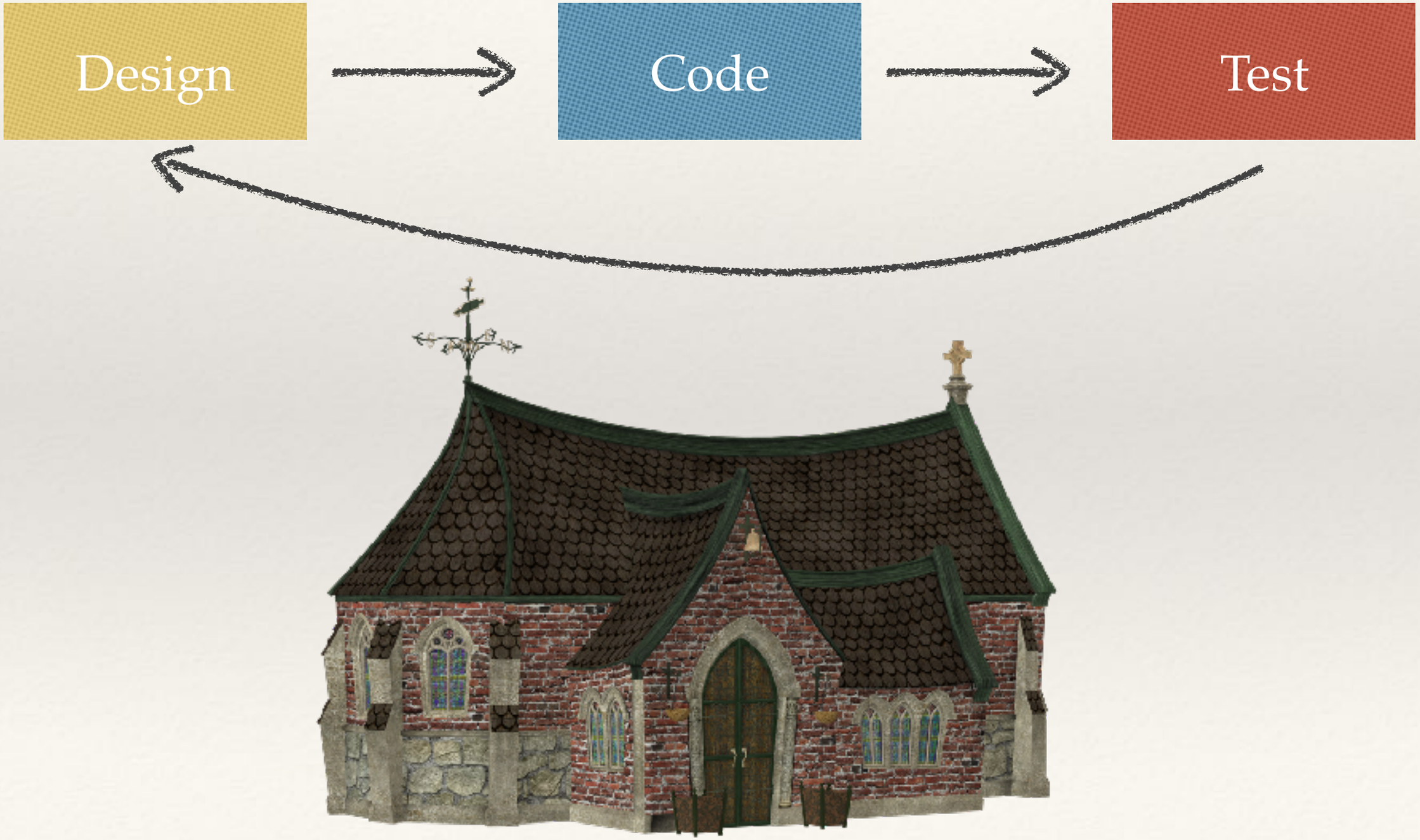
CodeOps Technologies

An Overview of Test Driven Development

Ganesh Samarthyam
ganesh@codeops.tech
www.codeops.tech

Old school approach



New school approach

What is TDD?

- Test-Driven Development (TDD) is a technique for building software that guides software development by writing tests. (Martin Fowler's definition)
- "**Test** Driven **Development**" is NOT primarily about testing or development (i.e., coding)
  - It is rather about design - where design is evolved through refactoring
- Developers write unit tests (NOT testers) and *then* code

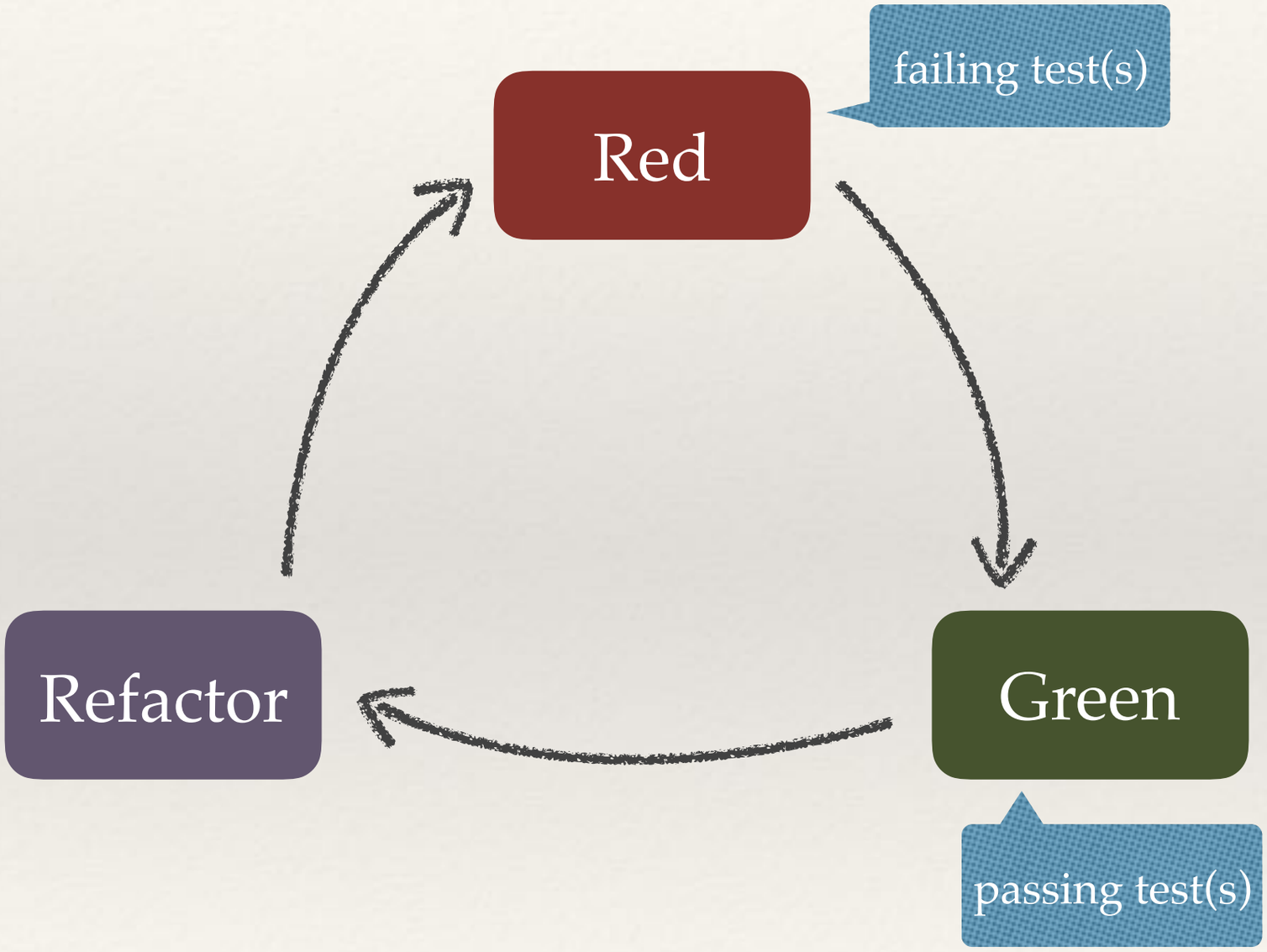
TDD? But I already write unit tests!

