Mocking - Unit Testing - TDD

Cel Pynenborg

AXXES

Cel Pynenborg

Full Stack Java Developer @ Axxes

2 Years

Projects:

- *HealthConnect*
- *TomTom*
- *ST Engineering (Current)*

AXXES

EForms Core/HealthLink

HealthConnect - Corilus

1Y3M

Communication between caregivers – organizations

Digital forms

Version 2

Persistence => Forms became Dossiers

AXXES

TT – Open Maps

TomTom

1Y

Added value data

Indexing all the oneway streets

Merging open-source with closed-source data

Big data project

ST-E – FnP

ST Engineering

Recently

Fault and Performance

Telemetry and metrics

Sattelite networks

Content

- Software Testing
  - What? Why? How?
  - Testing Levels
- Testing Plain Old Java Code
  - JUnit
  - Mockito
  - AssertJ
- Test Driven Development
  - What? Why? How?
  - TDD and Testing in modern software development
- Testing Spring Code / RL scenarios (Extra)
  - @SpringBootTest
  - TestContainers / RestAssured / HoverFly / Selenium

Software Testing

–

A byte sized explanation

What is Software Testing

- Supplementary (non-business) code / software
- Verify the behavior of business logic
- Objective and independent view into the software

AXXES

Why should I test my software

- Confidence in the source code
- Know the strengths and weaknesses
- Possibilities are infinite
- Sometimes there's more testing code than business logic
- Self-documenting
  - updateAccount_asAdmin_hasNotVerifiedAccount
  - BDD (Behaviour Driven Development)

```
new Time().equals(new Money())
```

AXXES

Things to consider

- Confidence in the test code
- “Good code” with bad tests cannot be considered reliable
- Bad code with good tests is a solvable issue
- Bad code with bad tests is a nightmare

Signs that you're doing it wrong

- Convoluted setup
- Duplicate tests
- Slow tests
- Flaking tests
  - Race conditions
  - Reliance on the order in which tests are run

These factors result in "loss of confidence" in the code base!

How to do it right

- KISS (Keep-it-simple)
- Isolated test scenarios
- Run tests often
- Treat test code with the same care as core logic

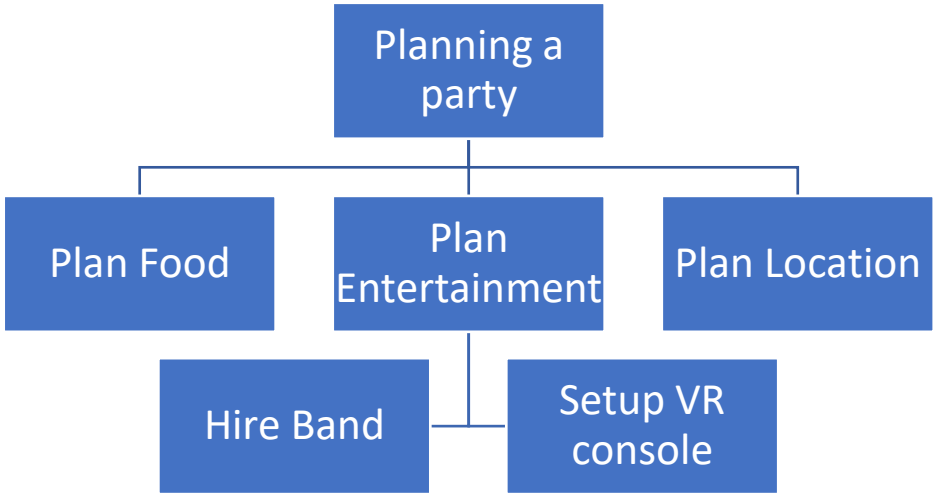
Testing Levels: Unit Tests

–

We’re supposed to be a unit!

What's a unit test?

- Automated test
- Fully Controls all the pieces it is testing
- Are isolated
- Are independent of each other
- Runs in memory
- Is consistent
- Fast
- Tests a single concept/class
- Readable
- Maintainable



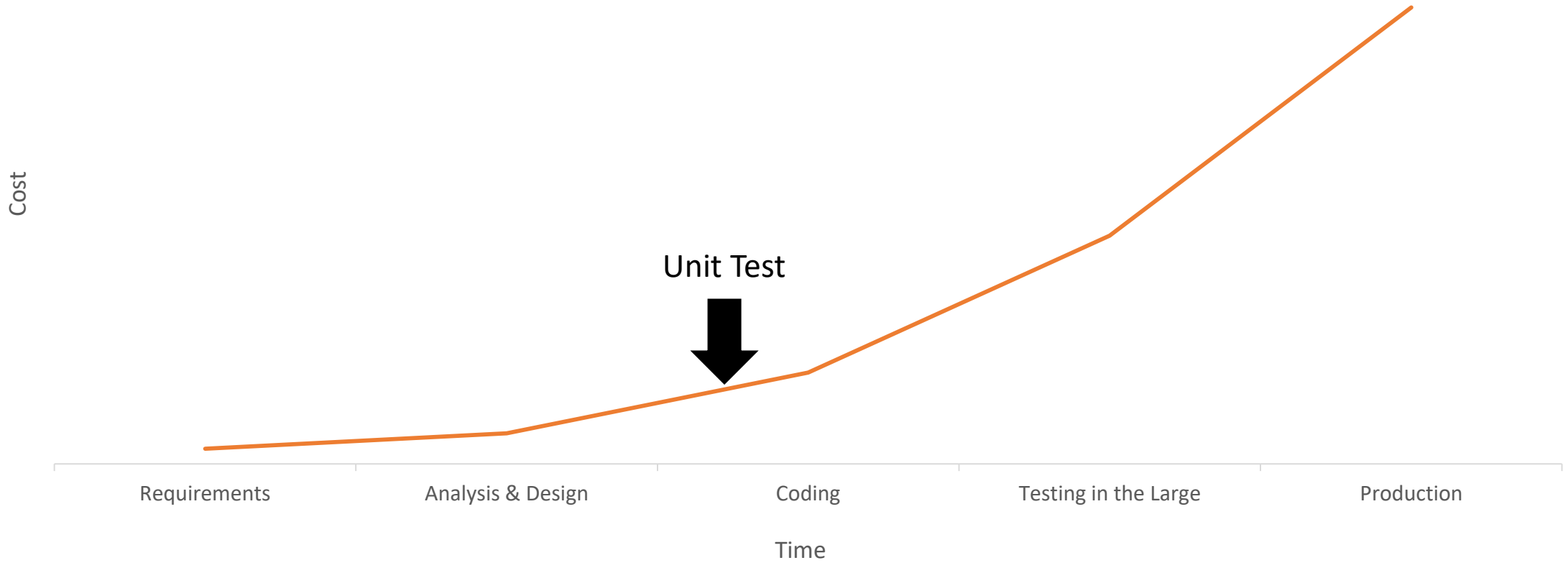

```
public class AuthenticationService {

        void authenticate(String token) {
            …
        }
}
```

