CHALMERS

Post mortem report - Alive & Clickin’

Chalmers University of Technology

Autumn 2015

Authors: Mats Högberg, Erik Öhrn, Oscar Evertsson, Rikard Hjort and Sebastian Sandberg

Education program: IT

Course code: DAT255

Abstract

This report is a reflection on the development on the Android application Waft. The project acted as an evaluation on how well scrum would function in a project with a tight deadline and with a team that wasn't used to working with agile development. The team used practices such as pull requests, sprints, and scrum boards as well as attending several workshops in order to gain information from the customers. The end result was two-fold. First, a fully working Android application was created, with several implemented functions, including the abilities to send and receive flags based on the current situation on a public transport vehicle. The UI implements Material design and the code of the app has been thoroughly read-through and documented. Second, a result of the project is the knowledge the team has obtained about how a software project can be developed using current software engineering techniques.

Introduction

Background

When it comes to structuring a project there seems to be almost as many methods as projects themselves. One of the most popular ones when it comes to software project is called scrum, where a team works in sprints, which are short bursts of development. At the same time that team must manage challenges not directly related to coding, such as deadline stress, group dynamics, planning and much more.

This is a post mortem report for one of many scrum based projects. It contains the reflections of one team on their work developing an Android application from scratch. It covers the various methods and processes used – both standard scrum processes and those developed by the team during the project – as well as time spent on each of them, their perceived effectiveness and the reflection on whether to use them future projects. The report also focuses heavily on what it was like working with a custom version of scrum, without any prior experience.

Goals

This reports aims to review how well each process and method worked with regards to the project as a whole. It will discuss both advantages and disadvantages of these, and draw conclusions. It furthermore aims to act as a platform for team reflection, stating both learning and problem areas, highlights and takeaways, in order for the team to asses their work and how to act in the future.

Time spent

A workload of 20 hours per week and person was decided upon in the beginning of the project. This proved to be a difficult schedule to keep, especially since all team members were eager to complete their tasks in time for the sprint reviews. We estimate that the total average adds up to 25-30 hours per week and person, with some weeks pushing into the 30’s.

Name	Oscar	Erik	Rikard	Mats	Sebastian	**Total**
Being at lectures	18	10	18	18	14	**78**
Coming up with an idea	20	20	20	20	20	**100**
Knowledge aquisition (not including code specific research)	34	35	35	20	20	**144**
Sprint planning	14	14	14	14	12	**68**
Sprint review	2	2	2	2	2	**10**
Sprint retrospective	6	6	6	6	6	**30**
Stand-up meetings	5	5	5	5	5	**25**
Organizing scrum board	3	2	7	8	2	**22**
Workshops and challenge activities	13	26	15	30	6	**90**
Continous reflection	2	1	7	2	1	**13**
Product owner: Product backlog	3	5	2	0	5	**15**
Scrum master: Summarizing notes and checking in with the team	0	0	8	8	0	**16**
Writing code and design (including research)	90	85	60	65	36	**336**
Writing external documentation	30	25	30	36	30	**151**
Reviewing pull requests	5	3	6	8	4	**26**
Sum	**245**	**239**	**235**	**242**	**163**	**1124**

Figure 1: A rough estimation of the total work put in by each member, both for each process and in total

Processes and practices

Sprints

During the whole project we divided our work into sprints, each sprint lasting Monday through Friday. On Mondays we held planning meetings, and at the end of the sprint we reviewed what had been done during the sprint. The goal was to have a potentially shippable product at the end of each sprint.

Advantages and disadvantages

The clear goals and deadlines of the sprint prevented us from getting stuck and pushed us to move forward rapidly, focusing on the big picture instead of the small details. It made us focus on what needed to be done right now, and helped define a shared vision and goals and for the product. The tight deadlines forced us to focus on delivery, and gave us a feeling of “we’re in this together” and not “I’m working on my feature, he is working on his”. It also made it easier to think of the next step of development since there was a working product to tinker with at the end of each sprint.

A great disadvantage of working with sprints was that it was inflexible. We planned a whole week ahead, and it was hard to change direction half way through the sprint if we realised that we weren’t developing the right thing. In longer projects this might be less of a problem, but to

us, every week was so precious that we couldn't "waste" time on developing unnecessary things.

Another disadvantage was that if all planned tasks were completed before the end of the sprint you couldn't grab something new to work on. There was also the reverse case: if there were tasks left at the end of the sprint, we felt pressured to complete all the work and ended up putting in more hours than we had agreed on.