- In Test Driven Development, tests mean "unit tests" (Note that in variations/extensions such as ATDD, tests are not unit tests)
- Just because unit tests are written in a team doesn't mean they follow TDD - they could be written even after the writing code - that is Plain Old Unit testing (POUting)
- However, when following Test Driven Development, or Test *First* Development approach, we write the tests first before actually writing the code

Writing tests after code: disadvantages

- There are many disadvantages in writing tests after code
  - Testing does not give direct feedback to design and programming => in TDD, the feedback is directly fed back into improving design and programs
  - Most of the times, after realising the functionality in code, unit testing is omitted => TDD inverts this sequence and helps create unit tests first
  - Writing tests after developing code often results in "happy path" testing => we don't have enough granular or "testable" code segments to write the tests

TDD mantra



TDD mantra

- Red—write a little test that doesn't work, perhaps doesn't even compile at first
- Green—make the test work quickly, committing whatever sins necessary in the process
- Refactor—eliminate all the duplication and smells created in just getting the test to work

Source: Test Driven Development: By Example, Kent Beck, 240 pages, Addison-Wesley Professional, 2002;

Best Practice

Make it green, then make it clean!

3 laws of TDD

- *Law 1:* You may not write production code unless you've first written a failing unit test.
- *Law 2:* You may not write more of a unit test than is sufficient to fail.
- *Law 3:* You may not write more production code than is sufficient to make the failing unit test pass.

Source: Professionalism and Test-Driven Development, Robert C. Martin, IEEE Software, 2007

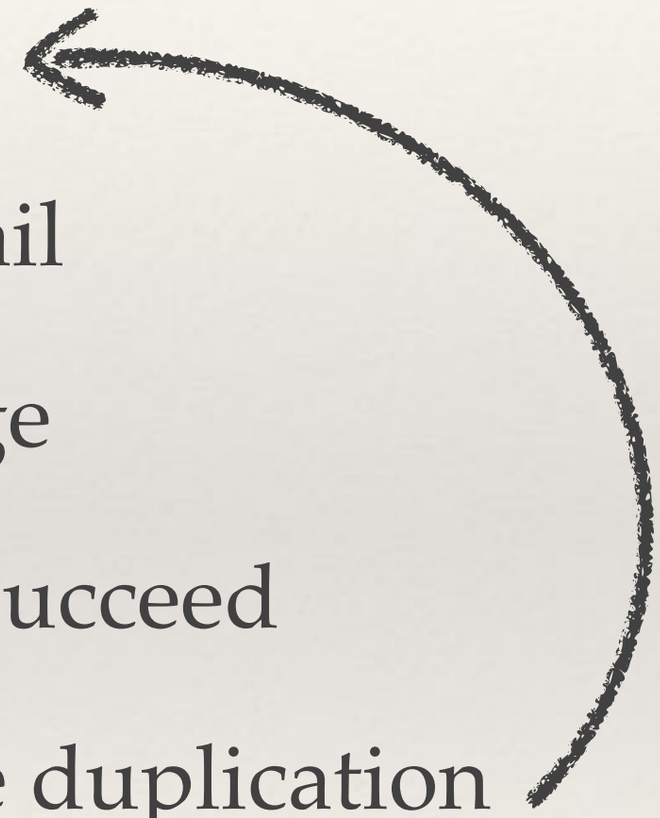
TDD in action: Case study

- "Over the last few years, Micah Martin and I've been working on an application named FitNesse (www.fitnesse.org).
- FitNesse is a Web-based application using a Front Controller that defers to servlets that direct views.
- Downloaded tens of thousands of times, FitNesse consists of 45,000 lines of Java code in nearly 600 files.
  - Almost 20,000 of those lines and 200 of those files are unit tests.
- Over 1,100 specific test cases contain many thousands of assertions, giving FitNesse test coverage of over 90 percent (as measured by Clover, http://cenqua.com/clover).
- These tests execute at the touch of an IDE (integrated development environment) button in approximately 30 seconds.
- These tests, along with the convenience of easily executing them, have benefits that far exceed simple software verification."

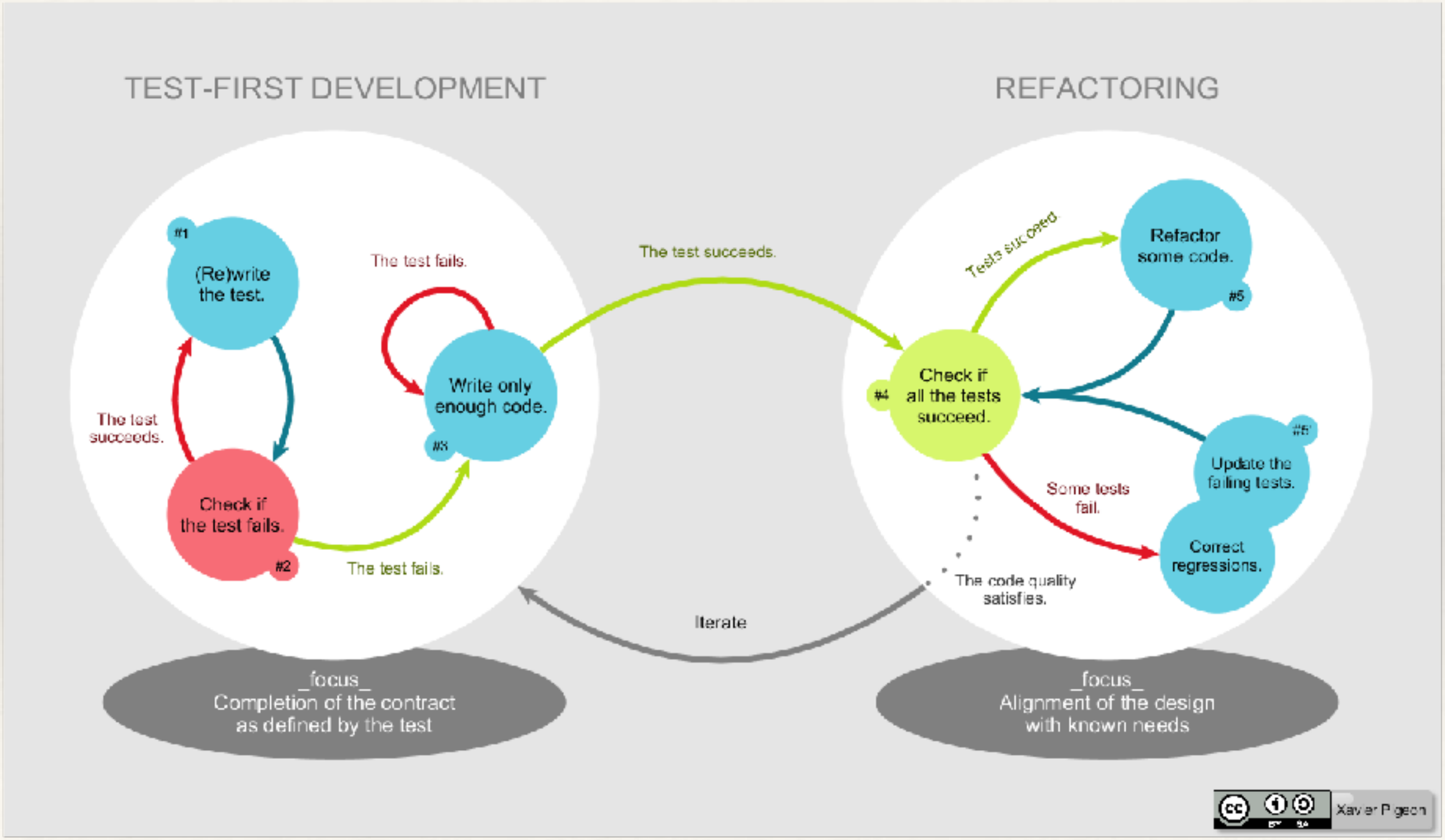
Source: Professionalism and Test-Driven Development, Robert C. Martin, IEEE Software, 2007

TDD process cycle

1. Add a little test
2. Run all tests and fail
3. Make a little change
4. Run the tests and succeed
5. Refactor to remove duplication



TDD and refactoring



https://en.wikipedia.org/wiki/Test-driven_development#/media/File:TDD_Global_Lifecycle.png

"You know you are working on clean code when each routine you read turns out to be pretty much what you expected. You can call it beautiful code when the code also makes it look like the language was made for the problem."