AXXES

Why write unit tests?

- Short term loss, long term gain
- Are owned by the team and are everyone's responsibility
- Super handy during development/bug fixing/refactoring
- Enables frequent integration
- Small units == less likely to make mistakes

COST OF CHANGE GRAPH



Testing Levels: Integration Tests

–

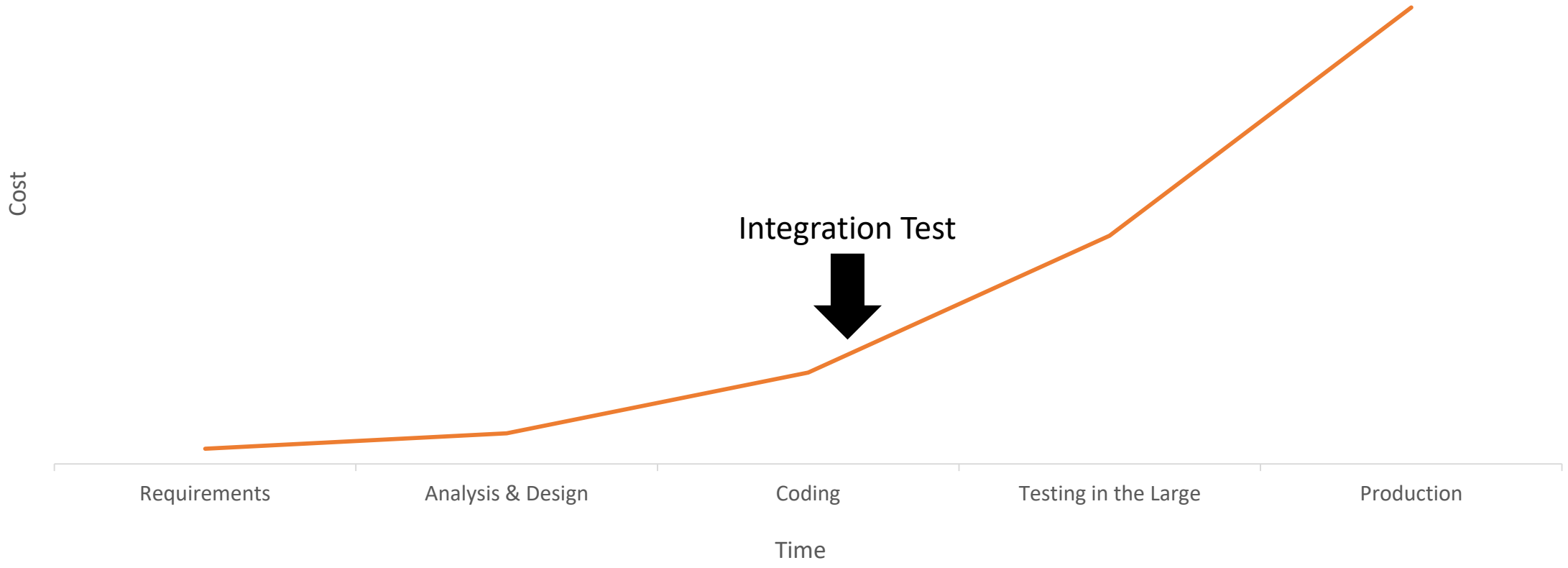
Verifying the interaction between components

What's an integration test?

