OTTO

Project Post Mortem Rep

Trucker boys

DAT255

2014-10-30

Author Information

Name	E-mail	Phone
Daniel Eineving	daniel@eineving.se	+46768455574
Mikael Malmqvist	mikael.malmqvist.web@gmail.com	+46768234669
Simon Nielsen	simon.c.nielsen@gmail.com	+46706990171
Martin Nilsson	martin.lennart.nilsson@gmail.com	+46723040021
Andreas Pegelow	andpegelow@gmail.com	+46768178940
Simon Petersson	peterssonaren@gmail.com	+46767875530

Contents

Introduction 3
Development Techniques 4
Scrum 4
Pair Programming 5
Tools 6
Git 6
Collaboration 8
Future implementations 9
Reflection 10
Appendix: Occlusion Test Results 11

Introduction

This post-mortem report is to be viewed as the final documentation and reflection regarding our software engineering project. The reader will be given a brief overview of how the project was done, what went well and what could've been done better.

Please note that this is not a technical report and will not treat subjects such as how specific parts of the app have been implemented on a deeper technical level. Thus we have limited the scope of this report to evaluate our development technique (e.g. Scrum), our most used development tools (e.g. Git), and how the work within the group has progressed, as well as some suggestions of what features could be added to the app in the future.

We have approximately spent about four hours a day on the project and we are very pleased with our total work effort. Most of the time we spent coding the application, but we did also put a lot of time in the front end design work and to continuously test the application. It is impossible to specify an exact number of hours that was spent on each part, because of different interests every team member was not involved in every part of every feature.

The final product is something that the group is proud of, but there are things that could be done differently and areas which could be improved.

Development Techniques

During software development you will come across a lot of different techniques which you can adapt depending on the situation. Below we will discuss the techniques we've adapted during this project and why.

Scrum

Throughout the project we have been using a variant of scrum for the project management. Because of the limited timeframe of the project we decided to use one week sprints in order to get an appropriate number of iterations before the project ended. In addition to this we have had two weekly meetings instead of daily standup meetings. The short sprints made it easy to adapt to requirements and design changes, but since the project had a very limited timeframe the sprints were hard to define. This was mainly because of how we structured the development procedure during the project. More specifically we ordered the features we implemented based on how important they were to the final product, rather than ordering them by the way they depend on each other. The short timeframe required us to simply having to deal with what was most important first.

One of the key advantages of using scrum was that it made it easy to keep track of the progress and to keep yourself busy working on relevant issues. Since we didn't have any permanent office or workspace we used waffle as an online issue board. One really useful feature in waffle is that you can comment and tag other people on issues, which made it easy for us to keep track of who is working on a specific task and how they are doing.

The two weekly meetings paid for themselves in time spent since they kept everyone on track and followed up on progress each week. They helped with making sure that each member knew what they were supposed to do during the week, as well as making sure everyone got a good overall look on what was happening with the project. The meetings also allowed for everyone to get feedback on the work they've done and getting everyone to be involved in important design decisions.

Our modified version of scrum with an online issue board and two weekly meetings combined was a very successful way of working on this project. The meetings were very efficient and the time spent moderating the issue board by the scrum master was a very minor overhead. In a larger project the weekly meetings are however redundant since you should have daily standup meetings. However we were unable to have daily stand-ups in this project, so we had to adapt.

Since we have had some problems using scrum due to the short lifespan of our project, we think that using scrum would be even harder in projects smaller than this one. However, any project larger than this will need some kind of project structure to function properly and efficiently.

Pair Programming

Pair programming is one of the methods that was utilized in a very natural way during the project. Unlike some companies that tries to always do pair programming, we applied it when it felt like the natural thing to do. The most common scenario is when we found a complicated issue or bug, or when we wrote a complex feature. These are situations where four eyes see more than two, and two brains can analyze problems better together than one brain can do by itself.