–Ward Cunningham

TDD - benefits

Better Design	Cleaner code (because of refactoring)
Safer refactoring	Increased quality
Better code coverage	Tests serve as documentation
Faster debugging	Most often, the failing code/test is in the most recently changed code
Self-documenting tests	Test-cases show/indicate how to use the code

Some Java Tools for TDD

JUnit

TestNG

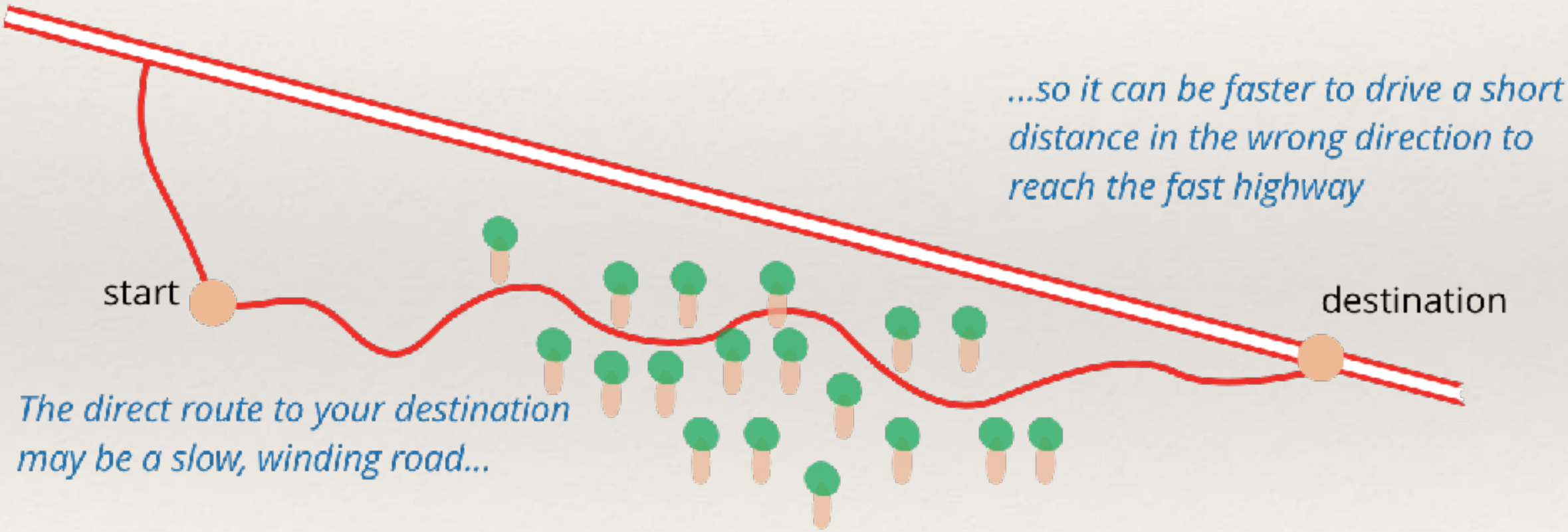
jWalk

Mockito

"TDD helps with, but does not guarantee, good design & good code. Skill, talent, and expertise remain necessary."

— *Esko Luontola*

TDD slows down development?



Unit testing (with JUnit)

Unit testing: common excuses

- I am paid to write code, not tests!
- I am a developer, not tester - so I'm not responsible for writing tests!
- We already have tests - why do we need unit tests?
- We are working on a tight deadline - where do I have time for writing unit tests?
- I don't know how to write unit tests
- Ours is legacy code - can't write unit tests for them
  - If I touch the code, it breaks!

What dependencies can unit test have?

- A test is not a unit test if:
  - It talks to the database
  - It communicates across the network
  - It touches the file system
  - It can't run at the same time as any of your other unit tests
  - You have to do special things to your environment (such as editing config files) to run it.

— Michael Feathers, in A Set Of Unit Testing Rules (2005)

The law of low-hanging fruit!

Start with something really simple. Implement an obvious test case.

F.I.R.S.T Tests

- Fast
- Independent
- Repeatable
- Self-validating
- Timely

JUnit

Original developers	Kent Beck, Erich Gamma
Stable release	JUnit 4.12
GitHub repo	https://github.com/junit-team/junit4
License	Eclipse Public License (EPL)
Website	www.junit.org

Source: wikipedia

"Never in the field of software engineering has so much been owed by so many to so few lines of code."

–Martin Fowler (on JUnit)

Getting started with JUnit

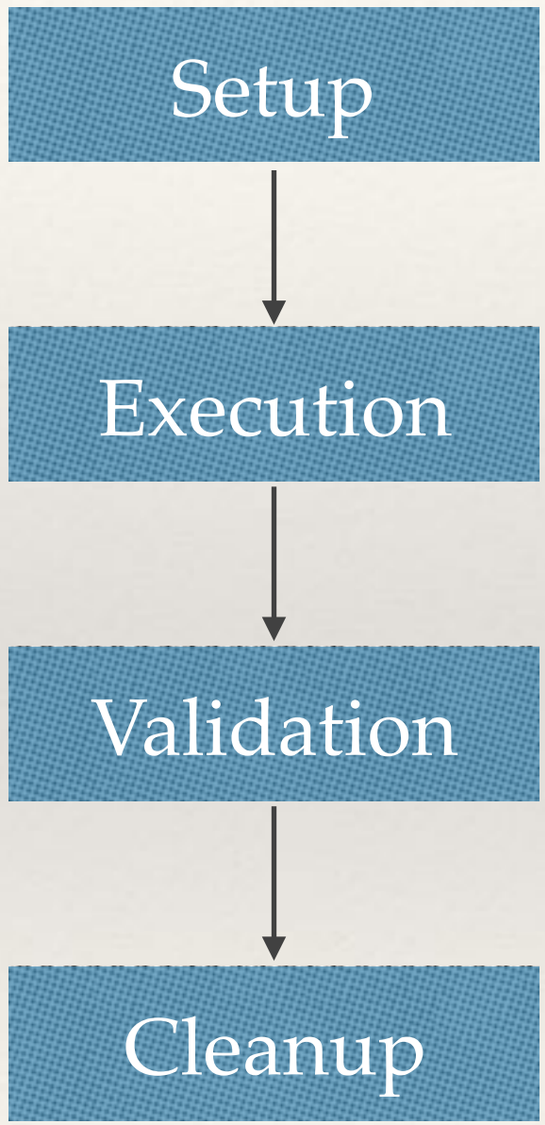
- Step 1: Download and Install JUnit
  - https://github.com/junit-team/junit4/wiki/Download-and-Install
- Step 2: Try "Getting Started" example:
  - https://github.com/junit-team/junit4/wiki/Getting-started

JUnit

- Tests are expressed in ordinary source code
- The execution of each test is centered on an instance of a TestCase object
- Each TestCase, before it executes the test, has the opportunity to create an environment for the test, and to destroy that environment when the test finishes
- Groups of tests can be collected together, and their results of running them all will be reported collectively
- We use the language's exception handling mechanism to catch and report errors