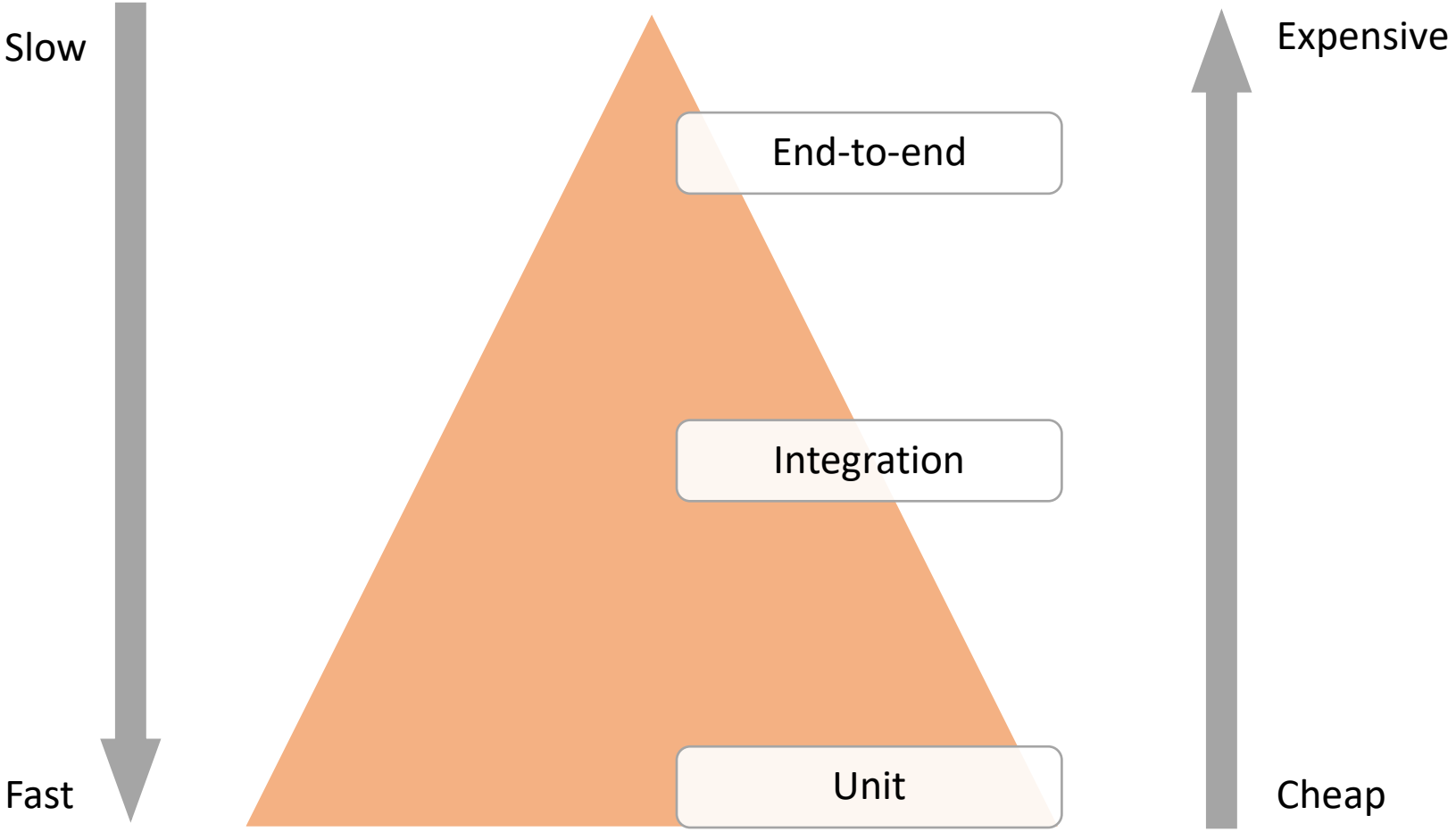
- Automated test
- Integration of multiple units/components
- No mocking, or at least as little as possible
  - As a result: less control over individual components => input/output validation
- Slower than unit tests
- More complex than unit tests
- Easily mappable to user stories
- Regression detection
- Harder to maintain
- Documentation

AXXES

COST OF CHANGE GRAPH



Test Automation Pyramid



Testing Plain-old Java Code

–

Arrange – Act – Assert

Testing Plain Old java code

- Using testing frameworks and tools
  - JUnit
    - Practically the standard when it comes to testing frameworks for Java
  - Mockito
    - Library that allows the creation of mock objects which allow for easy control of components in the unit test
  - AssertJ
    - Fluent-style assertion library
    - Boasts wide adoption
      - Specific implementations for specific cases (e.g. XMLAssert)

DEMO & EXERCISE

–

DIY

Test Driven Development

–

It’s a never-ending cycle

“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.”

