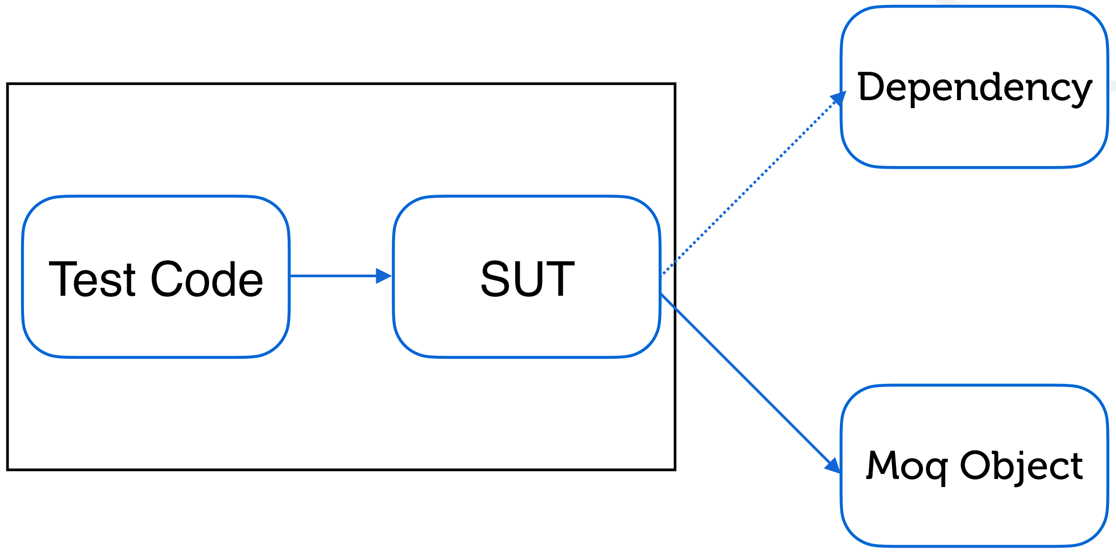
```
Mock<IT> mock = new Mock<IT>();
mock.Setup(t => t.Method(params)).Returns(Value);
```

- Matching Arguments

```
mock.Setup(t => t.Method(IsAny<Type>)).Returns(Value);
```

- Verification

```
Mock<IT> mock = new Mock<IT>();
mock.Verify(t => t.Method(IsAny<Type>))
```