Source: Test Driven Development: By Example, Kent Beck, 240 pages, Addison-Wesley Professional, 2002;

Test case structure



Assert methods in JUnit

assertEquals()/ assertNotEquals()	Invokes the equals() methods on the arguments to check whether they are equal
assertSame()/ assertNotSame()	Uses == on the arguments to check whether they are equal
assertTrue()/ assertFalse()	Checks if the given boolean argument evaluates to true or false
assertNull()/ assertNotNull()	Checks if the given argument is null or NOT null
assertArrayEquals()	Checks if the given array arguments passed have same elements in the same order

Arrange-Act-Assert (AAA) Pattern

```
@Test
  public void evaluatePlus() {
    // Arrange
    Expr exprNode = new Plus(new Constant(10), new Constant(20));
    // Act
    int evalValue = exprNode.eval();
    // Assert
    assertEquals(evalValue, 30);
}
```

- **Arrange:** Setup things needed to execute the tests
- **Act:** Invoke the code under test
- **Assert:** Specify the criteria for the condition to be met for the test to pass (or else, it is test failure)

Test fixtures

- You can provide a text fixture to ensure that the test is repeatable, i.e., start in the same state so it produce the same results and clean up after running the tests.
- You can use @Before typically to initialise fields. Similarly you can use @After to execute after the test is complete.

Unit testing best practices

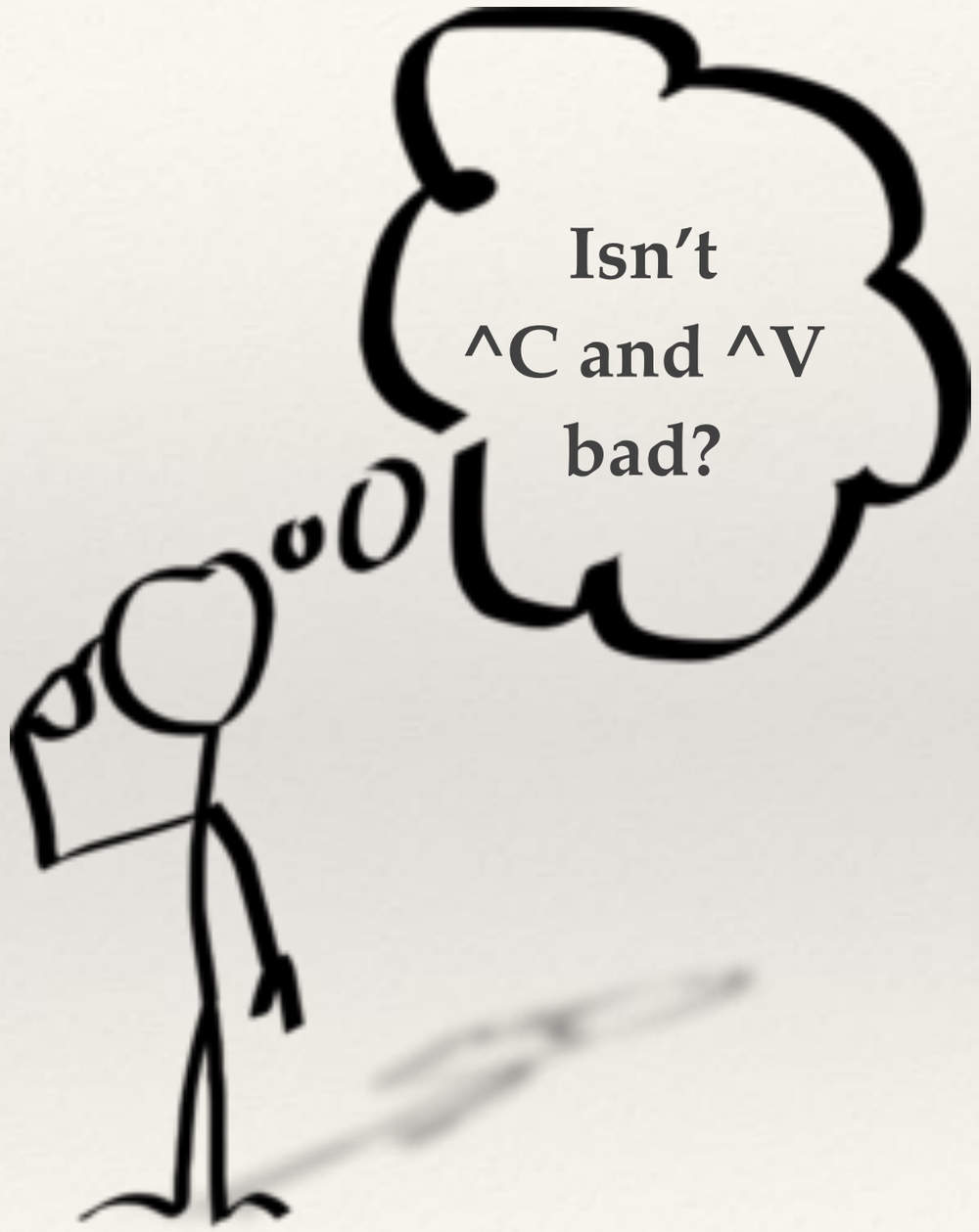
- Test name should reflect the intent/purpose & describe the feature/specification
- Unit test should test a *single* concept
- Readability matters in the tests - use comments as appropriate
- Test code also needs maintenance and factors like readability are important (just like production code)
- Tests obvious functionality as well as corner cases - both are important

Best Practice

Identify & refactor test smells

Duplicate code in unit tests?

- Duplicate code segments in code is bad
  - Violates ‘Don’t Repeat Yourself’ principle
- Duplicate code segments in tests is often needed
  - To test similar but slightly different functionality



Mocking

Tools for mocking

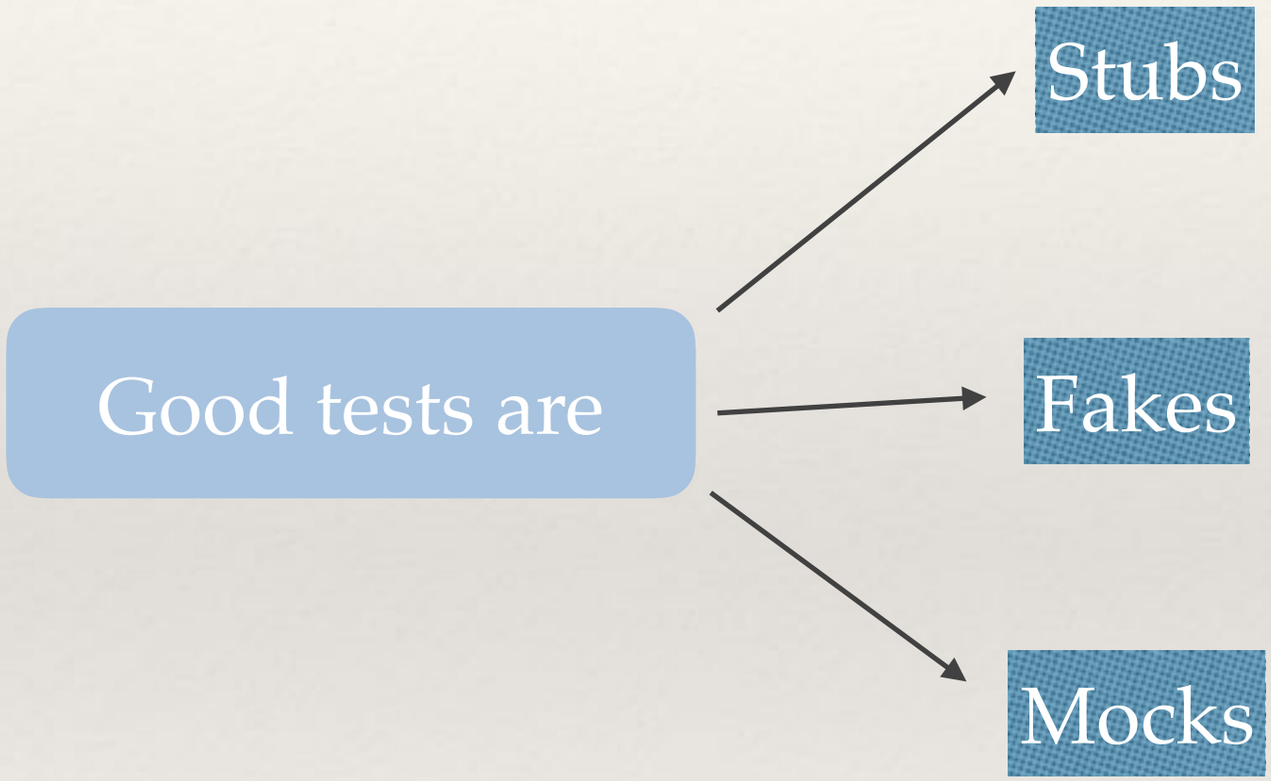
jMock

Mockito

EasyMock

PowerMock

Test “doubles”