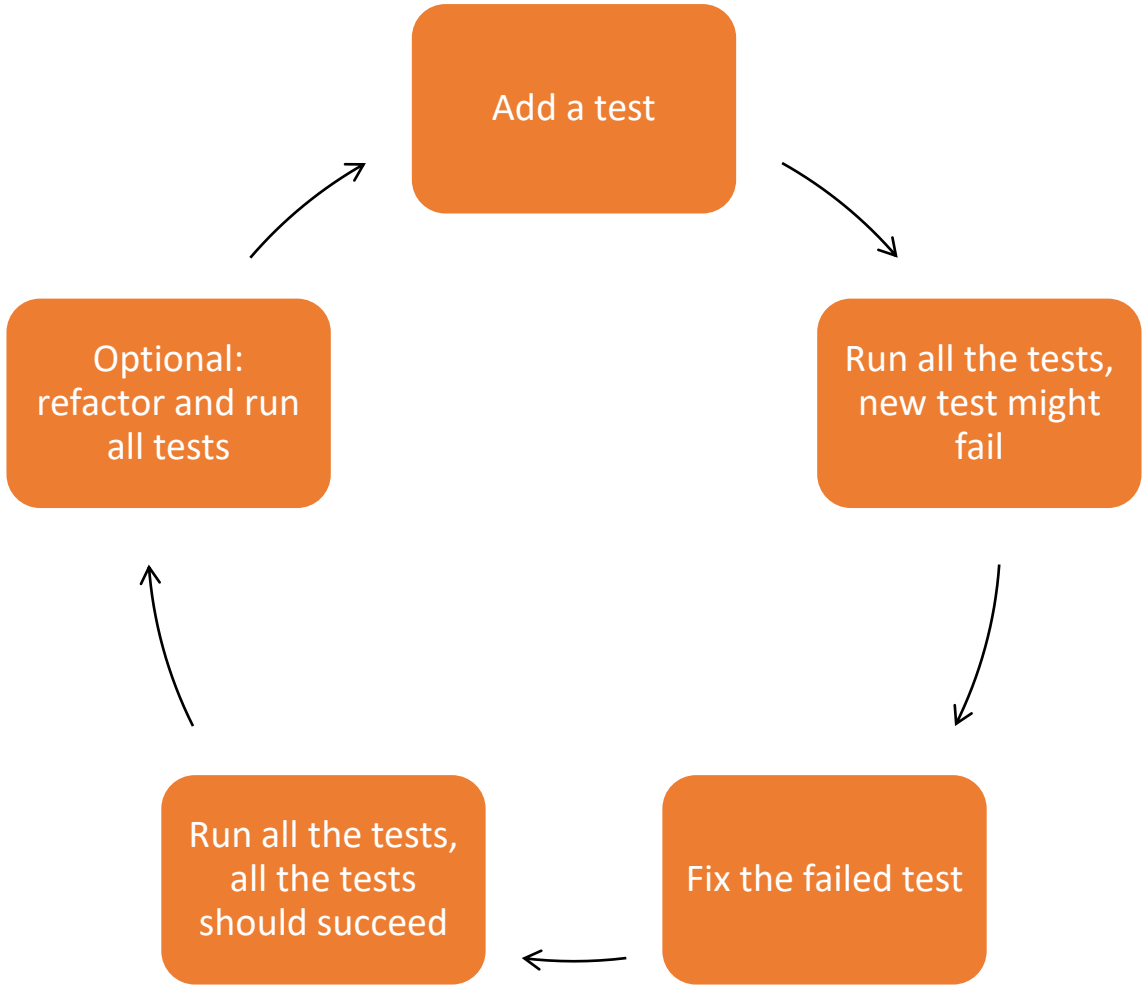
—

WIKIPEDIA

In other words

- Convert requirements into test cases
- Write tests while writing business logic
  - Added benefit of having tests of old and new functionalities
  - Increase development speed
  - Track the status of the project by running all the tests
- Continuous cycle of the same steps

TDD Cycle



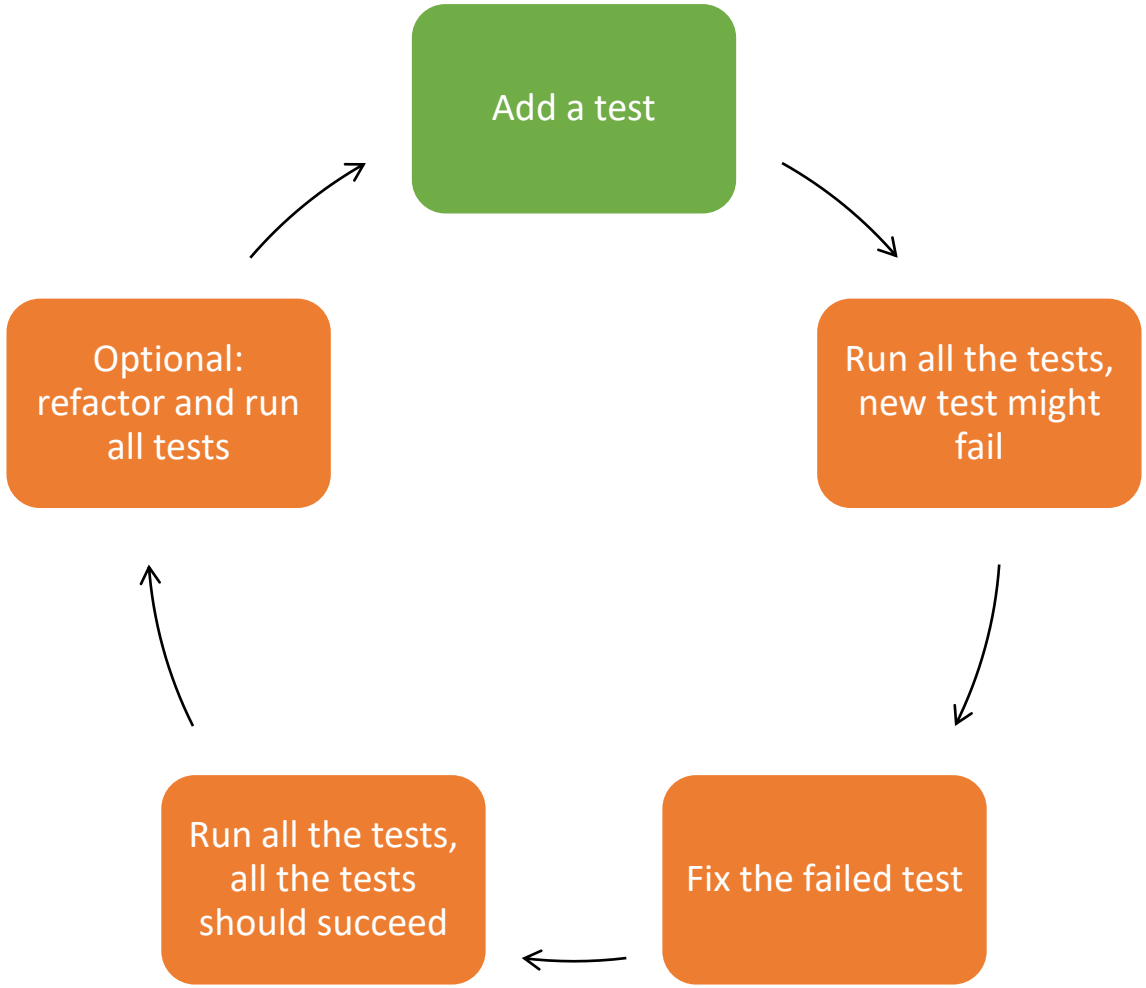
AXXES

Why the cycle?