Efficiency versus time

Focusing on delivering features hurt code quality. We introduced technical debt that we had to sort out during subsequent sprints. Yet, by developing features until we felt that we had to refactor saved us time in the end since it kept us away from premature optimisation.

The clear deadlines at the end of the sprints made us work harder, and made us more focused and aligned.

Using sprints helped us to get more done in less time, and it was one of the most efficient practices we used during the project.

Where to use in future

Sprints can probably be used effectively in all types of projects, personal and professional.

Sprint planning

We had planning meetings in the beginning of each sprint where we split user stories into tasks, estimated them, and chose which user stories to work on during the sprint. We estimated tasks using planning poker.

Advantages and disadvantages

The planning helped us stay focused during sprints, and it gave us clear goals for what to achieve in a sprint. The splitting of user stories into small tasks made time estimation easier, and also made coding easier. Estimating the user stories also sparked many healthy discussions about the system as a whole, and made the user stories more well defined.

These meetings took a huge amount of time, and resulted in a tired and grumpy team. It was also very hard to estimate user stories; we often had work left to do at the end of a sprint.

Before and After

The first weeks we split our user stories into large tasks that in hindsight weren't that well defined. The meetings were fast, but work suffered. We often found ourselves doing the wrong thing, and efficiency suffered.

As the project progressed we refined the planning, allotting it more time and effort, focusing on slicing our tasks more thinly. First, we sliced the user stories vertically, then sliced those slices into tiny, horizontal, bite sized pieces with technical descriptions. This made the meetings twice as long, but it resulted in higher productivity and work satisfaction during the rest of the sprint.

The sprint planning meetings is the practice that we have argued the most about within the group, and the one that we've adapted the most to our needs. Meetings went from being short but unproductive and unrewarding in the beginning, to really boosting our productivity in the end, but more time consuming.

Efficiency versus time

We spent a lot of time planning our sprints. One thing we could have done better was being less technical during planning, leaving the final horizontal slicing to the team member that grabbed a certain task. Despite the longer time spent we think that the planning meetings contributed more and more to our productivity as the project progressed.

Where to use in future

The process of splitting large tasks into smaller tasks is something that we believe can be used anywhere, and not just in technical projects. The same can be said of estimating how much time a task will take, as well as the process of planning your work beforehand, even when you use an agile process.

Stand-ups

We conducted daily stand-up meetings about what we had worked on, what we were working on, and if we needed any help. The way we held meetings changed a lot during the project.

Advantages and disadvantages

The stand-up meetings made us more aligned to a common goal, increased the team spirit, and helped avoid having two team members accidentally working on the same thing. Physically getting to the meetings felt stressful however, and we had problems arriving to

them in time. Even though the purpose of the stand-ups was great, they felt rushed, shallow and overall not very rewarding.

Before and After

The first weeks we met in person for the stand-up meetings, but we later switched to communicating with each other through a chat application instead. This removed the problems of stress and also made it easier to remember what had been said during the meetings. On the other hand, it was hard to remember to write on time every day and we often forgot.