Terminology

- System Under Test (SUT) is the part of the system being tested - and could be of different levels of granularity like a class or an application.
- Depended On Component (DOC) means the entity that SUT requires to fulfil its objectives.
  - Mocking can be used to decouple SUT from DOCs - this enables unit tests to run independently

Test "doubles"

- Simplest one is stub
  - It is a temporary one that does nothing and stands for the real object
- More sophisticated than stub is a fake
  - It provides an alternative simple implementation than the original one
- Mocks simulate the original object
  - Provides implementation of the object with assertions, returning hard-coded values, etc

TDD Principle

"Fake it till you make it"

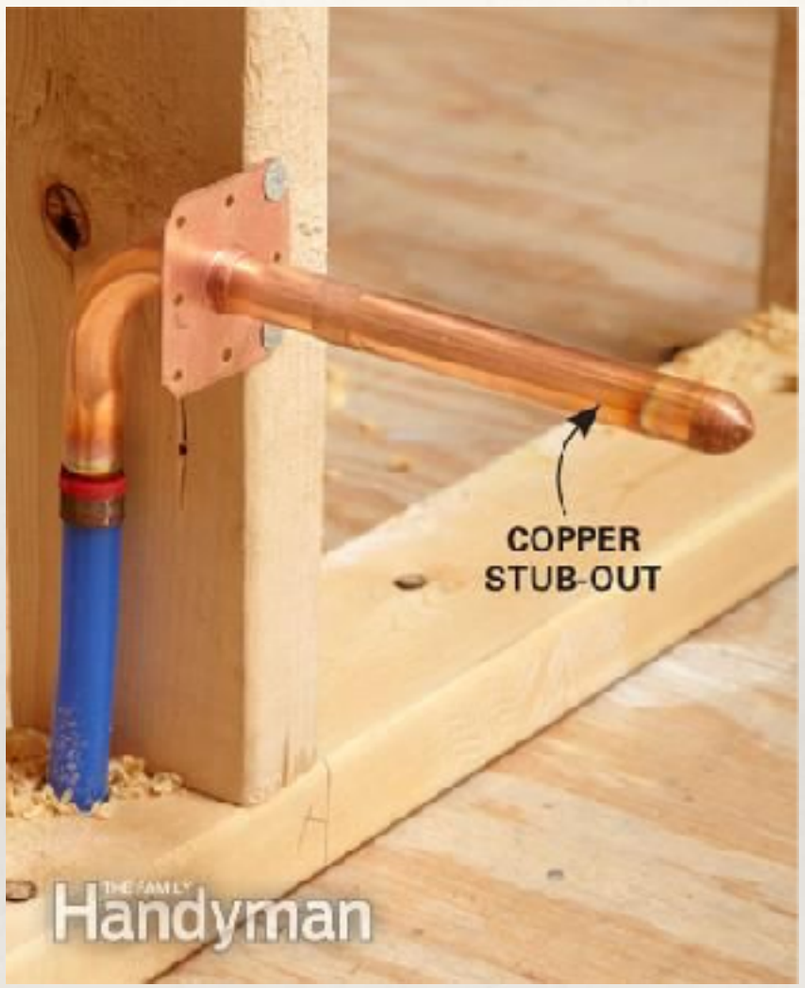
Fakes

- Fake is a type of test double.
- It is simpler, easier, or cheaper to use than the actual dependency.
- An example is using “HSQLDB” (in-memory database) instead of actual “Microsoft SQL” database.
- Fakes are mainly used in integration tests.

Why "test doubles"?

- Allow us to implement the code for specific functionality without getting all the dependent code in place (without getting "dragged down" by dependencies)
- Can independently test only the given functionality
  - If it fails, we know its because of the given code and not due to dependencies

Stub-outs



- Plumbing - “Stub-Outs” - Short pipes put in early during plumbing work during constructions - eventually, they will be replaced with real pipes and fixtures.

