Juhani Välimäki ver 2019-08-21

Software Development projects

Learning from tens of earlier projects

The earlier projects vary from excellent results to not that spectacular ones. But even for the most successful proejcts, many aspects could have been improved. It's interesting, that many of the points of improvements are the same, through the years, through different technologies and products.

I tried to remember as many core points as possible. This document will be constantly updated and it won't ever be ready.

Disclaimer 1: Not all here applies to all of us, nor is the idea to find the target for blaming. The idea is to help the projects achieve even better results.

Disclaimer 2: Students have different backgrounds – from no work experience to years of experience, having completed certain previous studies and project courses or not, from e.g. running authentic Scrum earlier to having Scrum mentioned on a traditional project management course.

Note: Read this document as fuel for your thoughts, not as sacred word or the Truth. Every project is different. The projects may vary greatly in nature. E.g. in many projects the customer gives just an idea and relies on the students to do the whole product development. In some others customer gives even detailed product design and expects just the implementation.

Old thinking of what SW dev project is

Each semester there will be some students for whom a real **agile customer product development project** is a totally new concept.

Students might have done "projects" before, but those were not really projects in the official sense of the word. E.g. teacher gave students a task to program, **ready-made** and **unchanging requirements** in the beginning of the "project" and gave a deadline when the results need to be demonstrated. Then students, using taught technology and possibly following teacher's example

wrote similar program or system. Often most of the code was written by a sole student and others just looked on the side.

Problem: Those were not software development projects, but should rather be called **programming tasks/assignments**. Yet almost each time there are students who will, in a real software development project draw lines from them and tell how the previous way was the correct way.

Advice: Try to be as open-minded as possible and forget the old "projects". Now you will be **part of a team**, working with real **requirements that change** . Doing constant **improvements and refactoring** to both the **product** and the **development process.**

Traditional project management ghost

(Some of you have worked on Scrum project before, at least you called it Scrum. Some have not even had that experience before). Even until the recent semesters the project management thinking of the students has been **more on the plan-driven than** on the **agile** side.

People have pulled out ideas showing that the traditional plan-driven (~waterfall, Gantt chart, milestones and targets) project management is guiding the thinking, even saying: "We should not have like this, we should have like that" and saying something which is totally against the agile development principles.

Good tests for being agile or not: (Using Scrum as an example)

1. Do you know what you will do in the Sprint after the current Sprint? Do you know what you will do during the following working day?

   The answer to both should be: **no**. Because what you will achieve today will affect the inspect and adapt of the next daily meeting or Sprint review meeting!

2. Do you have an electronic board which shows exactly which features (Product backlog items: user stories, other work, requirements, 10h-60h) are worked on right now, or are not yet worked on, but will in this Sprint? Do you have an electronic board that tells up-to-date information about each task (Spring backlog items: =tasks, 2h-6h) and who's possibly working on it?
   Do you have exact information what is planned, on-going, ready for review, and reviewed DONE.?

   The answer to all of this should be: **yes**.

And e.g. with one Trello board all this would be possible, but more than 50% of the teams fail in keeping Trello up-to-date. We try to show you how to do that.

CHANGE 1

The closest **synonym for Software development** project / Product development project is **Change**. This is because everything will, and also should, change.

One of the certain things that will change will be the customer's requirements. If somebody's mind is locked in the past idea of a "software project" or in the traditional project management thinking, that somebody will be in shear panic. That happens practically every semester…

In agile development the product vision is updated **all the time**. There are plenty of reasons. The most typical ones are:

- changing market makes the old product vision obsolete
- customer realizes what they really want after seeing / running the first demo
- customer realizes that the requirements were not clearly laid out after seeing the demo
- customer realizes that instead of the final product they want students to work more on the sound technical architecture and the customer will create more features later
- customer realizes that the initially asked technical architecture with few demo features could be turned into near final product

All in all, do **not** expect a project where the product vision would not **change**!

CHANGE 2

The other parts that will change during the project are:

- team organization and composition
- tasks, responsibilities, approaches to challenges
- tools
- technology (leads often to rework/refactoring)
- code (refactoring)
- architecture etc.
- the code you just wrote must be changed (rewritten because the architecture was changed, or just because of other refactoring)
- your code will be replaced totally by somebody else's code. (Don't fall in love with your own baby, "Kill your darlings.". The project can have only one architecture where

everything is in line with other components. Your solution might be good in general, but not compatible with this project)

Change means improvement.