- Increase coverage to near 100%
- Small steps
- Detect faults in the design/architecture
- Find bugs during development
  - Defensive programming

If all the requirements are converted into test cases and all the tests succeed, all the requirements are (correctly) implemented

TDD Cycle

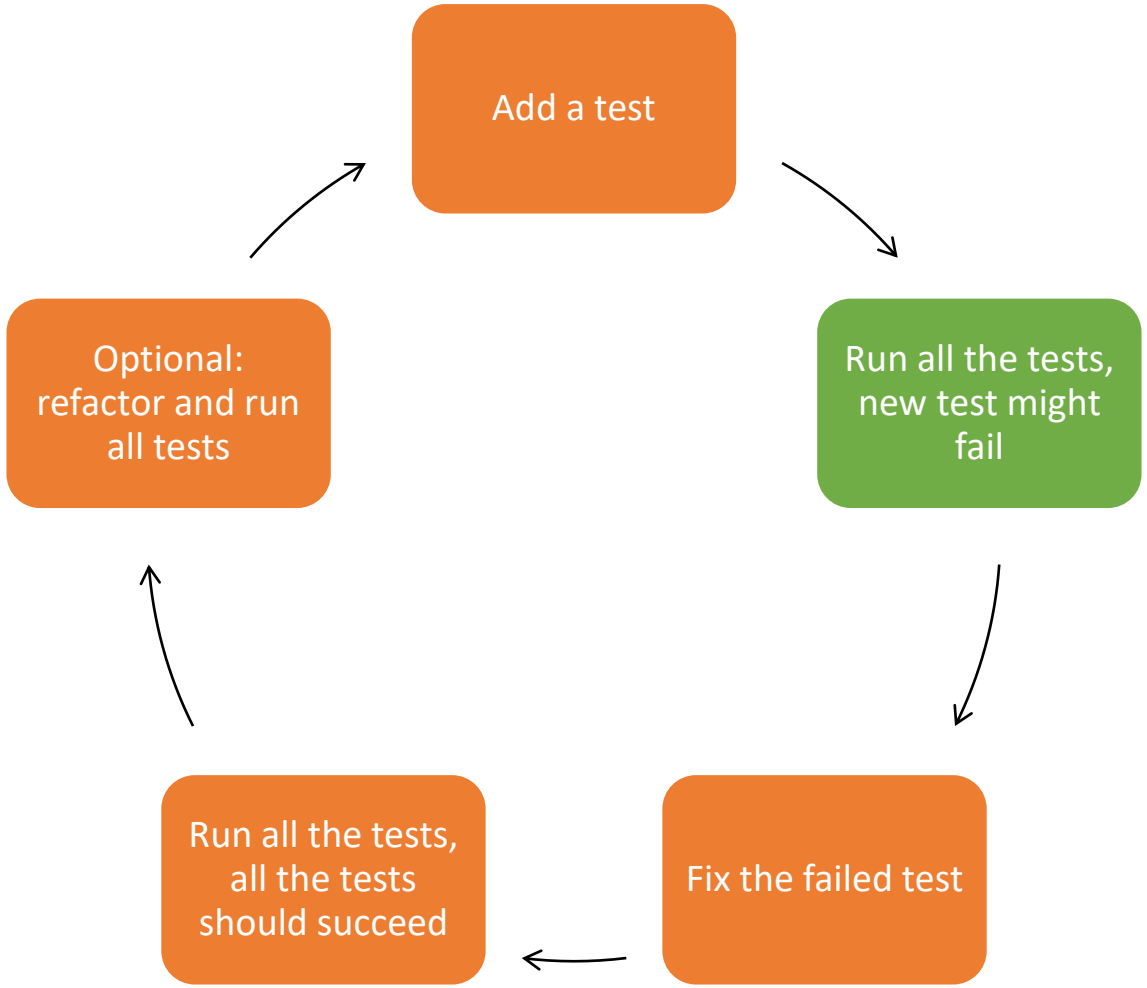


AXXES

Add a test

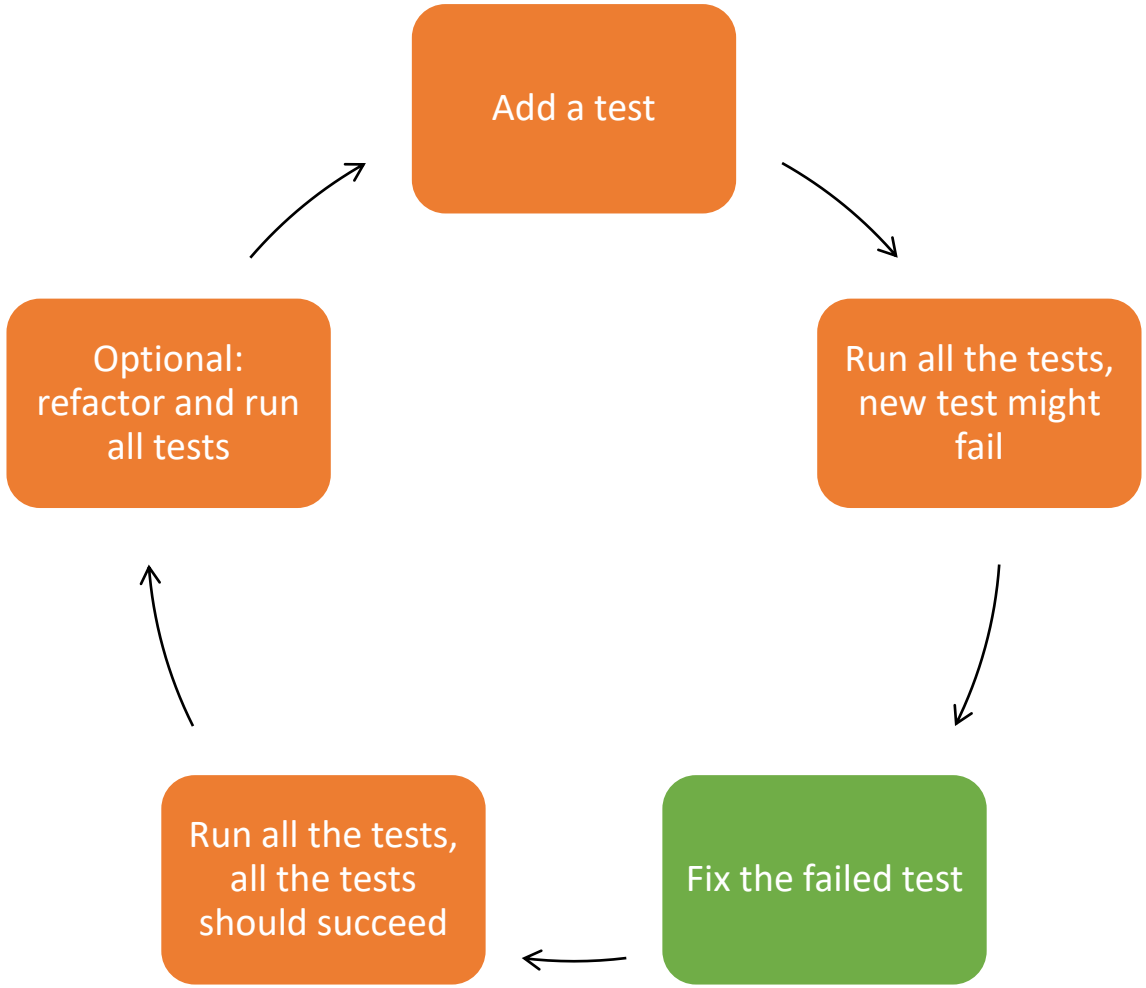
- Unit test
- Small increment
- One failing test at a time