Related topics

Preparatory refactoring

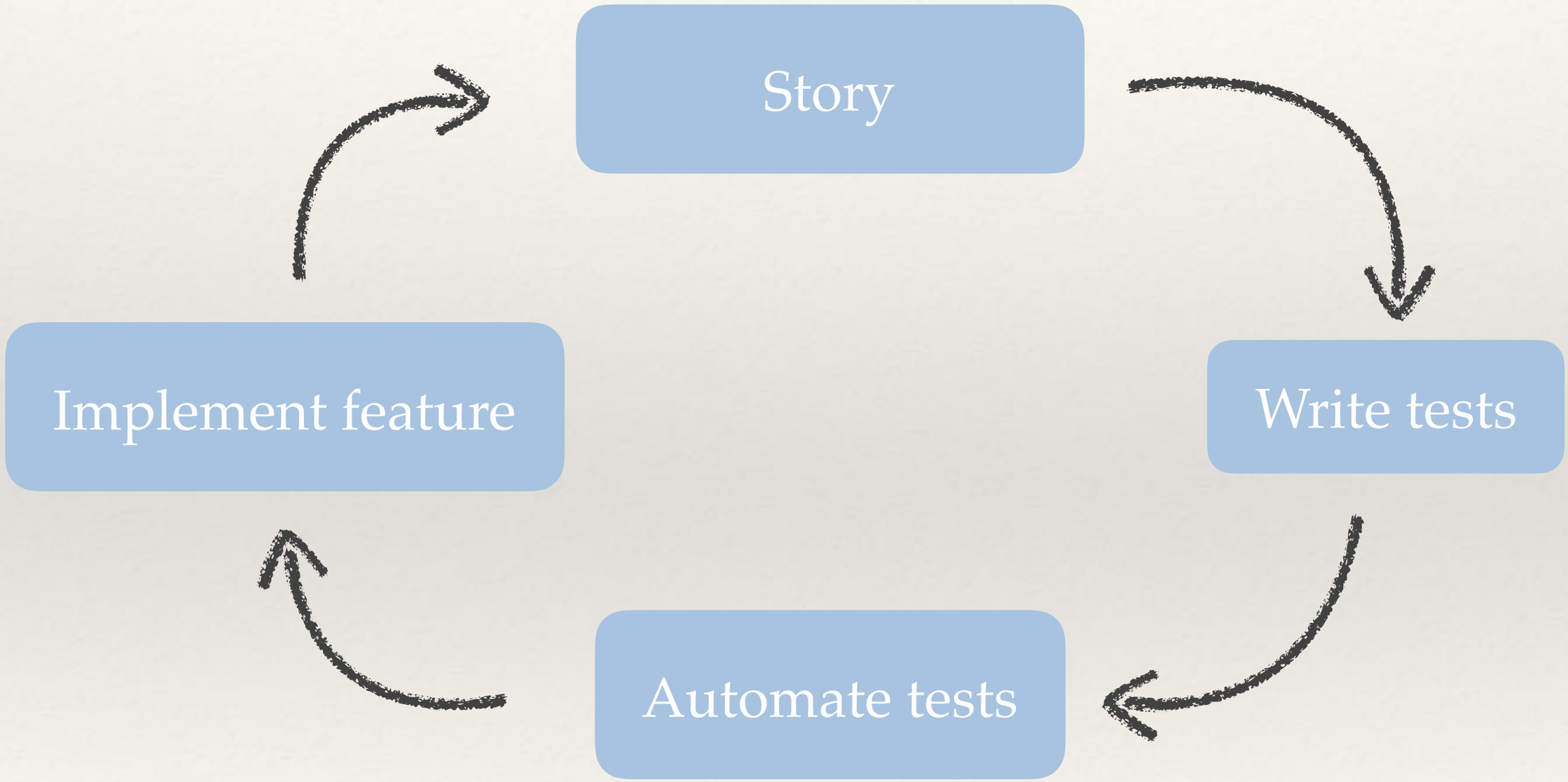
The Two Hats

In the Refactoring book, I passed on Kent's metaphor of the two hats. His notion is that when you're programming, you can operate in one of two modes: refactoring and adding function. When you're wearing the refactoring hat, every change you make preservers observable behavior, keeps the tests green, and allows you to make many small changes without going near a debugger.

When you add function, however, things are more open ended as you will add tests and break existing tests. The adding function hat is more stressful and riskier, so it's nice to wear the refactoring hat as much as possible.

Source: https://martinfowler.com/articles/preparatory-refactoring-example.html

Acceptance TDD



Code quality tools

PMD

FindBugs

CheckStyle

IntelliJ IDEA /
Eclipse analysers

Image credits

- https://s-media-cache-ak0.pinimg.com/736x/78/c9/9e/78c99e530a69406ec249588ef87a59a9.jpg
- http://www.datamation.com/imagesvr_ce/4130/development-driven.jpg
- https://s-media-cache-ak0.pinimg.com/736x/ae/22/01/ae2201013b69918a20b6de0adf1517a1.jpg
- http://blog.itexus.com/img/tdd_comics.png
- https://martinfowler.com/articles/preparatory-refactoring-example/jessitron.png
- https://i2.wp.com/www.gilzilberfeld.com/wp-content/uploads/2011/02/regret-testing.png
- https://pixabay.com/en/pegasus-horse-winged-mythology-586124/
- https://pixabay.com/en/horse-gallop-horses-standard-1401914/
- https://refactoring.guru/images/content-public/index-clean-code.png
- http://www.lifeisanecho.com/wp-content/uploads/2016/06/ar131070344825846.jpg
- https://pixabay.com/en/ball-chain-bug-jail-insect-46207/
- https://pixabay.com/en/home-old-school-old-old-building-1824815/
- https://pixabay.com/en/escalator-stairs-metal-segments-283448/
- https://devops.com/wp-content/uploads/2016/07/tdd-01-1.jpg
- http://www.nanrussell.com/wp-content/uploads/2015/08/Not-me.jpg
- https://cdn.meme.am/instances/500x/43446748/winter-is-coming-brace-yourselves-endless-client-revisions-are-coming.jpg
- https://t4.ftcdn.net/jpg/00/87/17/55/240_F_87175567_I7FK0h2XNxrwtnoYbufTzvpLv3p2cFrk.jpg
- https://cdn.meme.am/cache/instances/folder518/500x/64808518/yeah-if-you-could-just-if-we-could-stop-changing-requirements-every-5-minutes-that-would-be-great.jpg

Image credits

- http://optymyze.com/blog/wp-content/uploads/sites/2/2017/02/change.jpg
- http://bookboon.com/blog/wp-content/uploads/2014/03/D%C3%A9veloppez-votre-potentiel.jpg
- https://techbeacon.com/sites/default/files/most_interesting_man_test_in_production_meme.jpg
- https://cdn-images-1.medium.com/max/490/1*k-OkcZd2fAyZf1WBkharGA.jpeg
- https://akchrish23.files.wordpress.com/2012/12/far-side-first-pants-then-your-shoes.jpg
- https://image.slidesharecdn.com/its-all-about-design-1232847245981881-1/95/its-all-about-design-10-728.jpg?cb=1232825731
- http://www.fox1023.com/wp-content/uploads/2016/06/fail-sign1.jpg
- https://vgarmada.files.wordpress.com/2012/04/pass-sign.jpg
- http://codelikethis.com/lessons/agile_development/make-it-green.png
- https://refactoring.guru/images/content-public/index-refactoring-how.png
- http://geek-and-poke.com/geekandpoke/2014/1/15/philosophising-geeks
- https://employmentdiscrimination.foxrothschild.com/wp-content/uploads/sites/18/2014/06/20350757_s.jpg
- https://static1.squarespace.com/static/5783a7e19de4bb11478ae2d8/t/5821d2ea09e1c46748737af1/1478614300894/shutterstock_217082875-e1459952801830.jpg
- https://lh3.googleusercontent.com/-eM1_28qE1cM/U1bUFmBU1NI/AAAAAAAAHEk/ZqLcxFEhMuA/w530-h398-p/slide-32-638.jpg
- http://www.trainingforwarriors.com/wp-content/uploads/2015/03/3-Laws-Post.jpg