Share information, vision and status

One big solution for improving the results would be improved information sharing; between members of the team, and between the team and the customer (who usually acts as Product Owner). Two tools and proper use of them would help:

Trello-kind of Scrum board: Using e.g. Trello board correctly, fully, and up-to-date, developing all the time, daily and in Sprint Retrospectives how to use it will help immensely in these:

- sharing and developing the **product vision** (features, their details, even wireframes, graphics, etc.) between Product Owner and the team(s)
- **transparent project status** and detailed **self-organized teamwork**

Slack-kind-of-channels: In addition to the product vision and status there are many other pieces of information that is crucial for getting **the whole team** in the best level. Slack is one good example of team communication tools. Few years back I planned the following channels for one project, they have been tested now on several semesters:

- **Back-end** (Technical design, implementation of the REST-API back-end. Tools, installation, configuration, back-end programming language questions etc.)
- **Database** (Design and implementation, database related technical questions, database server and SSH pipe configs etc.)
- **Design** (Product vision, features, graphics, usability etc. If customer is not technical expert, this might be the only channel where customer participates actively. If Usability / UX is a core part of the development, it might deserve its own channel. But in many projects, we were fine without)
- **Front-end** (same as Back-end, just for Front-end, web or mobile, plus AJAX)
- **General** (The default channel. General hollering about events or such)
- **Git and version management** (Tool and version management process related questions and info)
- **Scrum** (How to use e.g. Trello, how to improve the Scrum project, etc. work organizing principles)
- (**Team1-Employee, Team2-ProjectManager, Team3-Administrator** etc. team-wise channel, **if** we have a multi-team project doing separate areas)

This has suited several full-stack projects well. You can use the same, or whatever you find to be the best channel mix for your team.

Advice: What we should think before sending messages to the channels:

- Is my message helping the team to progress and uplift each member's spirit. (Tired people make stupid mistakes)
- Am I using the correct Slack channel?
- How should I write my message so that it is clear, short, and understandable?

Sharing knowledge and skills

Every team, especially a school team, has members with different skill sets and skill levels. The team will be successful only if it will be able to share skills and knowledge. Somebody studies a tool or technology and then teaches it to the others. This is **a vital game changer** and failure here is one of the most typical reasons why student teams won't achieve as good results as would have been possible!

Often, we have had SW companies as customers. Students have asked for technology demonstration. After five minutes half of the students have stopped following the presentation. Later they explained that it went over what they could understand but nobody asked the presenter to explain or slow down.

Students often, if asked to give a demonstration, give a show-off about how much they know and how fast they can do it. Possibly because of nervousness and insecurity.

Please ask the presenters questions. Ask them to slow down. Ask them what was the window/folder/file they just opened, what's it's relation to the big picture.

At the moment it seems skills and knowledge mostly come through individual hard work and time spent, and the skill and knowledge sharing fail in most cases!

Hints for knowledge sharing:

- Don't pretend that you understand or know something, ask and try to grasp it
- Ask presenters to slow down
- When you present to others, slow down, read the audience's reaction, put them on the map / big picture, ask the audience to make sure they understand what you are doing
- Ask presenters questions, don't worry about how you look, others are as lost as you
- Be humble even when you know more than somebody else

If you are a technical guru

Share your knowledge and skills to others to get the whole team in speed.

Use the information channels to show others how to proceed. Offer clear, simple and slow-paced demonstrations.

If you are a non-technical person

Even if there are non-technical tasks in projects, even startups seem to have periods when every hands must be on deck. Business side of a product is clear to everybody, but the product has to be implemented.

Try your best from the very first week to get into the code. Understanding the architecture somebody is building. Aim is that at the end of the project all developers are able to (based on the common model) implement new features. You don't have to be the pioneer, but following somebody else's model solution is a lot earlier.

In the beginning of the project - Quality

From the very first week get to the speed with code / technology. Learning, coding, trying out. Making demos. 80-90% of programming is learning and information seeking, especially in the beginning.

But don't rush with filling the project with code. Otherwise you will have a terrible mess in October-November, code where same things are done 5-6 different ways.

Work on the concept and architecture, consolidating and refactoring the code, following e.g. single-responsibility-principle and define-only-once-in-one-place.

Over all quality over quantity! Skills matter, not the code line count. Applies to documentation as well.

At the end of the project – Efficient engine

You should have all team members understanding the architecture, able to add more and more features effortlessly in the clear code structure. Everybody is also able to test their parts in common way.

When the processes are in good shape the momentum takes care of the rest!

Make mistakes

Aim at doing all your mistakes in a school project. Trust me it feels better than doing them in your first real job!

It's better to make a lot of mistakes and learn from them than proceed slowly and try to avoid all errors.

Startup team spirit

You can think that these projects are your startup training. You learn new skills, but most of all, you learn to learn and improve. Code, skills, teamwork, transparency and information sharing.

You cannot change your team members, but that's how it's in most dream jobs too! So think how your behavior gets the best out of others, how the team could achieve more each week.

Stress and pain

Students describe the pains and stress they experience when starting their first job. I hope this school project could be the time **you experience the pains and stress** that push your personal and professional growth.

You don't want to experience all those growth pains in your first job!