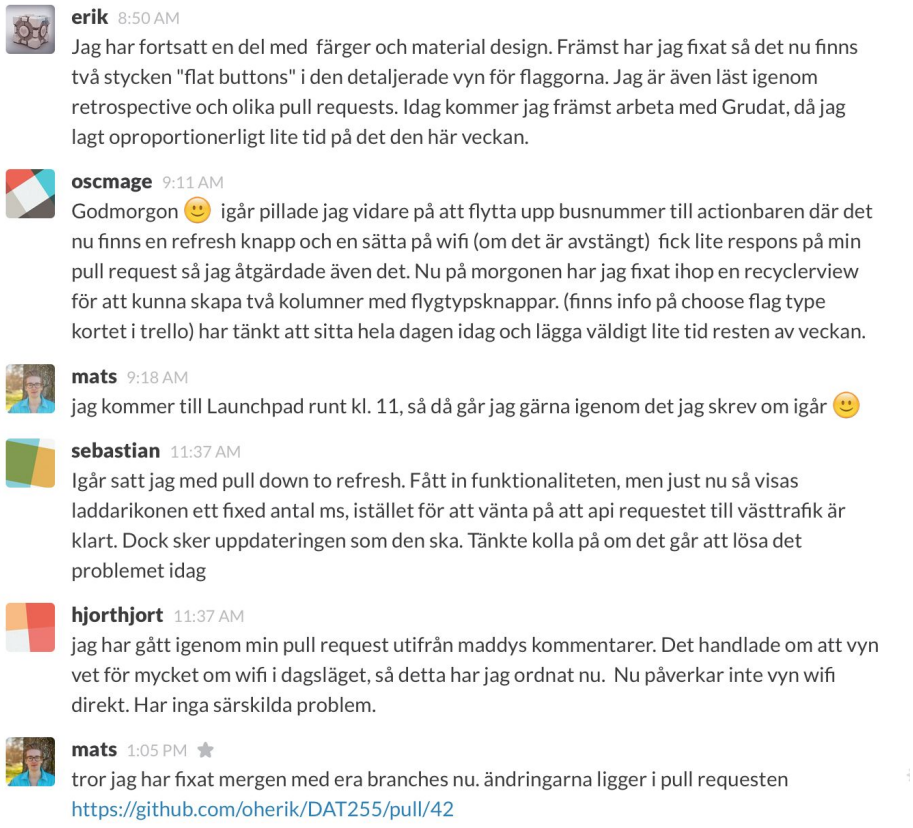

Figure 3: A screenshot of a virtual stand-up meeting in Slack

The switch to digital meetings solved some problems and introduced others, and we aren't sure of what the best way of communicating is. Maybe we should have scrapped the stand-up meetings altogether and just talked to each other when we needed to?

Efficiency versus time

It took a lot of effort to hold meetings, and we feel that they didn't add much to the overall efficiency, mostly since we sat together and communicated a lot, and had a scrum master keeping us in sync.

Where to use in future

We think that stand-up meetings would be a great addition to the workflow if you work in an actual office where everyone is already in place, as well as in distributed projects where it's otherwise difficult to be up-to-date with everyone. We think they would work best when you have set working hours, where the attending people don't have to rush to and from other work in order to make it to the meetings.

Scrum board

We used an online scrum board for keeping track of our product backlog and all our tasks. A task was represented by a "card", and each card was always in a state of to do, doing or done.