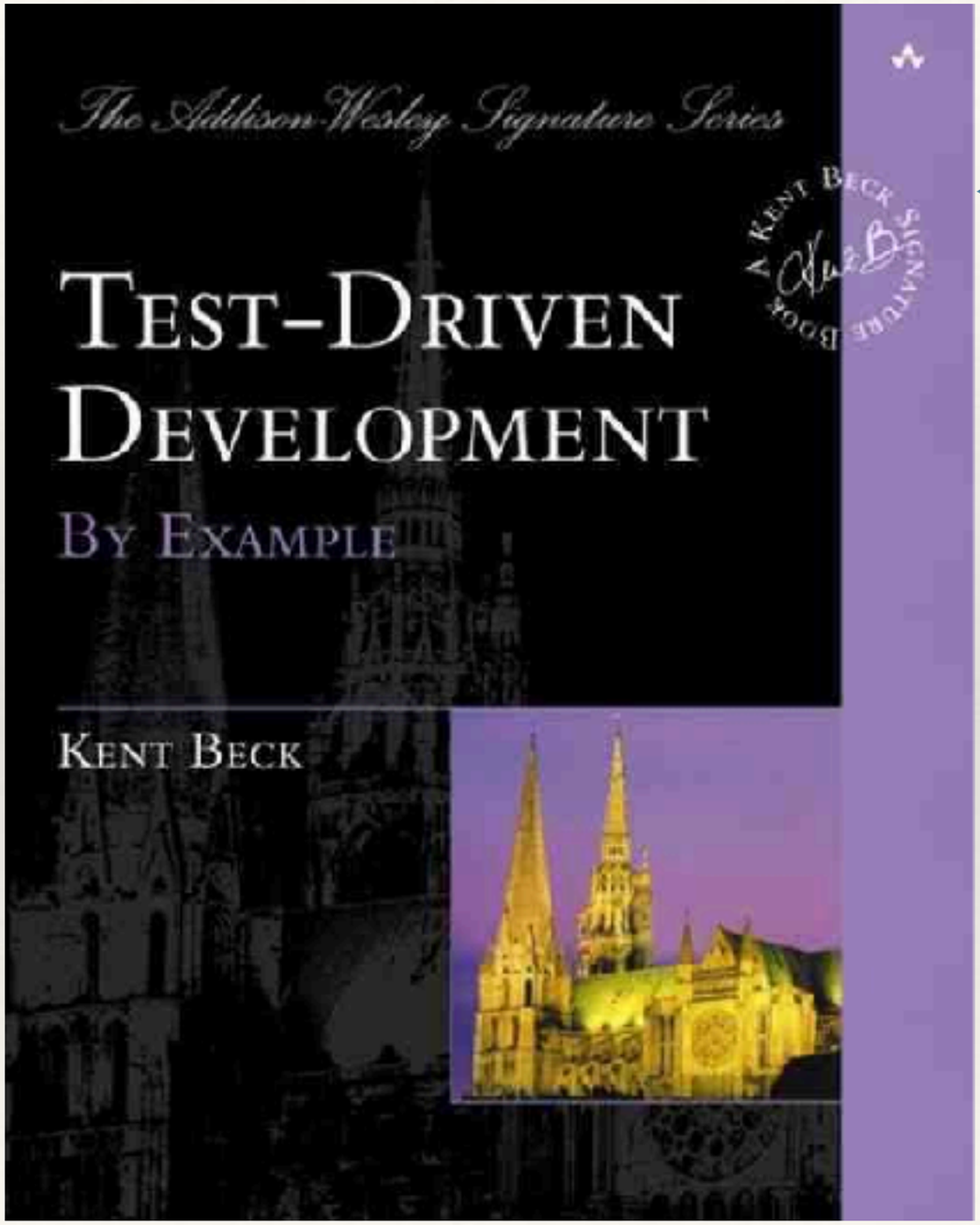
Image credits

- https://patientsrising.org/sites/default/files/Step%20Therapy.PNG
- https://1.bp.blogspot.com/-Q00OoZelCic/WFSmGIUCrGI/AAAAAAAAAx5U/i59y1h-czIIXNswq6aMdAOUGjgPLaPdxACLcB/s1600/awful.png
- http://s2.quickmeme.com/img/f4/f4b4744206cf737305f1a4619fefde7b0df54ecc0dc012adcceaadf93196a7e8.jpg
- https://pbs.twimg.com/media/CeZu1YjUsAEfhcP.jpg:large
- https://upload.wikimedia.org/wikipedia/en/thumb/f/ff/Poison_Help.svg/1024px-Poison_Help.svg.png
- http://data.whicdn.com/images/207820816/large.jpg
- http://orig04.deviantart.net/c7cb/f/2014/171/d/a/the_bare_minimum_bandits_by_shy_waifu-d7n8813.png
- http://fistfuloftalent.com/wp-content/uploads/2017/04/3c0f8f4.jpg

References

- Wikipedia page - JUnit: https://en.wikipedia.org/wiki/JUnit
- Wikipedia page - Test Driven Development: https://en.wikipedia.org/wiki/Test-driven_development
- Professionalism and TDD: https://8thlight.com/blog/uncle-bob/2014/05/02/ProfessionalismAndTDD.html
- Preparatory refactoring: https://martinfowler.com/articles/preparatory-refactoring-example.html

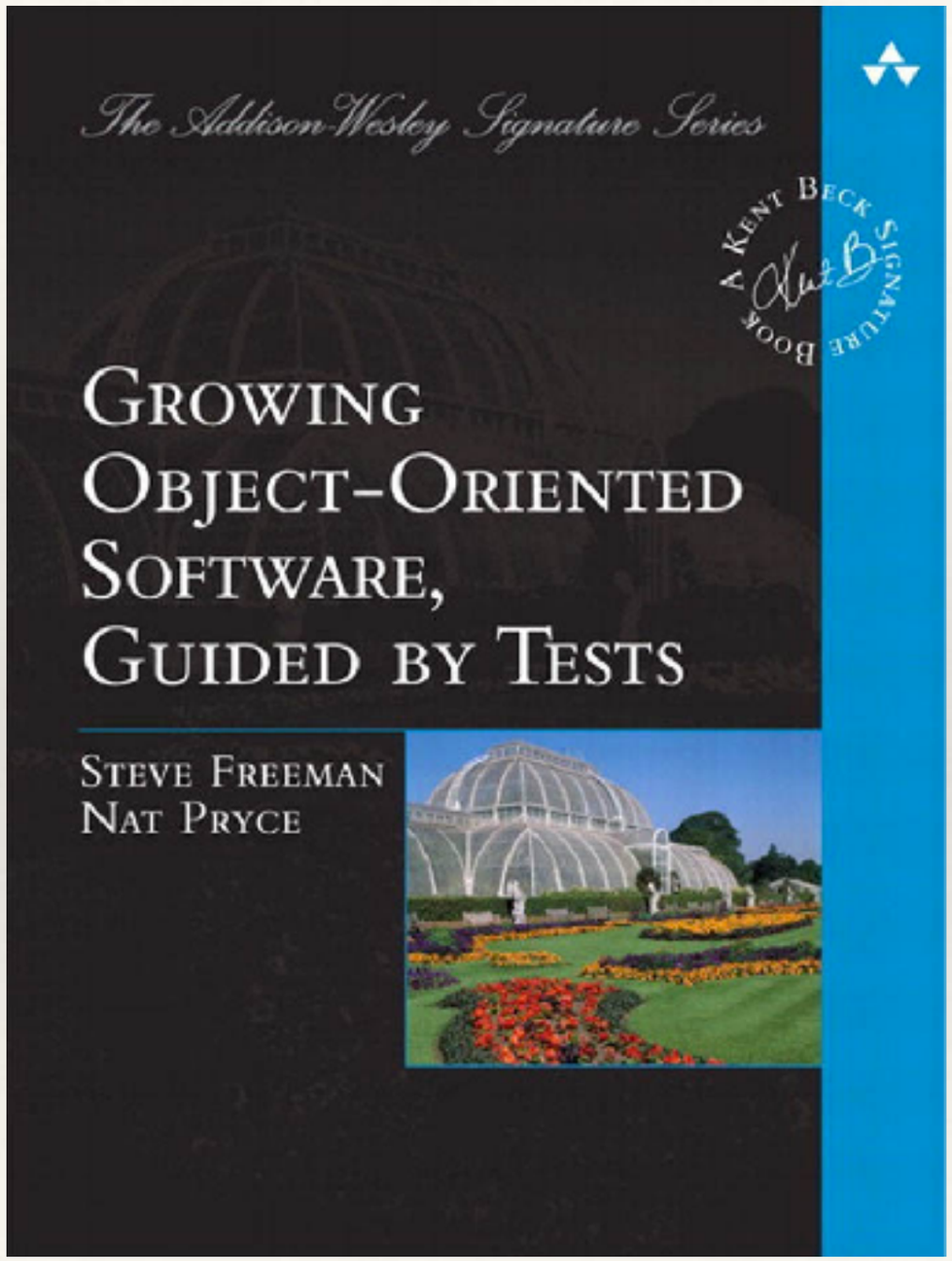
Recommended reading



Classic on TDD by its originator Kent Beck

Test Driven Development: By Example, Kent Beck, 240 pages, Addison-Wesley Professional, 2002;