But there is one major drawback with pair programming, one keyboard writes considerable less code than two, and in a project like this when we had a very limited amount of time the quality improvement we could gain by doing pair programming did not justify the loss of productivity. Most of the time you can achieve good code quality alone as well, especially since we often discuss different decisions within the group to make sure you have not missed something. In order for pair programming to work the people involved also need to be at the same place at the same time which was not always a possibility for the group. Pair programming is great when you get stuck or need to solve a complex bug, but it does not seem viable to do it all the time, unless you have unlimited time and money.

Tools

Here we will review some of the most relevant development tools we have used during the project. We will motivate why we used these instead of others, and our thoughts on them.

Git

We used Git as our version control system throughout the project, more specifically we used a branching model often referred to as *Git Flow*[1]. Several members of the team had used Git Flow in previous projects, which made it easier for the group to adapt the practice. Git Flow gave us a cleaner Git tree and made it easier for several people to work on overlapping features without disturbing one another.

This idea is great, but sometimes it backfires. Most of the time someone in charge of writing a feature branched out of develop, finished their feature and merged back into develop. Every now and then we realized that a feature that was halfway done, would depend on something else that was halfway done on another feature branch. This was often solved by pulling the other feature into your branch or by merging both branches into develop and then re-branching from that point, which is extremely bad if you want a clean Git tree. The best solution to this problem is better planning, so that you identify dependencies before you realize that you need them. It might be easier if one person on the team was a full time software architect and scrum master. That person could then direct the work of the team to minimize the amount of dependency issues between feature branches.

Even though we've had some minor Git related issues, it has worked really well and *Git Flow* has been a very effective method of structuring the repository. The members who were not familiar with *Git Flow* at the start of the project quickly adapted the practice. The workflow seems very scalable as well, even though some things will change with the size of the team and project, the core concept should work on most projects. In smaller projects than this, *Git Flow* might be redundant but would still work quite well.

One thing we realized towards the end of the project was that sometimes we might have overdone the branching. At some points in the project it would have been

[1] *A successful Git branching model* http://nvie.com/posts/a-successful-git-branching-model/

cleaner if two people worked on the same branch instead of creating their own feature branches. We also had a problem with the bigger features who lasted longer than one sprint. The branch quickly got outdated and in order to not have the other members waiting we merged it to develop with what we currently had that was working. This problem could probably have been avoided by defining the features more clearly in the beginning.

Collaboration

The techniques *scrum* and *Git Flow* have been adapted to fit our purpose and have thus worked very well for us. The group has also worked very well together. Everyone had high ambitions and most of the group worked late nights to make the product come together. We managed to distribute the work successfully between the team members, where everyone seemed to be pleased with their tasks both based on interests and skill level. There has been no actual project leader although we had a scrum master. We used a flat hierarchy instead of having a designated group leader. This combined with the fact that all of us have a genuine interest in software development made way for an efficient way of working. One of the downsides to this technique however, was that it sometimes caused issues where two people had different ideas of how things should work. This sometimes led to their parts of the code being incompatible with each other's. This could have been avoided with better communication and the process of setting up a base structure that everyone adapts their code to.

Miscommunication have been one of the most time consuming parts of this project. Much work had to be redone because members of the group working on different components did not know how the parts were designed and supposed to be connected together in the end. This combined with the iterative workflow makes for some hasty decisions being made.

In a future but similar project, the process of planning the application structure needs to be longer. We believe that this combined with daily stand-up meetings would result in a smoother development process. This is also an iterative process of course and applies to both programming design decisions and GUI decisions. An example of where it went wrong is that the application is supposed to follow the SICS[2] guidelines given by Volvo. One of the tab contents had to be completely redesigned very late in the project because the group had not thought about these guidelines enough.

[2] *Safe Interaction, Connectivity and State*

Future implementations

The application has been developed in a way that allows for a lot of modifications in certain modules such as the regulation handler and vehicle interface. This will result in easy implementations of other vehicle APIs and other sets of regulations such as the US regulations.