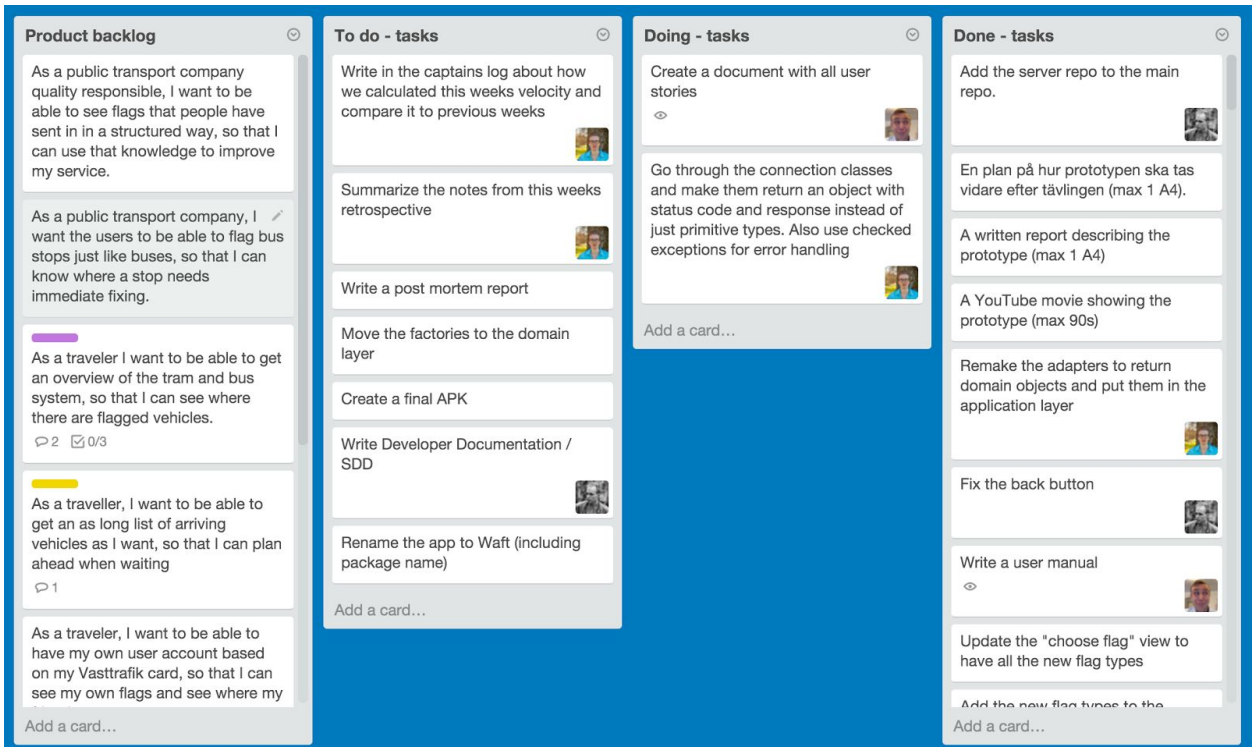

Figure 2: A screenshot of a part of the scrum board

Advantages and disadvantages

The board gave a general overview of the project, what work that was left to do, who was working with what, as well as providing a place to see the history of the project. While it was easy to pick a card and start working, the board was high-maintenance and, especially towards the end of the project, increasingly difficult to navigate. It was furthermore not always in a ready state -- some cards might've been moved to an incorrect place or forgotten about.

Having a physical scrum board instead of a digital one could have helped with keeping things organised, but it would have made it hard to work from home. Having a digital board was probably the best solution for us, but we should have kept it more organised, by making it a priority for the scrum master.

Before and After

In the beginning the board was kept well structured, but at the same time not as powerful since not only crucial and highly regulated content about the project was kept there. In the end it was a lot less structured and more difficult to navigate, but at the same time much more information could be found there.

Efficiency versus time

The board proved to be very useful, minimizing the time discussing what to do at any given time, as well as close to eliminating duplicate work and reducing the time needed for stand-up meetings. Efficiency was reduced somewhat when the board became rather messy and confusing at times, but the advantages outweighed the disadvantages.

Where to use in future

A scrum board would be useful in close to every project and work situation.

Continuous reflection

During the whole project one major part has been continuous reflection, both in the form of retrospectives every Friday, and as keeping a “captain’s log” (a log book the scrum master kept of issues that arose) and writing down personal reflections. Retrospectives was a time for reflection regarding the just finished sprint, while the captain’s log has proven useful in writing this report and as a basis for discussion.

Advantages and disadvantages

Doing a large amount of continuous reflection not only contributed to the project as a whole, but also gave a lot on a personal level. Reflecting on your own performance and contribution to the team was a big learning experience. Writing down conclusions after retrospectives, and more often in the captain's log, provided an accurate real-time experience in contrast to having to remember everything further down the line. It has made this report more honest and accurate. Internal communication also benefited greatly from the reflection. The only major downside was that is was rather time consuming.

Efficiency versus time

In grand total we've spend around 20 hours writing down personal reflections combined with the captain's log, while retrospectives have taken 40 hours. From a coding perspective spending this much time on reflection isn't always the best course of action, especially during this short time period. However in the long run as developers, and on a personal level, we have definitely gained much by spending this amount of time on reflection.

Where to use in future

Continuous reflection is definitely useful in big projects when you're going to work with certain people for a long time. However, if you're working together over a very short time span, tolerating a little bit of friction rather than discussing every issues could be more efficient.

Pull requests

Instead of merging our changes directly to our common Git branch we used pull requests on GitHub that others team members had to .

Advantages and disadvantages

Pull requests worked as a quality assurance through code review. It gave the group a better understanding of the application code, and also helped avoid feelings of code ownership. A lot of bugs were found when reviewing pull requests, so it improved the quality of the code base.

Efficiency versus time

The review process was time consuming (just reading plain code is hard), and we sometimes managed to get ourselves in a deadlock position where we wanted the code from a pending pull request to continue working on another task. Overall though, it was worth the time spent.

Where to use in future

Working with pull request is a good option for any coding project when you're working in a group. The exception would be projects with a very short time frame, like hackathons.

Scrum roles

During this project we had different roles: one scrum master, two product owners and three developers. The roles were held in two-week periods, in order for the people involved to be able to properly grasp their roles and getting feedback from a sprint retrospective before switching. In this project the scrum master mainly focused on his role as a coordinator with a helicopter perspective, while the product owners had double roles as both product owners and developers.

