DAT255
Software Engineering Project
Group Flygande Gurkan

by

Henrik Johansson
johanshe@student.chalmers.se

Magnus Källten
magkal@student.chalmers.se

David Strömner
stromner@student.chalmers.se

Benjamin Wijk
wijkb@student.chalmers.se

2014-10-25

Johansson Henrik
Källtén Magnus
Strömner David
Wijk Benjamir

Index:

1 The Application

The HazmatEasierManagement application is an application made for truck drivers that transport hazardous materials. The way you use it is that you add all the materials that you are planning to transport to the application and then press a button that generates a list of what you need to do in order to legally be able to transport it. It will for example tell you if a material can not be transported with another certain material depending on what group classification it has. Or if you want to transport more (weight) than is legally allowed in a single trip.

2 Processes and Practices

2.1 Scrum

We opted to use agile development, specifically Scrum, instead of the more standard waterfall method because it allowed us to be more flexible with the requirements around the program. Specifically it made the whole process a lot more dynamic and decisions we made early on could be reworked or thrown out without putting the whole development process on tilt. We chose Scrum' because it is a flexible 'way of work', and works well for smaller projects. Reason being that you can have one person that can keep track on where the project is going and delegate what needs to be done for each sprint. Also each person inside the team knows approximately what everyone else is working on so slip up in communication doesn't cascade into enormous problems. The meetings themselves was also quite short but regular which had the benefit of catching up what everyone else was working on, aiding in the communication between everyone and getting help if someone got stuck. The amount of time that was put aside to conduct these meetings was well worth the time since errors and bugs occurred less frequently and was fixed at a faster pace.

Although the Scrum is great for smaller groups and projects, but the method doesn't scale well to larger projects since it's impossible for a single person to keep track on a project if more than 100 people are working on it. It is also unreasonable to expect all the other people inside the project to keep track on each other. The daily meetings would take quite a long time to conduct if everyone has to say what they did yesterday and if they needed help, so meetings that should take 10-15 minutes would easily span hours.

2.2 Waffle

We used Waffle to keep track of the backlog and the sprints because it was easy to connect to the GitHub repository and made it easier to see what each member was expected to do each sprint. A downside was that normal activities on the repository, such as commits and pulls, was shown in the finished column in waffle, when we just wanted to see the finished sprints,user stories, etc.

Waffle is easy to use and organize. It doesn't take much time compared to how much it helps and we would most likely use it in future projects if we had approximately the same amount of people in the group, but not with a large group or if it is a bigger project due to the fact that it was difficult to see if there were too many stories in the backlog .

2.3 Git

To synchronize and work simultaneously on the code we used Git because it is much easier to collaborate with instead of manually sending code to each other which usually ends up causing a lot of problems when multiple versions are floating around and no one knows which version they should work against. Or try to synchronize with Dropbox which unfortunately caused a lot of sync problems to occur, and doesn’t give you an automatic version history. The downside with using Git was that not all members were confident in using it so we had to create a Cheat Sheet on how to use some of the more advanced features. We also had some problems with pushing to wrong branches and merging them together. Even though it took some time to learn Git it would not have been possible to complete the project without it and we will most likely use it with future projects unless some easier cooperative programs are created.

2.4 GitHub

We used different repositories on GitHub to enable working individually with the code and then easily being able to merge it into one. We had one person that was in charge of the main repository, consisting of two main branches, Master and Development, and many smaller branches that was used to implement and test smaller parts of the project. After something was done and documented on a small branch it was pushed up to the Development branch. The purpose for this constant merging was so that the program had less problems with merge conflicts. At the end of each sprint we finalized a version and pushed it up to the master so all pushes made on the master branch contained finished, documented and tested code.

2.5 TDD (Test Driven Development)

We wrote a couple of initial tests that defined some basic functions that we wanted the classes to pass, and then wrote the method bodies so they could pass the basic functionality tests. We then revisited the tests and wrote more advanced ones to assure that no special case could break the functionality that we strived for in the first tests. This made it so pinpointing possible problems when merging two different

parts together became easier. The tests are a great way to prove that your part of the code work as intended, it also helps others to see how they should use your methods if the documentation is lackluster. You earn a lot in time in writing tests since you do not have to manually re-test parts of your code if changes are made, so spending an hour writing up proper tests can save you multiple hours and headaches in the future.
One problem with tests is that it does not prove that something actually works, but only if something does not work. If a test works but the code is not correct it can lead to problems when debugging. This is useful when working on medium to large projects but might use up precious time with smaller projects with less people involved.

3 Reflection

3.1 Ups & Downs

While not everything went as planned, many parts of the process and techniques that we used during the development worked quite well. Even if it was our first time working properly with an agile process, we have used the ideas behind the process before. None of us have ever succeeded in doing a waterfall product since it is intempestive to design an entire software product in just one iteration. So we are used to doing small iterative additions to our projects and as such Scrum was a great way to handle the process. At the same time it puts some pressure on getting parts of the product done and not leaving it aside for later.

