# Software Projects

## How to make student projects efficient and enjoyable

13.12.2022


Juhani Välimäki

# How Softala etc. school projects differ from the usual ones in business

- The teams are comparably bigger than would be set in real life
- Even if all ICT work is basically learning or finding out, here even more so
- There is usually no product running in ”production environment”
- Other members need your contribution faster than normally (the architecture, folder and file structure, frontend routing,  API end points, … )
- Students are not working full time for this project nor this tech stack
- Students do more roles than SW developers *typically* (incl. Product Owner, Product and UX design, customer

# The goals in our projects

Basically you want to create a “well-oiled machine” or  “heavy but steady train” that slowly but inevitably starts to produce results, **especially towards the end of the project**

- It’s also important to have enjoyable challenging environment, where the pressure is manageable and will not cause stress
- We want to make the mistakes here you don’t want to do in your first job!
- Most important is not the product, but developing the process/project so that your next project would start suberbly
  - Of course if that is really successful you have a good version of the product at the end of the project. Good sound first version of the product but not the best.
- If all process and project factors are sound, your team might produce more product results on the last 2-3 weeks than a immature project will produce in the whole 16 weeks!

# These could make our projects efficient & enjoyable

- Scrum understood, philosophy digested and followed. Teams can get fully Scrum-autonomous
  - Also some visually pleasant and intuitive Scrum tool used properly
- Communication and reactions
  - Communication tool and channels, e.g. 10 well-thought channels in Teams/Slack.
  - Meeting and file naming. Reactions to someone working at Saturday night.
- Agile documentation
- Architecture and sound SW philosophy
- Technical skills
- Git process and review skills
- Spreading the skills and knowledge across the team(s)
- Courage
- Lean prototyping approach to product development (avoid the “analysis paralysis”)
- *Restrospective of all the above (“inspect and adapt”)*

# Scrum

- E.g.
  - https://github.com/valju/scrum_learning/tree/master/ScrumDrawings git pull the repo and open this folder
  - https://github.com/valju/scrum_learning/blob/master/ScrumDrawings/02_AboutScrumRolesAndTasks.pdf
  - https://github.com/valju/scrum_learning/blob/master/ScrumDrawings/03_ScrumMeetingTypes_ActionsAndAttendees.pdf
  - https://github.com/valju/scrum_learning/blob/master/ScrumDrawings/04b_ScrumSchedule_for_OneWeekSprint_TypicalMiddleOfWeekCloseAndStart.pdf
- Scrum understanding and knowledge has to go to a level where at the end nobody even thinks we are using Scrum. Everybody automatically does all things right
- In the beginning, on the otherhand, it's better to recap e.g. the meeting type before the meeting. What's done there and what's NOT done there

# Architecture and SW design principles

- E.g.
  - https://github.com/haagahelia/swd4tn023/blob/master/06_ohjelmistoarkkitehtuurit_ja_patternit/SoftwareArchitecturesAndPatterns.pdf
    - The principles of sound architecture, choosing appropriate architecture, principles in SW design, … basically philosophy = ways to think automatically
  - https://github.com/haagahelia/swd4tn023/blob/master/06_ohjelmistoarkkitehtuurit_ja_patternit/uuid.pdf
    - Just one technical example related to architectures
- When you digested the correct philosophies you tend to make correct choices without thinking!

# Git process

- E.g.
  - https://github.com/haagahelia/swd4tn023/tree/master/03_infra_ja_automaatio/BasicGitBranchingMinimumForBigTeams
- Four level professional model? Or the simplified two level model that has been successfully used in many of our student projects, as it fits possible better our project nature, look at the beginning of this slide set

# Agile documentation

- E.g.
  - https://github.com/haagahelia/swd4tn023/blob/master/06_ohjelmistoarkkitehtuurit_ja_patternit/documentation_principles_for_sw_projects.pdf
- Read those 8 slides above
- Also in this important topic we have the non-simple thinking:
  - While learning in school we might do heavy documentation, where the process of creating is important
  - While doing real project we emphasize maintainability and agility also for documentation

# The long version of the *project lessons learned*

- Based on experiences from tens (or hundreds if counting thesis projects) of student projects
- Basically nothing earth shaking, but these things require thinking so that we can avoid these pitfalls from the start.
- http://myy.haaga-helia.fi/~valju/perma/sw_project_courses/Software%20Development%20projects.pdf