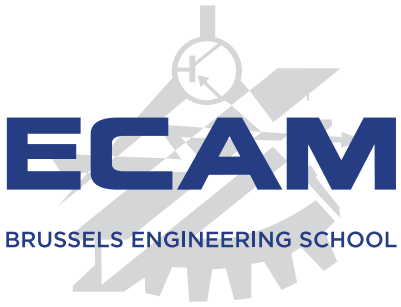


I402A Software Architecture
and Quality Assessment

Project 1

Test-Driven Development and Continuous Integration

Sébastien Combéfis

Fall 2019

This work is licensed under a Creative Commons Attribution – NonCommercial – NoDerivatives 4.0 International License.

Objectives

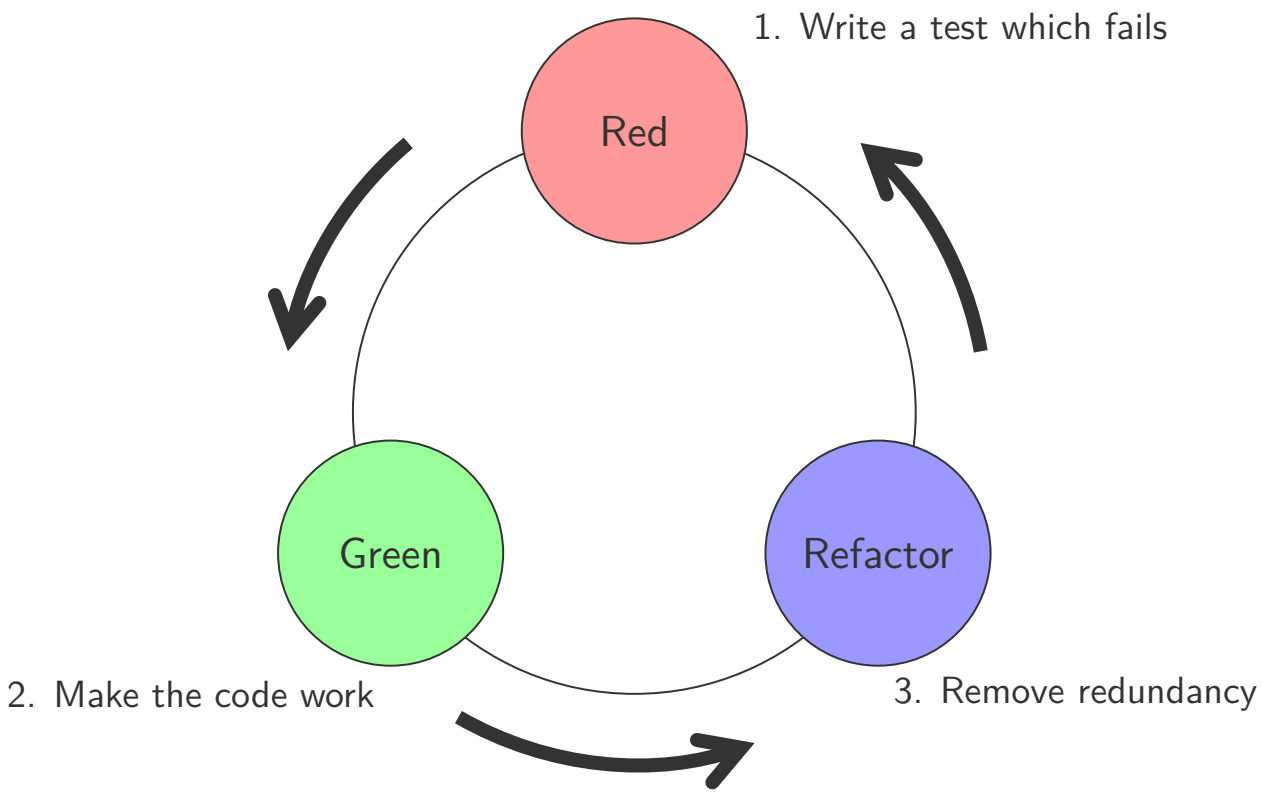
- Introduction to TDD and CI based development
  - Test-Driven Development is about testing the code regularly
  - Continuous Integration is about automated checks and builds
- A realistic roleplay to practice TDD and CI
  - Working on group on the definition of a project
  - Starting to implement it, following a TDD and CI approach
  - Stopping and taking back another project on its way


TDD and CI

Test-Driven Development (TDD)

- Development driven by tests and continuous refactoring

Two excellent XP practices to improve software quality



Continuous Integration (CI)

- Integrate code into a shared repository several times a day

  Automated build to ensure that several checks are OK

- Main goal is to solve problems quickly

  Quick detection and better localisation for a fast fix

"Continuous Integration doesn't get rid of bugs,
but it does make them dramatically easier to find and remove."