Advantages and disadvantages

Having a scrum master was very advantageous. The developers were able to relax and focus on the coding and design, since the scrum master had a good overview of the project. This also improved the team spirit. The developers felt it was very nice having someone that went around and checked on them from time to time, as well as coordinated the tasks.

We felt it was a good decision to have two product owners, so they could complement each other, especially since they also developed. However, despite the fact that the product owners helped with decision making, we feel that they didn't add much to the project.

Before and After

During the first week the scrum master also did development work. During the following weeks he focused more on the specific scrum master tasks and less on coding, trusting and helping the developers instead. This led to less total coding time, but at the same time made the developer team far more efficient.

Efficiency versus time

We estimate that a total of 60 man-hours were spent doing scrum role specific work that could have been spent on coding or other work. The scrum master role was a very efficient one – while its tasks took quite a bit of time it made for a happier and more efficient team. The product owners were efficient in organizing and prioritizing the backlog, but were at large unsuitable for this project in their current form.

Where to use in future

A scrum master role should be in every project with more than three people. A product owner role would be beneficial when the team has an actual, full time product owner with close access to stakeholders and customers, and not a person working two roles.

What worked well, and what didn't

Knowledge acquisition

Before getting into coding we spent three weeks gathering knowledge about Android. We looked through tutorials, learned about scrum, read up on what the ElectriCity Challenge required and, in addition, what the course demanded.

Reading up on the requirements and Android so rigorously saved time later on, compared to if we just had thrown ourselves into coding for Android directly. We could have done this better though. For starters we all looked at the same tutorial for Android, which probably didn't give us many different perspectives on how to address problems. Also, instead of picking a very in depth guide to Android we could have opted for something more, giving more overview.

Scope creep and "fun-to-haves"

A problem we had in the beginning of the project was that we developed things that weren't needed. For example, we developed a module for handling GPS positioning when we only needed to use the WiFi on the bus to determine the user's location. This was mostly because the specifications we agreed upon during the planning meetings were a bit fuzzy – it simply wasn't clear to everyone in the team what functionality the user stories should include. That, coupled with the fact that we didn't talk to the product owners enough, made us do some stuff that ended up not being used. We largely managed to solve these issues when the planning meetings evolved (see Scrum planning).

Having a dedicated scrum master helped minimizing scope creep and gave us more time for relevant things.

Verification

We had many ideas for what to build, some that we felt more strongly for than others. There were games, travel companions, apps that help people with vision impairment, and other

things. That we finally opted for the idea of flagging vehicles was not because it seemed the most fun, or even the most relevant to us. It was because, when we bounced ideas off of stakeholders, competition managers, and people around us, this was the idea that stuck.

During the whole project, we made a strong point of not going for what we ourselves want, but rather ask the stakeholders in the ElectriCity project what they want. Through workshops we got inside contacts to the two stakeholders most relevant to our app, Västtrafik and Keolis. We kept close contact with these through email (see Appendix A), and got help from them figuring out what they felt was important got reported, and what information they would need for the flags to be useful to them.

From our discussions with stakeholders we gathered that many were impressed with our ideas for making our app into a profitable product. We dabbled with business models, and noticed that stakeholders seemed to get excited of our entrepreneurial ambitions. Since we spent a lot of times at Chalmers Ventures, we got even more positive feedback for our business idea, and we naturally started moving in the direction of gathering free data from users, and selling it to public transport companies. We were confident this gave us a competitive edge in the challenge. Twice we were told at workshops that this really set us apart, and stakeholders seemed generally impressed. During the last workshop, based on some inside tips from a stakeholder what they wanted us to have, we cancelled our sprint and put all our effort into making the required changes. Flagging positive things and storing journey id:s, for example, were entirely based on stakeholder feedback.

In the end, we found that what the stakeholders had been asking for and what our final jury felt was important were two very different things. A solid business model, which we had been convinced was our edge, turned out to not increase our chances of winning by much. This we figured out after talking to the jury after the finals.

To be self critical, we could have directed our attention more towards the stated goals of the competition, taking less into account the feedback we got from specific people. Perhaps talking more to the board of the competition would have helped. But to be completely honest, we feel this was very hard to figure out beforehand – we couldn’t even know exactly what the jury was looking for, even though we sure tried. And today we stand here with an application that we believe has a chance of becoming a business, and we are happy that we set ourselves up for future development this way.