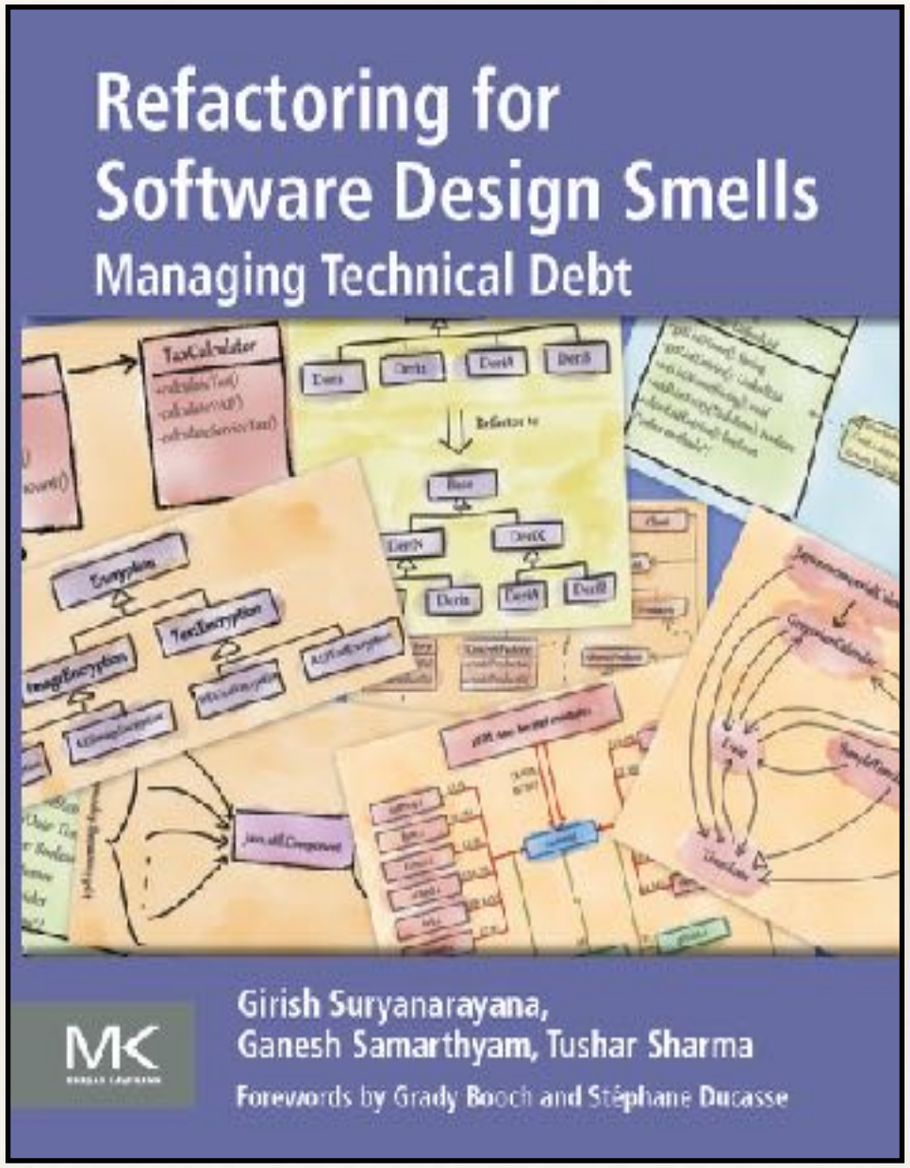
Recommended reading



Covers principles, best practices, heuristics, and interesting examples

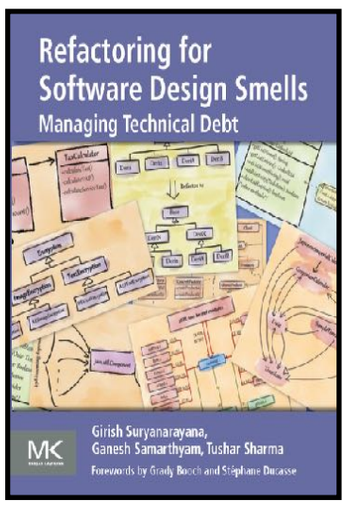
Growing Object-Oriented Software, Guided by Tests, Steve Freeman, Nat Pryce, Addison Wesley, 2009

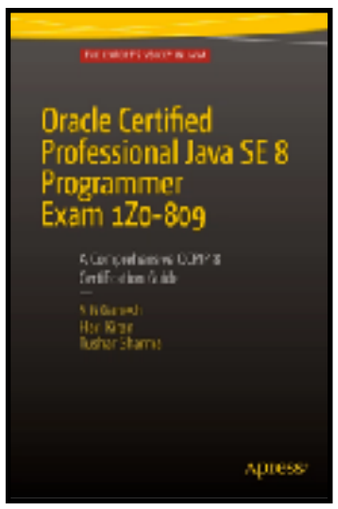
Recommended reading

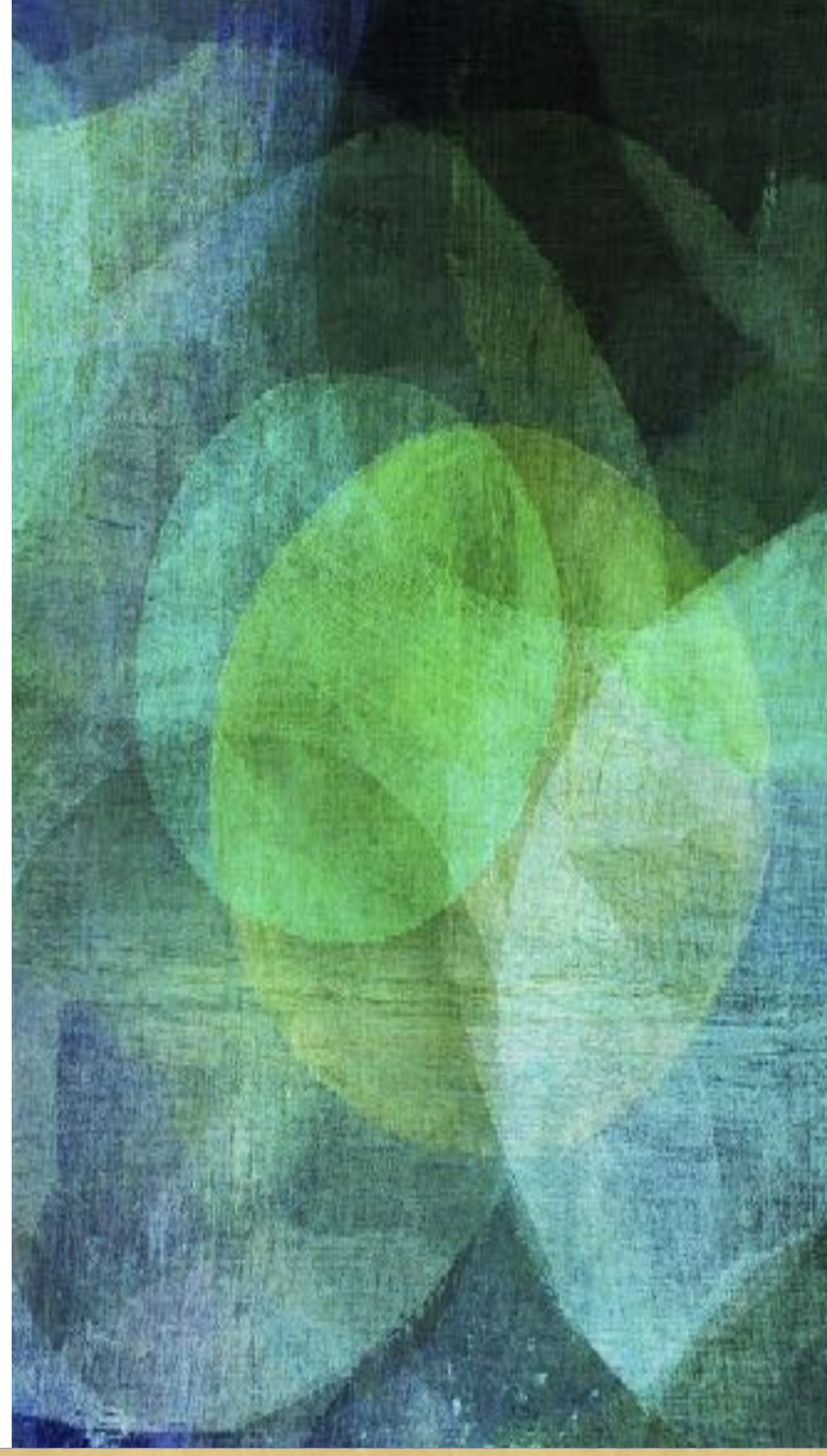


Covers design smells as violations of design principles

"Refactoring for Software Design Smells: Managing Technical Debt", Girish Suryanarayana, Ganesh Samarthyam, Tushar Sharma, ISBN - 978-0128013977, Morgan Kaufmann/Elsevier, 2014.







ganesh@codeops.tech

www.codeops.tech

+91 98801 64463

@GSamarthyam

slideshare.net/sgganesh

bit.ly/sgganesh

CodeOps