TDD Cycle



AXXES_

TDD Cycle



AXXES

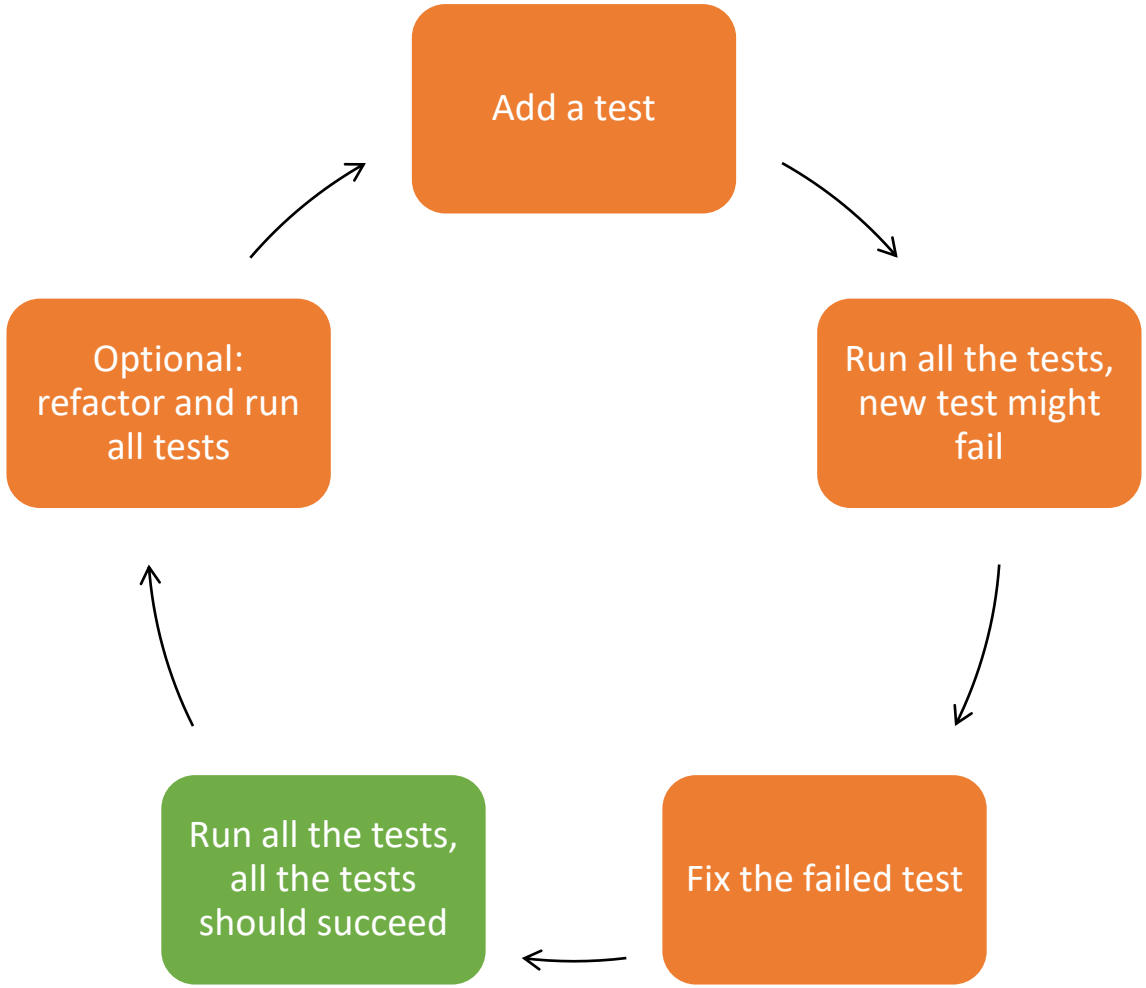
Implementing a test

- Strategies:
  - Fake it till you make it (Mocking)
  - Obvious implementation
  - Triangulation
    - Do not limit yourself to the happy path
    - Verification by elimination of non-happy paths

Triangulation in TDD

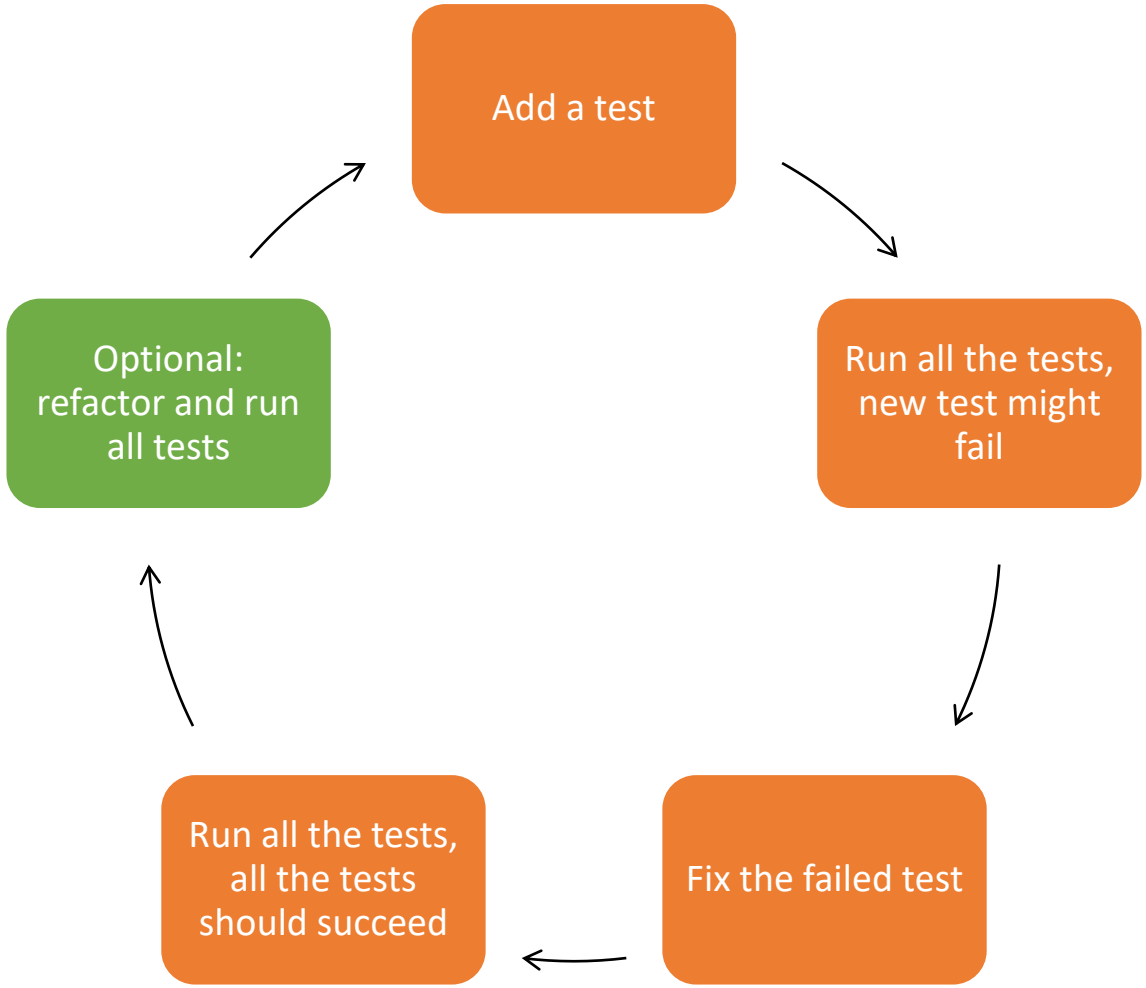
- Case: Testing a method which accepts parameter between 1 and 10
  - Write a test with parameter == 5
  - Write a test with parameter == 0 or 11
  - Write a test with parameter == null
  - Write a test with parameter == “Five”

TDD Cycle



AXXES

TDD Cycle



AXXES

Refactoring

- Rewrite existing code by:
  - Simplifying
  - Abstractions
  - Other small changes
- Alter the internal structure of a code block, w/o changing the external behavior
- Validate nothing broke by running all the tests again
- Arguably the most important step in the cycle