Code quality and testing

We felt that having the features that we wanted and a polished GUI mattered more than having really solid code tests. Our reasoning was that the jury in the competition wouldn't look at the code at all, and that the technical debt that was accumulated wouldn't matter that much because the project time span was relatively short. We reasoned that we would have plenty of time after the competition was over to go through the code thoroughly to ensure it's quality.

One thing that we could have done better would have been to make a clear decision about how we wanted to do with testing. What we did was to decide in the beginning of the project to "write tests where it makes sense, and to test stuff that can be tested". This is pretty vague, and it resulted in tests being written for some classes, but not maintained. It would have been better to decide either that we should have no tests, or that everything should be tested.

Some of the problems that did arise could have been solved earlier if we had implemented test driven development in our project. This would have prevented a lot of code being written only to be deleted during a refactoring process.

Workshops

Our main goal with the workshops was to gain as much information as possible from the stakeholders. In order to do this we decided to attend every workshop, even those that weren't very relevant to our idea. In order to select who should attend these workshops, we first selected those that showed an interest in the subject. We ranked workshops according to our preferences and divided them up accordingly.

At the workshops we made sure to find representatives from the most relevant stakeholders and ask them questions regarding our app, especially questions that related to the workshop theme. Notes and photographs were taken on site and after each workshop a report based on these notes were written in a document and shared with the rest of the team.

If we had a chance to do it again, we would have been more prepared with questions for the workshops, and made sure to have a clear goal of what we wanted to achieve at the workshop. We still think it was worthwhile attending even the workshops that weren't directly related to our app, since it was an opportunity to expand our horizons, thinking of possible extended uses of our app, as well as getting more direct contacts to stakeholders.

We would have preferred to have more personal meetings with the companies with the opportunity to talk to them about our specific product, instead of having workshops that resembled lectures from them.

Non-process specific decisions

Server

Since our users should be able to share flags between devices, we had to set up a backend to store flags. We chose to roll our own server for doing this, and to set up a REST API for communicating with it.

Why our own server instead of an existing service?

There are a lot of free services out there that enables you to communicate directly with a database from the application (for example Firebase and Mongolabs). In the beginning of the project we didn't have any knowledge about these kinds of services, so we simply thought that we had to have our own server to store data. We were also eager to learn more about server setup and server programming, and saw this as a perfect opportunity to learn more about it.

This decision cost us a lot of time early on in the project. Mostly because we didn't have that much prior knowledge about how to set up servers. However, it also made it easier for the app to communicate with our database. We set up a simple REST interface for the server and used it from the application. This enabled us to make changes to our backend without changing anything in our application. If we had used Firebase or Mongolabs directly from the application, we would have been forced to update the application code if we wanted to make changes to the backend.

Having our own server also makes it easy for us to expand our backend in the future. We can easily switch out our entire implementation, including the database, to make the backend scale better. This is not something that we need to worry about right now though, so one could argue that it is premature optimization and that we would have been better of just using an existing service instead. In conclusion: we should have researched our options more thoroughly before deciding on how to do.

Group dynamics

Figure 4: The Alive & Clickin' team. From left to right: Mats, Sebastian, Oscar, Rikard and Erik

Time

One of the biggest problems we had in the group was punctuality. Early on we had issues with people not being on time for stand-ups, which resulted in lengthy discussions about how to solve it. Around halfway through the project we decided to remove physical stand-ups and instead introduce digital stand-up reports. This removed the irritation that happened in the group because of the time issue. The discussions regarding how we view punctuality were very fruitful and gave us an understanding of each other, and of different personality types. This will make us better team players in all future projects.

Physical workspace

During the whole project we had a physical space to work at, at Chalmers Ventures Launchpad. This space made us tighter as a development team. All our longer meetings, such as the sprint planning, retrospectives and reviews were also held there. Meeting every day to work together helped keep the project on track. It was when people were working from other locations that we started having issues. One issue was when two people worked on the same thing, and some hours were wasted before they started talking to each other.

Communication

In order to have an ubiquitous language for everything in our project, we decided that all written documentations should be in English. This helped us in naming classes and functions since we didn't need both Swedish and English names for things. We also decided keep all non-physical communication on Slack (Slack Technologies, 2015). With the help of Slack we could easily categorize our discussions and resources in different channels.

A problem we had was to ask each other for help, as we could have saved a lot of time on asking someone else in the group, rather than keeping our big ego and try to solve problems ourselves.

