Testing+

Nikolai Denissov

0. ToC

1. Testing (I talk)
2. Example (I show & talk, you talk and guess)
3. Tooling (I talk again)
4. Real world project example (I show, you investigate)
5. Discussion (everybody ~~dance~~ talk)

1. Testing

1.1 What is testing?

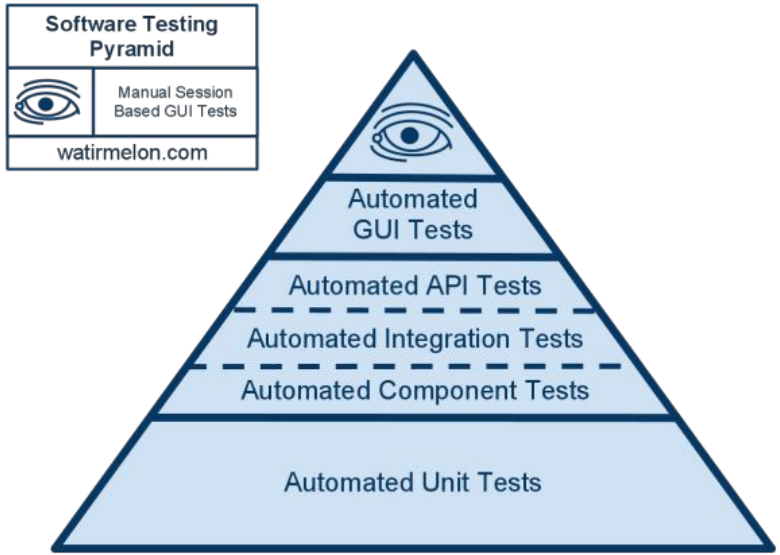
- Practice that allows to **verify** and **validate** the software.

Verify is for ensuring it works as it should.

Validate is for confirming the quality of the software (that it does not crash and burn of the very first use).

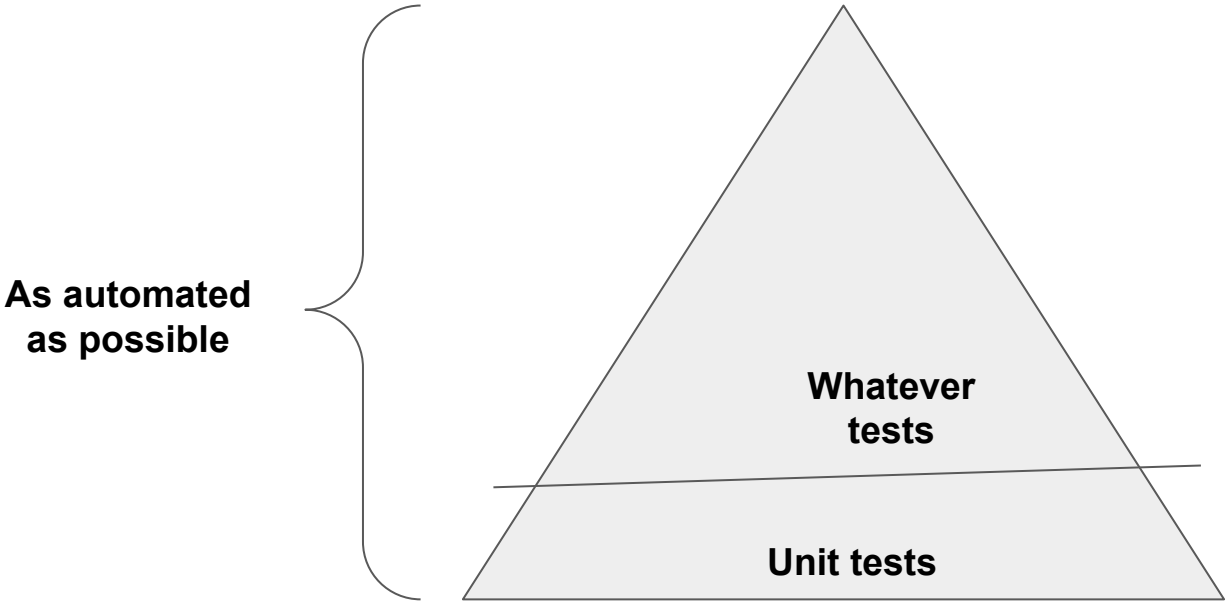
1.2. Testing flavours (1)

- Testing pyramid by the book



1.3. Testing flavours (2)

- Testing pyramid



1.4. What and how much to test?

What to test?

- Methods/functions
- Units (several methods/functions together)
- Service Interactions (Integration)
- UI
- ...

What to take into account?

- Effort
- Service expected lifespan
- Execution time

1.5. When to make tests?

- Whenever, as long as done

2. Example*

2.1. Task as a user story (yeeeey!)

As a researcher, I want to classify animals from the “Cat” family (*Felidae*).

We build an API endpoint, that takes a name of the animal from the “Cat” family as an input, and responds with a corresponding classification.

AC:

We will “create” a “transformer” tool.

2.2. Task + AC

As a researcher, I want to classify animals from the “Cat” family (*Felidae*).

We build an API endpoint, that takes a name of the animal from the “Cat” family as an input, and responds with a corresponding classification.

Acceptance Criteria aka AC:

1. CAT is a domestic animal.
2. TIGER is a wild animal.
3. …

Acceptance criteria are mostly about **verify**.

2.3. Task + better* AC

AC:

- ***Given*** a “CAT”,

  When the transformer is called,

  Then the result is “domestic animal”.
- ***Given*** a “TIGER”,

  When the transformer is called,

  Then the result is “wild animal”.
- ***Given*** no animal,

  When the transformer is called,

  Then the result is “no animal”.
- ***Given*** any animal,

  And the animal is neither “CAT nor “TIGER”,

  When the transformer is called,

  Then the result is “unknown animal”.

