Testing & CI/CD Automation Tools

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?

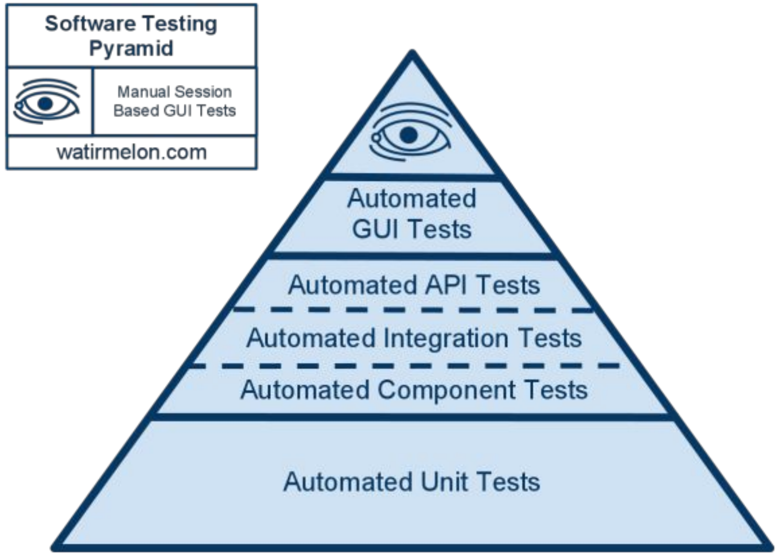
- Practice that allows to **verify** and **validate** the software.

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).

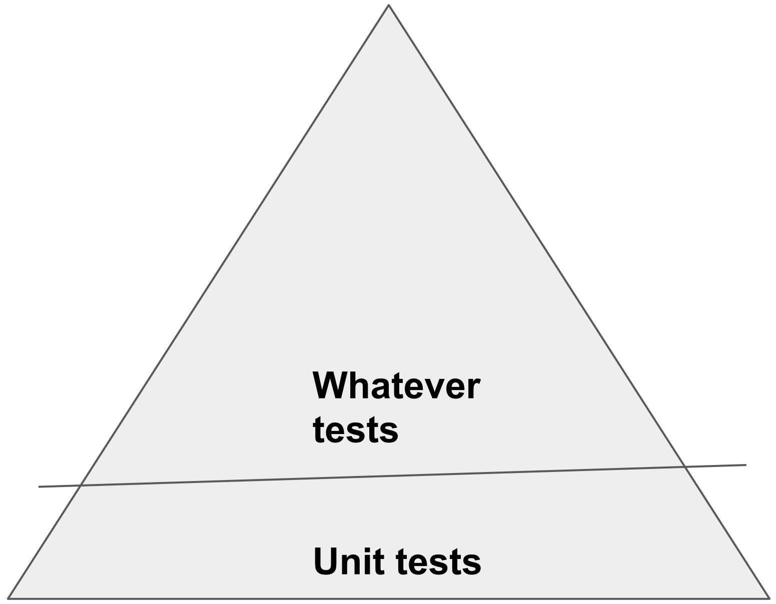
1.2. Testing flavours (1)

- Testing pyramid by the book



1.3. Testing flavours (2)

- Testing pyramid



1.4. What and how much to test?

What to test?

- Methods
- Units (several methods together)
- Component
- Services
- ...

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 names from the “Cat” family (*Felidae*).

What type of data are we working with.

AC:

We will “create” a “transformer” tool.

2.2. Task + AC

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,

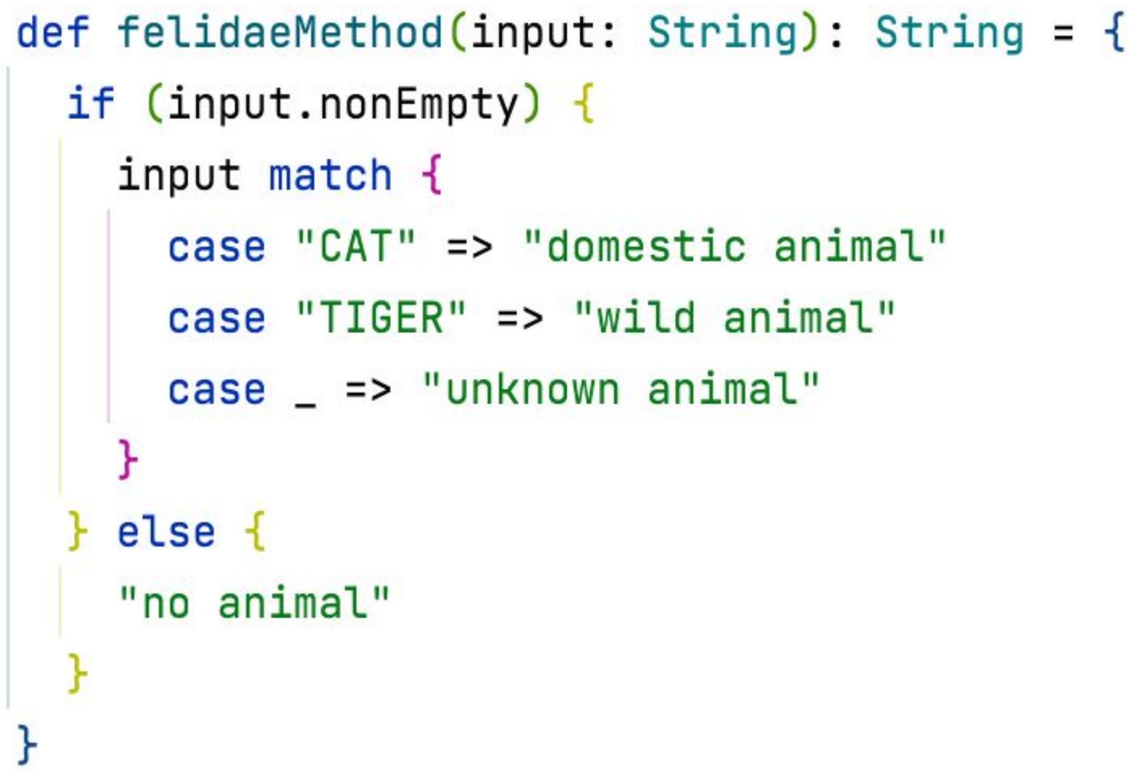
  Then the result is “no animal”.
- ***Given*** any animal,