Truck factor

We tried to keep our truck factor as high as possible during the entire project, but we felt that some of us developed special knowledge about different parts of the application, like the server, the GUI and the general structure of the application code. This was mostly because we often took tasks that we felt confident with, since it would make us more effective and able to develop more features faster.

The timespan of the project was relatively short so this didn't cause any problems, but we think that it could have developed into a serious problem over time. If we would have had more time on our hands, we would have made it a priority within the group to take more tasks that you don't feel that confident about, and take help from experienced team members.

One thing that really helped increase our truck factor was to use pull requests on GitHub instead of merging code directly into our common Git branch, since it meant more people looked at the code and understood it. This ensured that there evolved no "black boxes" of code that just one person could decode.

Conclusion: What would we do differently in a similar project in the future

The use of sprints in the project worked very well. It gave us a clear goal and distinction between each week. This is something we definitely would use in the future.

We would use product owners differently. We noticed that having an internal product owner was almost pointless for the team. We believe that we would have benefitted more from having a dedicated product owner that could talk to the external stakeholders more. This would have helped keeping a clear vision of the project and not being hindered by development problems.

We believe that the way we used the scrum master could have been better. In the beginning of the project, his role was more of a scrum board manager, while in the end he acted more as a person with a helicopter perspective, to keep track of the team's progress and problems. The helicopter perspective made the team move faster and easier in the development process, since issues could be addressed directly towards him and he would help you find solutions. This is more preferable, and is how we will work in the future.

In conjunction to this, we also think that our communication could have been even more open. Our retrospectives and standups helped us vent problems, but by sharing space even more often and asking questions we would have encountered fewer problems.

An important lesson is that, although we share many personality traits and work very well together as a team, we are all individuals. This manifested mostly when it came to opposing values, most notably on the issue of being on time. Being aware of different personality types and actively working to make these work together is key to make future projects roll smoothly.

References

Driessen, Vincent. (2010-01-05) *A successful Git branching model.*
http://nvie.com/posts/a-successful-git-branching-model/

Slack Technologies. (2015) *Slack homepage*
https://slack.com/, fetched 2015-10-28

Trello Inc (2015) *Trello homepage*
https://trello.com/, fetched 2015-10-28

Appendix A: Emails from stakeholders

1. Email from stakeholder

Subject: Innovatin Challenge
From: Lund Per Erik - To: erik.ohrn@outlook.com, mats@hgbrg.se - Date: 27 Sep 2015 20:28, Attachments: image001.png

Jättekul att ni engagerar er i Bussbranschen.

Så här ser våra huvudgrupper ut
(försök även att få tag på hur Västtrafiks indelning ser ut). Fråga efter Jennifer Elsren på Västtrafik hon var med på Arendal.

Nivå indelning

1. Hot/våld/säkerhet/brand
2. Bussfel
3. Bilfel
4. Felanmälan
5. Hållplatsservice
6. Trafik
7. Olycka/ordning
8. Bussbeställning
9. Personal

Som komplement till er ide. Kunde det vara bra att söka svaren på hur resenären vill resa.
Bygga upp en kunskapsbank om hur resenärerna vill utveckla sin/sina busslinjer för att bli ännu bättre. Frågor om önskad resväg (start;slut på hela resan), snabbare/smartare resväg, missade hållplatser, rätt avgångstider, viktiga tider(ex.skolstart), samtrafik mm…

Kör stenhårt, det här kan bli något!!
Lycka Till!

PS! Återkom gärna med kompletterande frågor.

Med vänliga hälsningar

Per Erik Lund
Trafikutveckling

Keolis Sverige AB
Box 47298, 100 74 Stockholm
Besöksadress: Järnringen 1, 433 30 Partille
Telefon: 0725- 60 66 98
e-post: pererik.lund@keolis.se
http://www.keolis.se

2. Email from stakeholder

Subject: VB: Tyck till om trafiken - undersökning från kollektivtrafikmyndigheten Västra Götaland
From: Lund Per Erik - To: erik.ohrn@outlook.com, mats@hgbrg.se - Date: 1 Oct 2015 08:15, Attachments: image003.png Pling_141124.pdf Slutrapport digital dialog Kollektivtrafik Västra Götaland.pdf

För kännedom:
Det finns tydligen en början på att "Tycka till om trafiken"

Från: Blöndal Sölvi
Skickat: den 1 oktober 2015 07:00
Till: Utbult Jessica; Svensson Magnus; Olafsdottir Asdis; Lund Per Erik; Höglund William
Kopia: Andreasson Elisabeth; Blöndal Sölvi
Ämne: Tyck till om trafiken - undersökning från kollektivtrafikmyndigheten Västra Götaland

Hej trafik och marknadsutvecklingsrådet

Förra året genomförde kollektivtrafikmyndigheten Västra Götaland en undersökning om kollektivtrafik.
Resultatet kanske kan vara till nytta för oss?