None of us are experts with working with Git but after using it for an earlier course last year the amount of problems that comes up when using Git for the first time were significantly decreased. Git helped immensely with keeping track of versions and integrating parts of the project into each other.

We had done testing in previous projects, specifically JUnit, so we had some experience with how tests should be written and the benefits of them. That is why it was easy to write tests to prove that our code worked as intended.

Of course we ran into some problems, most of them related to how pressured we were for time. The main problem was that we did not start focusing and working on the project until halfway through the course which made everything else lag behind. In the beginning when we started working, the log was severely neglected, this made it difficult to remember what each member did on what day and how long it took. Even with detailed User Stories and Sprints this could have gone smoother if their final dates were fixed, and did not span over deadlines. Because of time constraints we never got around to implement what we had planned with the AGA jar files. If we had been more comfortable with the Android platform we might have had more time to try and implement it.

3.2 Non-Process Specific Decisions

A lot of the originally planned functionality for the app was removed over time. It seems to have been a combination of time constraints and over-ambition when starting out. We are used to larger projects and it is natural that some estimations were faulty, especially when learning new software at the same time.

The idea of the application we ended up going with came from an interview with a semi-truck driver and we had a couple of suggestions for an application that we thought would benefit him in his daily life. Most of the ideas we brought up he already had or knew a solution for. He then suggested an application that would help him and his co-workers with transporting hazardous materials, since there were so many rules to keep track of. The following paragraph is an excerpt from the interview we conducted:

"... If it helps at all, myself and a lot of other drivers would really appreciate a app that helps with hazmat. There are so many rules regarding hazmat that it is difficult to keep track of. There are certain materials that can't be shipped together, and if one material weighs a certain amount, we have to put a special placard on the truck. I don't know if you guys could make anything like that, but I know that there is nothing like that out there today."

Which sounded like a great application for this project, and that we all agreed that is was better than the ideas we had thought of.

3.3 Reflection Group

An issue we had with our group was that out fifth and final member never showed up, so on instructions from the tutor we pushed forward as four, this obviously meant that we could not do as big of a project since we were only four members. However a smaller group meant communication and assigning people to different areas of the project was made easier, as well as reducing the amount of conflicts with Git when pushing and merging.

Unfortunately we had a couple of hiccups when it came to meeting the deadlines for our sprints. The sprints were short, and we had some issues meeting up on a daily basis because our schedules were quite different. So if a problem occurred that someone needed help with it took one to two days before we could sit down and discuss a solution to the problem. This ultimately affected later sprints since material from earlier sprints had to be done before we could move forward with new functions. In the end the sprints were more of an advantage than a disadvantage since it forced people to spend extra time, if needed, so they would not delay the deadlines.

Some things that went well in the group were the discussions we had with each other as well as the app idea. Brainstorming what application to make could have taken a lot longer because figuring out what kind of application a truck driver would want, that also incorporates their and others security was a difficult task. The reason for that being that no group member had any extensive hands on experience on driving a truck.

Some things that could have went better was the tutoring sessions, which was mostly the groups problem. We had the idea that the tutor was supposed to guide us and help us to set up sprints and deadlines. In reality the session was mostly us updating him on how far we had made it, and if we had any questions he would answer them if he could. This in turn made the first meetings extremely short and did not give much interesting results to incorporate into the project.

Another thing that we thought was a problem was the layout of the course, the fact that we didn't start programming and really discussing the layout of the application until almost halfway through the course. It would have been better if we in the first week had to do a small application and make some minor pull request to a remote repository with Git. This way groups without much experience in these areas could have had a faster startup. Could even be optional so it wouldn't be tedious for more experienced groups.

4 Things That We Would Change

If we did a future project with approximately the same settings we would most definitely try to get all of the administrative task done within the first week, compared to the three weeks it took us to start coding. Missing out on two extra weeks meant that we had to cut a lot of features we agreed were critical for our application, for example some sort of history, so you could save a template of your cargo, and use it again in future deliveries. The time we had left was not enough for the full idea we had in our vision.

We had some minor problems with assigning too many and too unclear user stories for the first sprints, which caused us to lag behind and extend some of the stories over two sprints. So if we could do it again we would write up more specific user stories so it would be easier to judge how much workload we had week to week.

5 Time

Scrum

Total Johansson	6h
Total Källten	4h
Total Strömner	7h
Total Wijk	5h
Total	22h

Database

Total Johansson	28h 30min
Total Strömner	30h 15min
Total	58h 45min

Other

Total Johansson	16h
Total Källten	9h
Total Strömner	11h
Total Wijk	3h
Total	39h

GUI

Total Källten	55h 30min
Total Strömner	5h
Total Wijk	40h 30min
Total	96h

All total

Person	Time
Johansson	52h 30min
Källten	71h 30min
Strömner	63h
Wijk	50h 30min
Total	237h 30min