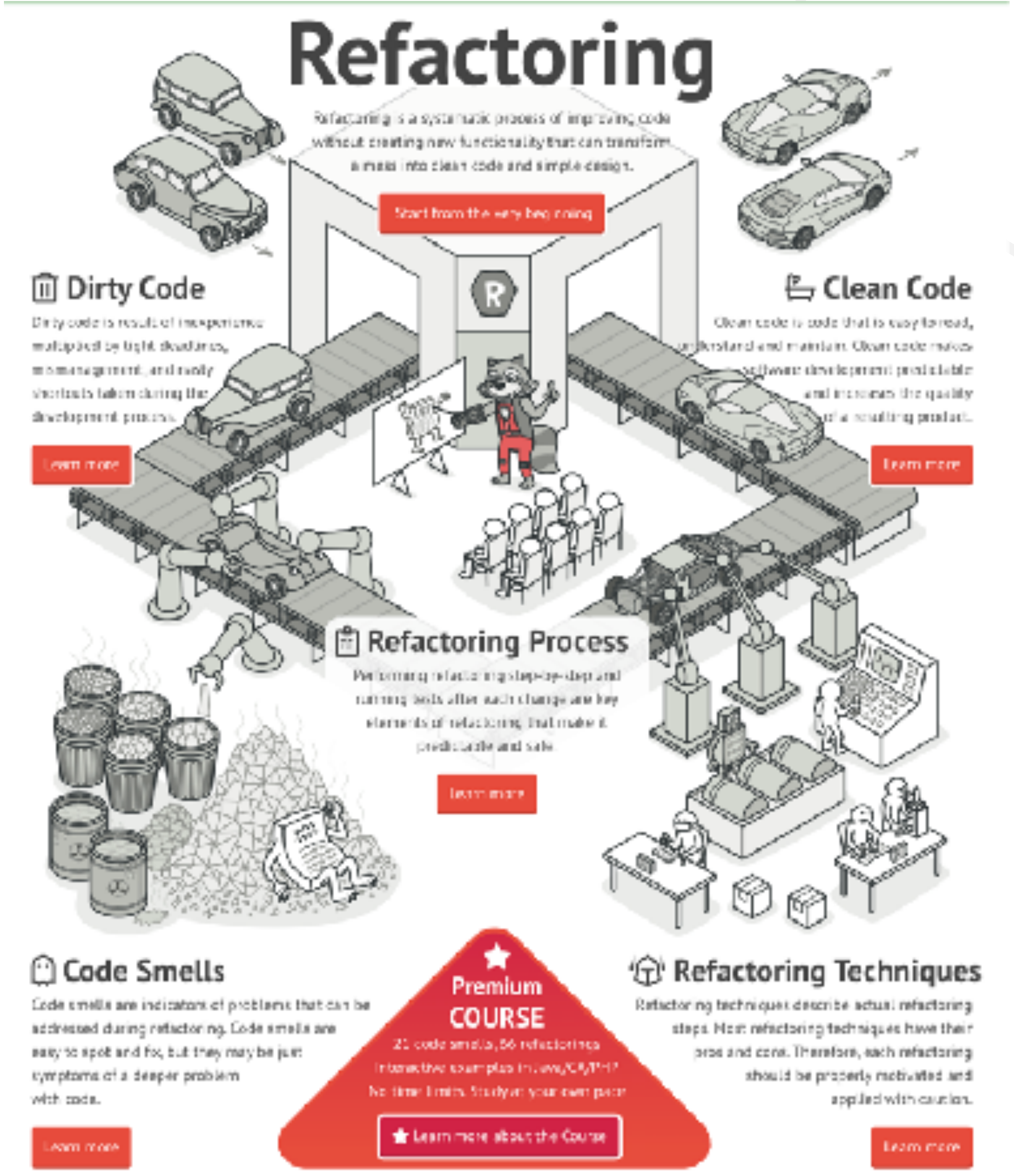
https://github.com/Moq/moq4/wiki/Quickstart

Refactoring

© 2013 - 2023 Siam Chamnankit Company Limited

Refactoring

- [refactoring.com]()
- [Code Smells]()
- [Code Smells Cheatsheet]()
- [Refactoring Kata]()



Refactoring

Refactoring is a systematic process of improving code without creating new functionality that can transform a mess into clean code and simple design.

Start from the very beginning

Dirty Code

Dirty code is result of inexperience multiplied by tight deadlines, mismanagement, and nasty shortcuts taken during the development process.

Learn more

Clean Code

Clean code is code that is easy to read, understand and maintain. Clean code makes software development predictable and increases the quality of a resulting product.

Learn more

Refactoring Process

Performing refactoring step-by-step and running tests after each change are key elements of refactoring that make it predictable and safe.

Learn more

Code Smells

Code smells are indicators of problems that can be addressed during refactoring. Code smells are easy to spot and fix, but they may be just symptoms of a deeper problem with code.

Learn more

Premium COURSE

21 code smells, 66 refactorings

Interactive examples in Java/C#/PHP

No time limits. Study at your own pace

Learn more about the Course

Refactoring Techniques

Refactoring techniques describe actual refactoring steps. Most refactoring techniques have their pros and cons. Therefore, each refactoring should be properly motivated and applied with caution.

Learn more

ไม่ว่าแง่มุมไหนที่ได้จาก Workshop นี้

จะเหมือนหรือแตกต่างกันอย่างไร

สำคัญที่สุดคือ การนำไปปรับใช้

และ ส่งต่อประสบการณ์จากการลงมือทำ

จาก เรา สู่ เพื่อนพ้องน้องพี่

เพราะ การแบ่งปันก็คือความใส่ใจ

© 2013 - 2023 Siam Chamnankit Company Limited

Thank You