Software Projects

How to make student etc. projects efficient and enjoyable

16.1.2024

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 SW firms
- Even if ICT work often is learning and research, here even more so
- We usually have 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, access management, …)
- Students are not working fulltime for this project nor this tech stack
- Students do more roles than SW developers *typically* do. (incl. Product Owner, Product and UX design, customer contact)

The goals in our projects

Fundamentally 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 prolonged stress
- We want to make the mistakes here that you don’t want to do in your first job!
- You want to start your career project later with skills, knowledge, experience and a slightly thicker skin.
- Thus the only important thing 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!
  - The product is also a test of the project process’ maturity

These factors make our project efficient & enjoyable

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

Scrum running smoothly

- E.g.
  - https://github.com/valju/scrum_learning/tree/master/ScrumDrawings git pull that 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 finally nobody even is thinking that we are using Scrum. Everybody just automatically does all things right way.
- In the beginning, on the other hand, it's better to recap e.g. the meeting type before each meeting. What's done in that kind of meeting and what's NOT done there

Architecture and SW design principles

- E.g. the principles of sound architecture, choosing appropriate architecture, principles in SW design, … basically philosophy = ways to affect your thinking automatically
  - https://github.com/haagahelia/swd4tn023/blob/master/06_ohjelmistoarkkitehtuurit_ja_patternit/SoftwareArchitecturesAndPatterns.pdf
- Just one technical example related to architectures
  - https://github.com/haagahelia/swd4tn023/blob/master/06_ohjelmistoarkkitehtuurit_ja_patternit/uuid.pdf
- When you have 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? (Link in presentation above)
- 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 pics and docs in the link above)
- How our school projects differ was handled earlier:
  - Often no running production version.
  - Often no test branch for separate test team.
  - Team needs our common parts contribution published faster, and not hidden in a separate branch for longer.
    - Fast integration and sharing even more important than avoiding all problems at any means

Agile documentation

- 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. This is an example of 2 ‘conflicting’ approaches
  - While learning in school we might do heavy documentation, where the learning process steps are important. Heavy
  - While doing real project we emphasize maintainability and agility also for documentation. Lighter

- A bit different but related important hint:
  - **Make your own notes!** While e.g. meeting customer, make a text file for yourself, name it well and save to good folder before the meeting. Put there date, which customer was present and then all of your notes. Fast written, format not important, 100% correctness is not important. In every project we happen wonder what was said in the meetings. Plus we get all the future development ideas written down when they occur. Shows respect for the customer too.

(extra) 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 and reflecting so that we can avoid these pitfalls from the start.
- Not a Pulitzer Prize winner, but certainly lists points that some projects could have used in past.
- Software Development projects.pdf