Agile Software Development

L04 – Code refactoring

Ezequiel Scott
ezequiel.scott@ut.ee
Agenda

• Recap
• Organizing User Stories
• Refactoring
• Code Smells
Recap

• Software development processes, agile terminology
• Current state of Agile worldwide
• Test-driven Development (TDD)
• Handling requirements in ASD
  • Writing user stories
• Organizing User Stories
  • Kanban Board
Agenda

• Recap

• Organizing User Stories

• Refactoring

• Code Smells
Organizing User Stories

How to envisage the entire product or service as a series of tasks which the user completes?

Two mechanisms:

• **Kanban board**

• **User Story Mapping**
Kanban Board

- It visualizes work and the process it goes through
- Generally more sophisticated than “mere” task boards
Kanban board and WIP

WIP → the number of work items you have going at the same time

- The WIP limit isn’t a strict rule; it’s a trigger for discussions

WIP too high = work idle

WIP too low = people idle

From KANBAN IN ACTION by MARCUS HAMMARBERG and JOAKIM SUNDEÑ, Copyright 2015.
The 3 principles of Kanban

• **Visualize what you do today** *(workflow)*: seeing all the items in context of each other can be very informative → Kanban Board

• **Limit the amount of work in progress** *(WIP)*: this helps balance the flow-based approach so teams don’t start and commit to too much work at once

• **Enhance flow**: when something is finished, the next highest thing from the backlog is pulled into play

Kanban promotes continuous collaboration and encourages active, ongoing learning and improving by defining the best possible team workflow.
Organizing User Stories

How to envisage the entire product or service as a series of tasks which the user completes?

Two mechanisms:

• Kanban board

• User Story Mapping
Story Mapping

• **Story mapping** consists of ordering user stories along two independent dimensions
  
  • horizontal axis → **order of priority** (or “the order in which you would describe activities to explain the behaviour of the system”)
  
  • vertical axis → it represents increasing sophistication of the **implementation**

• The first horizontal row represents a “walking skeleton”, a barebones but usable version of the product

• Working through successive rows fleshes out the product with **additional functionality**
User Story Mapping

https://plan.io/blog/user-story-mapping/
User Story Mapping
https://plan.io/blog/user-story-mapping/
User Story Mapping

https://plan.io/blog/user-story-mapping/
User Story Mapping

[Link to User Story Mapping](https://plan.io/blog/user-story-mapping/)
User Story Mapping
https://plan.io/blog/user-story-mapping/
User Story Mapping

https://plan.io/blog/user-story-mapping/
User Story Mapping

https://plan.io/blog/user-story-mapping/
Story Mapping
https://plan.io/blog/user-story-mapping/
Story Map in 7 steps

1. Frame the journey
2. Build your story backbone
3. Identify and group activities
4. Break large tasks into subtasks
5. Fill in the blanks
6. Prioritize tasks and subtasks (but leave your backbone as is)
7. “Slice” groups of tasks into iterations

https://plan.io/blog/user-story-mapping/
Story map – example (2)
Story map – Example (1)
Release vs Iteration planning

- A **release** is made up of one or more **iterations**
- **Release planning** refers to determining a balance between a projected timeline and a desired set of functionality
- **Iteration planning** refers to selecting stories for inclusion in this iteration
- The customer team and the developers are both involved in release and iteration planning
Release planning

Product vision drives product roadmap
Product roadmap drives release plans

Release plan establishes the iterations

Iteration plans schedule feature development
Prioritized features delivered by user stories

Tasks created to deliver user stories

Release Plan

Iteration 0 | Iteration 1 | Iteration 2 | Iteration 3 | Iteration n

Feature A (User Story 1) | Feature A (User Story 2) | Feature B (User Story 3) | Feature C (User Story 4) | Feature D (User Story 5)

Task A | 5 Hours
Task B | 8 Hours
Task C | 4 Hours
Task D | 12 Hours

User perspective
Developer perspective
Planning a release

• Give **priority** to the user stories, which includes:
  • The desirability of the features (customer)
  • Technical risk, dependencies (developers)
• Cost → The cost of a story is the **estimate** given to it by the developers
  • How much work can the team complete in one iteration?
• **Effort estimation → next lecture!**
Agenda

• Recap
• Organizing User Stories
• Refactoring
• Code Smells
Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.

Martin Fowler, Kent Beck (2012)
Refactoring – common pitfalls

• Refactoring does not mean:
  • rewriting code
  • fixing bugs
  • improve observable aspects of software such as its interface

• Refactoring in the absence of safeguards against introducing defects (i.e. violating the “behaviour preserving” condition) is risky

• Safeguards include aids to regression testing including automated unit tests or automated acceptance tests
Refactoring – expected benefits

• Refactoring improves objective attributes of code (length, duplication, coupling and cohesion, cyclomatic complexity) that correlate with ease of maintenance

• Refactoring helps code understanding

• Refactoring encourages each developer to think about and understand design decisions, in particular in the context of collective ownership / collective code ownership

• Refactoring favors the emergence of reusable design elements (such as design patterns) and code modules
Refactoring – When?

✓ Refactor when you **add a function**
✓ Refactor when you need to **fix a bug**
✓ Refactor as you do **code review**

✗ Do not refactor when it is **easier to start from the beginning**
✗ Do not refactor when you are **close to the deadline**
But...

In 2010, Soetens and Demeyer found surprisingly little correlation between refactoring episodes, as identified by version control logs, and decrease in cyclomatic complexity

• methodological issues, or

• gap btw research and common practices ???

Studying the Effect of Refactorings:
 a Complexity Metrics Perspective

Quinten David Soetens and Serge Demeyer
Department of Mathematics and Computer Sciences
University of Antwerp
{quinten.soetens, serge.demeyer}@ua.ac.be
Refactoring opportunities – How?

Refactoring opportunities can be motivated by:

- Casual inspection
- Code reviews
- Tools
- Bad smells
Bad smells

• Code duplication

• Class / method organization
  • Large class, Long method, Long parameter list, Lazy class, Data class...

• Lack of loose coupling or cohesion
  • Inappropriate intimacy, Feature envy, Data clumps...

• Too much or too little delegation
  • Message chains, Middle man...

• Comments

• ...

https://refactoring.guru/
https://refactoring.com/catalog/
Life-cycle of a smell
Automated tools

• Many languages have IDEs which **automate many common refactoring**s

• **Such tools aren't essential** - I often work in programming languages without tool support, in which case I rely on taking small steps, and using frequent testing to detect mistakes.

  -- Martin Fowler
Problems with code smells

- Only a good recipe book and nothing more
- It is not always easy
- It is not always useful
- Most of them are specific to Object Oriented Programming
  - Declarative programming? → Handout Practice #3!!