— Martin Fowler

Refactoring

- Code transformation which preserves his behaviour

  "A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour" —Martin Fowler

- Objectives
  - Make it easier to add new code
  - Improve the design of existing code
  - Better understand a code
  - Make a code less annoying

Software Quality

- Two ways to improve software quality
  - Creating more unit tests to get the best code coverage
  - Requiring developers to make more *"tests by hand"*
- Positive dynamics of software improvement
  1. Choosing quality criteria
  2. Stating how to measure/evaluate there criteria
  3. Defining what is a good quality software

Standard Metric

- Measuring a criterion to better understand it

  Getting a value to be able to evaluate, compare, etc.

- Making measures so that to be able to control

  Evaluate and improve the quality depending on the measure

- Not easy to determine what to measure

  Measure and criteria to evaluate choices is important

Tools

Source Code Management

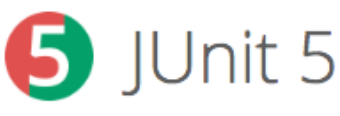
- Managing the source code of a project is very important
  - Backup strategy and versioning for a history of modifications
  - Possibility for the developer to propose several revisions a day
  - Team work and possibility for development branches
- Several possible systems can be used to manage source code

  Centralised or not, public or private, etc.

Unit Testing

- Automated tool to run tests on code
  - Can be executed on compiling code
  - A set of tests is executed on each function
  - Necessary to have (in)formal specifications for each function
- Several tools do exist, in particular the xUnit serie

  JUnit, PyUnit, PHPUnit, NUnit, CUnit, etc.



Build Automation

- Managing a project build with reporting
  - Compiling code source into binary code
  - Packaging binary code with dependencies
- Can take the form of a utility or server

  Different levels of automation can be provided

Code Linting

- Analysing the source code to flag several faults
  - Can identify programming errors and bugs
  - Can flag stylistic errors and suspicious constructs
- A bunch of existing tools that are highly configurable

  Check a specific style, search for some misconstructs, etc.


Roleplay

Phase I: Set up

- Imagining a small project idea to be implemented in Java
  - Size of the project should be 10–15 classes
  - Must use at least one design pattern
  - Can be a command line tool, or have a user interface, etc.
- Analysing the project and producing several documents
  1. Short description of the project and the problem that is solved
  2. Realising use cases diagrams and drafting a class diagram
  3. Defining or choosing a coding convention
  4. Choosing quality criteria and metrics to assess them

Phase II: Implementation

- Starting to implement the defined project
  - Using a source code management system
  - Following an agile approach with continuous integration
  - Setting up and configuring Jenkins
- Maintain the documentation during the project development
  - Make commits with relevant messages
  - Maintain a changelog for the main steps
  - Document new features proposals, with the analysis

Phase III: Project Switch

- STOP working on your project and switch to another one

  The switching procedure will be imposed

- Make an audit of the new affected project
  - Analyse the documents attached to the project
  - Make a point about the quality of the project
  - Continue to work on it to improve its quality

References

- Graham Wright, *Software Delivery, DevOps and CICD for the uninitiated: Series Intro*, July 4, 2018.
  https://medium.com/@gwright_60924/software-delivery-for-beginners-series-intro-751b90fbe078
- Tahin Rahman, *The mindset behind Test Driven Development (TDD)*, April 22, 2018.
  https://medium.com/@tahins/the-mindset-behind-test-driven-development-tdd-b02deccbff81
- Brenn, *Noobs Guide: Continuous Integration & Continuous Delivery*, January 29, 2019.
  https://medium.com/@brenn.a.hill/noobs-guide-continuous-integration-continuous-delivery-continuous-deployment-d26ac4f2beeb
- yasiru nilan, *Introduction to Continuous Integration & Continuous Deployment*, June 4, 2018.
  https://medium.com/pulseque/introduction-to-continuous-integration-continuous-deployment-38c3ebc07221

Credits

- Logos pictures from Wikipedia.
- Michael Wyszomierski, December 12, 2005, https://www.flickr.com/photos/wysz/72999649.
- Carlos Lorenzo, December 6, 2010, https://www.flickr.com/photos/carlos_lorenzo/5752177729.
- Apollo Scribe, May 8, 2016, https://www.flickr.com/photos/apolloscribe/26789464482.