Ni hittar det på sidan: http://www.hurvillduhadet.nu/
Eller kontakta Tomas Zeljko och Maria Larsson på kollektivtrafikmyndigheten.
Bifogar deras slutrapport

tomas.zeljko@vgregion.se
010-441 21 84

Maria Larsson
Analysansvarig
maria.a.larsson@vgregion.se
010-441 20 32

http://www.vgregion.se/Vastra-Gotalandsregionen/startsida/Kollektivtrafik/Statistik-och-uppfoljning/

/Sölvi

Tyck till om trafiken

Nu testar kollektivtrafikmyndigheten ett nytt grepp. För att få in synpunkter från medborgarna använder de ett webbverktyg.

– Vi vill veta vad människor tycker. Vilka vägar ska vi gå för att nå målen de närmaste tjugo åren, säger Tomas Zeljko, kommunikationsansvarig på myndigheten.

Ett nytt trafikförsörjningsprogram ska tas fram de närmaste två åren, och i en sådan process brukar det ingå att medborgarna får vara med och tycka till.

– Men det brukar vara svårt att nå ut brett, säger Tomas Zeljko.

Ofta blir det en presentation i en samlingslokal, eller liknande.

– Men vår erfarenhet är att småbarnsfamiljer och ungdomar inte kommer till tals. Det brukar också vara betydligt färre kvinnor än män, säger Maria Larsson, analysansvarig på myndigheten.

Därför har kollektivtrafikmyndigheten tagit hjälp av ett webbverktyg den här

Höj rösten via webben. Maria Larsson och Tomas Zeljko på Västra Götalandsregionens kollektivtrafikmyndighet har varit med och tagit fram ett webbverktyg för att få in synpunkter till det nya trafikförsörjningsprogrammet för Västra Götaland. – Det går att börja redan i dag, och alla som vill får vara med, säger Maria Larsson och Tomas Zeljko. FOTO: JEANETTE LARSSON

Chatta med nämnden
Den 18 december mellan 11 och 13 kan du chatta med kollektivtrafiknämnden. Sju ledamöter är uppkopplade och svarar på frågor. Flera bussar, bättre hållplatser, nya spårvagnslinjer … säg vad du tycker. Ledamöterna som deltar är Margaretha Yngvesson (C), Ulrika Frick

3. Email from stakeholder

Hej Erik,

Vi har en herrans massa kategorier här på Västtrafik.

Här kommer kategorierna:

Ärendetyp
nivå 1-
Synpunkt
Fråga
Reklamation

nivå 2-
Beröm
Försenad resa
Klagomål
Önskemål
Övrig fråga
Övriga reklamationer

Kategori
Nivå 1-
Försäljning
Marknad
Trafik
Betalcentralen
Betal
Info
Infra
KIC
Öresundståg

Nivå 2
Annan trafikpersonal
Beställning
Beställningscentralen
Betalmaskin/kortläsare

Biljettkontroll
Byte/Samtrafik
Drift och Trafik
Ersättningstrafik-Omläggning
Facebook
Fordon
Fullsatt fordon
Förare
Försäljningskanaler
Gick inte i utlovad tid
Hemsidan – Internet
Hittegods
Hållplts/utrustning
Information massmedia
Information på fordon
Kundservice
Köp/resevillkor
Körde förbi hållplts
Miljö
Olycka/trafikolycka/skada
Pendelparkeringar
Pris och produkter
Reklamkampanjer
Resecentrum/terminaler
Resegaranti
Reseguiden
Sjuk/serviceresor
Störningsinformation
Tidtabeller
Tillstånd
Trafikutveckling
Övrigt
Övrigt – trafikrelaterat
Ledsagning
Marknadsundersökningar
Områdestrafik
Priser och produkter
Telefoni/talsvar
Twitter

Utgångna kort

Nivå 3

Under nivå 3 finns 56 olika kategorier… nedan följer ett urklippt urval…

Nivå 4

Jag hoppas detta kan hjälpa er.

Med vänlig hälsning
Jennifer Elsren