2.4. Task + better* AC + quality parameters

How to ensure the proper quality:

- Common sense
- Peer review
- Team internal quality parameters aka DoD
- Quality verification tools

Continuing with the example:

- Access roles must be defined (also if none)
- For any endpoint, the @Swagger annotation must be provided to generate the docs
- Docs must contain basic use cases (at least the “sunny path”)
- Error handling, at least 4xx and 5xx must be provided
- Unit/Integration/e2e tests pass (from DoD)
- Response time is less than 2ms for valid input

2.4. OK, let’s code (Scala 2.13 styled)

```scala
def felidaeMethod(input: String): String = {
  if (input.nonEmpty) {
    input match {
      case "CAT" => "domestic animal"
      case "TIGER" => "wild animal"
      case _ => "unknown animal"
    }
  } else {
    "no animal"
  }
}
```

What is the minimal amount of test cases is reasonable to have here?

- 0, it won’t compile even
- 2
- 4
- 5

2.5. Task change

AC:

- ***Given*** a “LION”,

  When the transformer is called,

  Then the result is “wild animal”.

2.5. OK, let’s code again (Scala 2.13 styled)

```scala
def felidaeMethod(input: String): String = {
  if (input.nonEmpty) {
    input match {
      case "CAT" => "domestic animal"
      case "TIGER" | "LION" => "wild animal"
      case _ => "unknown animal"
    }
  } else {
    "no animal"
  }
}
```

What happens to the existing tests?

How many test cases we should change?

- 0
- 1
- 2
- 5

2.6. OK, error handling (Scala 2.13 styled)

```
override def onClientError(request: RequestHeader,
                           statusCode: Int,
                           message: String): Future[Result] = {
  logger.debug(
    s"onClientError: statusCode = $statusCode, uri = ${request.uri}, message = $message")

  Future.successful {
    val result = statusCode match {
      case BAD_REQUEST =>
        Results.BadRequest(message)
      case FORBIDDEN =>
        Results.Forbidden(message)
      case NOT_FOUND =>
        Results.NotFound(message)
      case clientError if statusCode >= 400 && statusCode < 500 =>
        Results.Status(statusCode)
      case nonClientError =>
        val msg =
          s"onClientError invoked with non client error status code $statusCode: $message"
        throw new IllegalArgumentException(msg)
    }
    result
  }
}
```

What is the minimal amount of test cases is reasonable to have here?

- 0, it won't compile even
- 2
- 4
- 5

2.7. Unit vs. Other tests

- It’s mostly the scope, that matters

2.8. What about the Testing Frameworks?

- Implementation language specific stuff: Play, ScalaTest, Jest, Cypress, Selenium, etc.

2.9. How often to run tests?

- As often as possible...

3. Tooling

3.1. Automation

- How to run the tests often?
- How to run the tests with the least effort?
- When to use automation (and when not to)?

3.2. Automated Quality Analysis Tools

- Code static analysis tools I
  - IDE itself or via extensions
  - linters
  - the compiler
- Code static analysis tools II
  - Sonar
  - Black Duck
  - Snyk
  - Fossa
  - Etc.

Take a look at GitHub student pack: https://education.github.com/pack

3.3. Automation Deployment Tools

- Jenkins
- GitHub Actions
- GitLab CI
- Travis/Circle/Whatever CI
- Cloud-specific ones (Azure, AWS, GCP)

4. Real World Example

4.1. Intro

https://github.com/Aalto-LeTech/aplus-courses

4.2. Tests (1)

- Unit tests: https://github.com/Aalto-LeTech/aplus-courses/blob/master/src/test/java/fi/aalto/cs/apluscourses/utils/ArrayUtilTest.java
- Platform tests (contract testing, in a way): https://github.com/Aalto-LeTech/aplus-courses/blob/master/src/test/java/fi/aalto/cs/apluscourses/intellij/services/PluginSettingsTest.java
- API/Integration tests (against the external platform): https://github.com/Aalto-LeTech/aplus-courses/blob/master/src/test/java/fi/aalto/cs/apluscourses/integration/ApiTest.java

4.3. Tests (2)

- Concurrency testing: https://github.com/Aalto-LeTech/aplus-courses/blob/master/src/test/java/fi/aalto/cs/apluscourses/utils/PostponedRunnableTest.java
- Manual testing: https://github.com/Aalto-LeTech/aplus-courses/blob/master/TESTING.md
- e2e testing:

  (private access) Forte DSL example

4.4. Tools

- **Sonar**: https://sonarcloud.io/project/issues?resolved=false&id=Aalto-LeTech_intellij-plugin
- **Snyk**: https://snyk.io/test/github/Aalto-LeTech/intellij-plugin?targetFile=build.gradle&tab=dependencies
- **GitHub Dependabot**: https://github.com/Aalto-LeTech/aplus-courses/pull/962
- **GitHub Actions**: https://github.com/Aalto-LeTech/aplus-courses/blob/redo-platform-tests-scala/.github/workflows/build.yml
- **Fossa**: https://app.fossa.com/projects/git%2Bgithub.com%2FAalto-LeTech%2Fintellij-plugin/refs/branch/master/141352a6ce81d4a5fa35e7066dc660644eef0b51/issues/licensing/2534509?filter=policy_flag&revScanId=34721709

Drinks for stickers game

At any point during any discussion you can give away your “D” sticker to the person who speaks most actively or suggests best ideas.

For any 2 (two) “D” stickers you can get a ~~warm~~ soft drink
as a reward for active participation, as talking makes you thirsty.