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

  When the transformer is called,

  Then the result is “unknown animal”.

2.4. 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"
  }
}
```

2.5. Quiz!!1

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

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

2.6. Task change

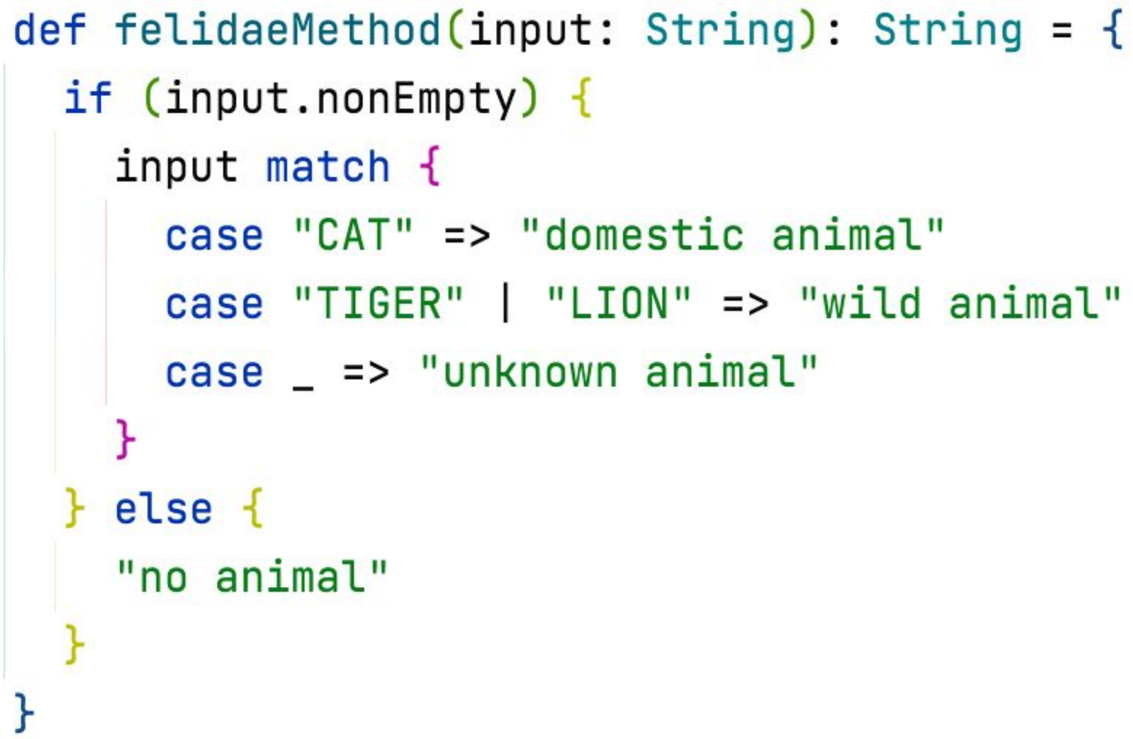
AC:

- ***Given*** a “LION”,

  When the transformer is called,

  Then the result is “wild animal”.

2.7. 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"
  }
}
```

2.7. Quiz!!1

What happens to the existing tests?

How many test cases we should change?

- 0
- 1
- 2
- 5

2.8. Unit vs. Other tests

- It’s mostly the scope, that matters

2.9. What about the Testing Frameworks?

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

2.10. 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,
  - 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 CIs~~
- 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: https://github.com/Aalto-LeTech/aplus-courses/blob/master/src/test/java/fi/aalto/cs/apluscourses/intellij/services/PluginSettingsTest.java
- API 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: https://github.com/Aalto-LeTech/aplus-courses/blob/master/src/e2e/kotlin/fi/aalto/cs/apluscourses/e2e/fixtures/CommonFixtures.kt

4.4. Tools

- Sonar: https://sonarcloud.io/summary/new_code?id=Aalto-LeTech_intellij-plugin
- Snyk: https://snyk.io/test/github/Aalto-LeTech/intellij-plugin?targetFile=build.gradle&tab=dependencies
- GitHub Actions: https://github.com/Aalto-LeTech/aplus-courses/blob/master/.github/workflows/build.yml

Some curious read

- 12 Factor app: https://12factor.net/
- Agile Manifesto: https://agilemanifesto.org/