We planned for a very ambitious application, but quickly realized that all features that we could come up with would not fit the development timeframe. Therefore we had to restrict ourselves in certain areas. This however means that we have a lot of ideas for features in a later iterations of the application such as: to make the user experience smoother and allow for different users on the same device, all calculations should be moved to a dedicated server. This refactor would also allow for route optimization based on current traffic situations, since more complicated calculations would not be a problem. Here follows a few more examples:

- Tutorial on first startup
- Application color scheme to match daylight and streetlights
- Big Data for fleet managers
- Rest location filtering (user favorites, truck driver recommendations etc.)

Reflection

With the project behind us we can all agree that scrum was an efficient way of working for our group and it has a lot of benefits compared to other development techniques such as waterfall[3]. Even though we have not followed the scrum guidelines by the dot because of the relatively small size of our software engineering project, we feel that the basis of the scrum workflow has fit us very well, mostly because of the weekly follow-up meetings that we had. Therefore we'd definitely say that scrum is a development technique we would use in future projects, however we would devote more time on planning ahead of the development procedure. Making sure that we have a set order of what is to be developed next. This would reduce a lot of issues further into the development procedure.

The distribution of workload has been successfully divided both based on skill level and ambition. Although we have sometimes slowed each other down because of several people working on couplings between modules at the same time. This is something we would like to improve on in future projects by simply preparing our couplings and making sure that everyone is on the same page with how modules should work together, before we start developing our separate modules. This miscommunication and poor planning has led to modules not working together correctly when merged together, which has been a struggle.

The previously mentioned flat hierarchy we applied was also a great approach for a friendly and creative development environment and might be applied again in similar projects, but this is mostly dependent on the group members and the size of the group. We feel that this has worked well because we have been such a small group and because everyone is so ambitious. Our way of working would not have worked as well as it did if some of us were not as motivated to produce a good application that we could be proud of.

[3] *Development technique watterfall, http://c2.com/cgi/wiki?WaterFall*

Appendix: Occlusion Test Results

We tested our app using the new occlusion software. The tested the different use cases that are available when driving in high distraction mode, since that is when the occlusion testing is relevant. The tests were conducted on a bunch of Chalmers students, it would probably be better to test it on real truck drivers, but sadly we didn’t have time for that. The primary use cases that we tested were Change Rest Location and to check the time you are allowed to drive.

Name	Birth	License year	License type	Task	ottt	tsot	uttt
Neda	1995	2015	B	Change Stop Location	3	7866	2
Viktoria	1994	2012	B	Change Stop Location	2	6576	2
Viktoria	1994	2012	B	Change Stop Location	2	7022	2
Michelle	1994	2012	B	Change Stop Location	2	5024	1
Katarina	1994	2012	B	Change Stop Location	2	6896	2
Poya	1992	2015	B	Change Stop Location	2	6707	2
Vidar	1993	2011	B	Change Stop Location	2	6464	2
Oskar	1991	0	B	Change Stop Location	2	5508	1
Oskar	1991	0	B	Change Stop Location	2	5746	1
Markus	1990	2008	B	Change Stop Location	3	7891	2
Michelle	1994	2012	B	Check Time Left	1	3766	1
Markus	1990	2008	B	Check Time Left	2	6577	2
Oskar	1991	0	B	Check Time Left	1	3348	1
Vidar	1993	2011	B	Check Time Left	1	2887	0
Poya	1992	2015	B	Check Time Left	2	5452	1
Poya	1992	2015	B	Check Time Left	2	6351	2
Katarina	1994	2012	B	Check Time Left	2	4960	1
Viktoria	1994	2012	B	Check Time Left	1	4048	1
Viktoria	1994	2012	B	Check Time Left	2	5612	1
Neda	1995	2015	B	Check Time Left	2	4927	1