Clean code == happy team

AXXES

TDD in modern development

–

Modern problems require modern solutions

TDD in modern development

- TDD is not a “new” concept
  - Related to Extreme Programming (1999)
- Shift in the industry to TDD
  - It’s a philosophy
    - Applicable to new and legacy projects
    - Implement it in the way your team is comfortable

AGILE SOFTWARE DEVELOPMENT

–

Agile practices are defined as discovering requirements and building solutions through the collaborative effort of self-organizing and cross-functional teams (and their clients)

~ Wikipedia

Agile

- Iterative development
- Each iteration is called a "Sprint"
  - Duration varies, 2 weeks is pretty much the standard
- Hold a stand-up meeting every day
  - Should be as brief as possible
  - Outline for each member:
    - Achievements the previous day
    - Goals for today
    - Blockers
  - Bigger issues are talked about post stand-up
- Every Sprint starts with a Retrospective ...
  - Reflect about the previous sprint
- … followed by a planning session
  - New requirements, long-term and short-term plans and action points

AXXES

Agile and TDD

- As requirements change, so does code
  - TDD and Agile are both iterative processes
  - Both have short feedback loops
  - Rewriting code with tests is a lot less painful
- Working agile goes hand in hand with doing TDD
  - TDD guarantees working code
  - While coding, you keep requirements in mind
    - You'll pick up edge cases and prevent bugs earlier rather than later
  - Confidence in the code
    - Knowing the state of the code allows making accurate predictions

Continuous Integration / Continuous Deployment (CI/CD)

–

Combining CI and CD

Continuous Integration (CI)

–

Merging every team member's progress on a main branch several times per day

~ Wikipedia

Continuous Deployment (CD)

–

Automatically deploying every team member's progress on a staging or production environment several times per day

~ Wikipedia

CI/CD

- Automatically keep the main/master branch up to date
- Tests are key
  - Ran at every commit on a PR (CI)
  - Ran after merging the PR (CI)
  - Ran before every deploy (CD)
- Automatically deploy a new version and retire the old version
  - Pipelines (Jenkins, Azure, Airflow, ...)
- Staging vs Production
  - Different environments before a version is released

Practical TDD

–

Implementing the cycle

Requirement

- Write a method `String greet(String name);` that does String interpolation with a simple greeting.

- Input: `name = "Bob"`
- Output: `"Hello, Bob"`

Requirement

- Handle `nulls` by introducing a stand-in.

- Input: `name = null`
- Output: `"Hello, there."`

AXXES

Requirement

- Handle shouting. When `name` is all uppercase, then the method should shout back to the user.

- Input: `name = "JEFF"`
- Output: `"HELLO JEFF!"`

Requirement

- Handle two names of input. When `names` is an array of two names, then both names should be returned. (You can change the signature to `String greet(String... names);` to retain backwards compatibility)

- Input: `names = ["Jill", "Jane"]`
- Output: `"Hello, Jill and Jane."`

Requirement

- Handle three names of input. When name is an array of three names, then all names should returned.

- Input: `names = ["Amy", "Brian", "Charlotte"]`
- Output: `"Hello, Amy, Brian, and Charlotte."`

Requirement

- Allow mixing of normal and shouted names by separating the response into two greetings.

- Input: `names = ["Amy", "BRIAN", "Charlotte"]`
- Output: `"Hello, Amy and Charlotte. AND HELLO BRIAN!"`

Requirement

- If any entries in `name` are a string containing a comma, split it as its own input.

- Input: `names = ["Bob", "Charlie, Dianne"]`
- Output: `"Hello, Bob, Charlie, and Dianne."`

Going a step further (Extra)

–

Integration Testing tools

@SpringBootTest

- Annotation on a JUnit test to allow Spring functionalities
- Starts up the Spring Application Context
  - Allows you to autowire dependencies to write integration tests
  - Using the actual dependencies your service/components need
  - Minimal mocks, keep mocked beans to a minimum
    - Reduce the effectiveness of the integration test
  - Less control over the components
    - Good thing in this case
    - Control Input/Output using database assert tools/email clients/rest assertion tools

TestContainers

- Library to allow starting up docker containers for tests
  - Deep integration tests with other services
    - Only if your organization owns the other service
  - Easy setup for integration tests
    - Database
    - ElasticSearch
    - …
- Running on your machine
  - Allows you to really dig into I/O

Selenium

- Browser automation
  - Frontend-testing
  - Webcrawling
- Chromium wrapper
- Integration with JUnit makes it very powerful
  - Automated e2e tests

HoverFly

- Proxy
- API Simulation
  - Deep HTTP call validation
  - Powerful matching strategies
    - HTTP Headers
    - Request/Response bodies
    - Query Parameters
  - Reusable!
  - Simulate network latency, rate limits or other failures
- Microservices architecture
  - Development
    - While building one service, mock the responses of another
    - Actual HTTP Calls
  - Testing
    - Assert HTTP Calls

RestAssured

- Replaces @SpringBootTest for Rest Layer
  - Tests ran against a running service
    - Docker in CI!
    - Staging environments
- Performs actual HTTP requests
  - Request/response validation
  - Easily setup a blackbox testing framework for integrating projects

“The best day to start writing tests was yesterday”

Thank you!

Questions?
cel.pynenborg@axxes.com

Entrepotkaai 10A,
2000 Antwerpen

Leonardo Da Vincilaan 9,
1930 Zaventem

Ottergemsesteenweg Zuid 808
bus 300 , 9000 Gent

T +32 3 23499.58
info@axxes.com

www.axxes.com