# Post Mortem Report

GROUP 4

Process management application for Lidköpings Stenhuggeri

Chalmers University of Technology

DAT255 - Software engineering project course

2013-10-26


**Project members**

Alexander Härenstam, Simon Bengtsson, Robin Grönberg, Olliver Mattsson, Anton Jansson, Kim Kling

# Summary of contents

# Preliminary

Everything in this report is written in regards to a software development project with the ambition of simplifying the process management at the stone masonry Lidköpings Stenhuggeri.

Lidköpings Stenhuggeri is a company that produces and delivers graveyard stones to different cemeteries in Sweden. Today all orders and processes are managed through papers being handed from station to station which implies that a lot of time is lost looking for papers or asking around for information.

Our solution is to digitalize their stone creation process and therefore increase the efficiency.

As project members we've had limited previous experience of the techniques involved. Both in terms of project management techniques and development techniques.

This report is a reflection over the processes and practices used, time spent, how well the tasks were managed as a group and at last rounding it all up with conclusive texts how it all went.

# Processes and practices

## Agile

There are several ways of developing incrementally with agile development and during the project several agile development methods were used. In the following texts we dig into the branches of agile development which were used in this project in particular.

### Extreme programming

Some parts of Extreme Programming (XP) were tested during the development process. Pair programming was one of them and proved to be very powerful, even though it was time consuming.

Test Driven Development was as well an essential part since it allowed us to make sure that the created class fulfilled our requirements for it. Including unit testing, another XP practice, which was used on a large scale. This was superior for finding bugs in our components because in many cases after merging two code branches, issues with some component occurred. Fortunately the tests could immediately show where there was a bug which made testing really useful.

### SCRUM

For documentation we started off by creating a product vision with the overall goal for the project followed by a product backlog based on user stories which was one of the more major parts of the documentation. Each week we then started a new sprint planning and selected what should be done, ending up in a sprint backlog, which consisted of parts from the backlog.

In the start of every following new week we had a sprint review where we discussed what went well and what didn't go well during the last sprint. Even though the reviews often were short, they provided us with an understanding of what could be improved.

To update everyone on the work each team member had been doing we tried having daily stand-up meetings. However, working as closely together resulted in us often forgetting about them and any issues that occurred were dealt with straight away instead. Mostly in the beginning though, before we got used to this new method.

## Time spent

According to our own schedule we have worked approximately 120 hours per week which is 900 hours in total based on 7,5 weeks. This means that each group member has approximately worked 20 hours per week which is 5 hours less than what is expected from us, which is 25 hours per week. But then, this is still just an approximation. Some weeks there may have been more, and some there might have been less.

Throughout the entire project we booked specific working periods where we planned to work on the project, as a group. We scheduled approximately one or two working periods a day which were mostly 2 hours long but could be up to 6 hours or longer as well. These sessions were fairly efficient and the entire group worked well during these over the entire project time. Certain sessions were obviously more effective than others though but as a whole they were quite successful and almost all features were completed during these periods.

Every friday we worked approximately 8 hours to get the release ready. These days everyone was at the worksite and worked efficiently. This was key in getting a release done every friday and we often sat past the time we anticipated to be done. This led to that we often had to sit 2 hours longer than the general estimate of 6 hours each friday. Because of this safety margin we always had a release ready which was runnable.

We always put in our sprint backlog how long we thought each task would take but we often misjudged it and put in too little time for each task. This resulted in that the actual time spent was often 50-75% longer than the estimated time and that the sprint couldn't be finished in time.

The reason why it often took longer than estimated was because we ran into bugs and problems that wasn't anticipated.

Our sprint backlog says that we each week put in about 60-80 hours per week as a total but this is quite misleading. We often missed putting up tasks that had to be performed in order for the task we were working on to work and didn't put that time in the task at hand. This made several hours disappear from the sprint backlog which should have been put there as separate tasks needed in order to complete the task that was being done. A good example of this is when a need for refactoring arises or if an unexpected bug appears.

# Techniques and Practices

For this project we used practices such as standup meeting, pair programming and Test Driven Development (TDD) among others, as mentioned earlier. The techniques worked with varying efficiency as some were very good, and some were not. In the following texts we will discuss them in more detail.

### Pair Programming

Pair programming proved to be very efficient in most cases even though one person were not practically able to write any code because the benefits still outweighed that.

When using pair programming bugs are easily found and eliminated. It also makes it a lot easier when something tricky comes up because it's solvable by simply discussing it through with the other person. The downside is of course that as stated earlier, one person is hindered from programming since two people are working together. Luckily in the long term, you can save a lot of time because it allows you to discover potential bugs and errors much more easily. This means that even if the two people would work separately the same amount of time, it would in the end require more time than if the pair worked together.

This technique would be useful in projects where you are more than three people since if you only are just a few, you already work close enough and can stop what you are doing and help each other without any hassle. The more people you are though, the better pair programming is because then you can spare resources by having less people code at the same time and making sure that everyone has someone to discuss any uprising problems with. Without having to disturb anyone else's work flow.

### User stories

User stories have proven to be quite useful since they are quick to write down and it's easy to pick out tasks out of them. However they are difficult to administer and keep up to date since they are dynamic and changes as the project develops.

To write all the user stories at once for the backlog is time consuming but pays off in the sense that it is easy to know what to do once they have been written. Stories works well in all sorts of projects which has a clear user. If the project has no clear user it will take up too much time to come up with different stories. Therefore it's always essential to have a target group for your project when using this technique.

### Standup Meetings

Standup meetings works really well most of the time because you get up to speed on what the rest of the group have been doing and they discover what you have been doing. There is however one downside if the case is that someone has an issue that requires immediate attention. The result often tends to be that parts of the group drifts away on solving the problem instead of completing the standup meeting, hindering the ones not necessarily needed for solving the problem from going back to work.

The main advantage of the meetings is that they are often very short which means that you get the information-sharing short and compact which simply leads to a greater understanding of the project as a whole in the short amount of time spent.

The time spent on the standup meetings is therefore well spent but this type of meetings most likely works best in a medium sized group though. If you are in a smaller group with only three persons or less the standup meetings would only get in the way because you would probably know what the others were doing anyway. However if you are more than approximately ten people, the standup meetings would instead take too long and be inefficient.

### Test-Driven Development

TDD proves to be very useful in finding bugs but it requires more time spent on writing code. The good thing about TDD is though that you can assure yourself of that the code you wrote actually works. In some cases time is well spent and in some, it really isn't due to the time-consumption of writing tests that isn't necessarily worth the time. Simply since the bugs could in some cases be ridiculously easy to find. On top of that you can still not be sure if the code works until you actually test it in practice.

If you are working on a very complex class it is extra essential to write a test since finding bugs in complex classes is extremely time consuming. Although in less complex classes or classes that delegates most of its functionality to other classes, it's obviously not worth the time. The reason why is since bugs are most of the time super easy to find and you can probably instantly see them by just looking at the code in these cases.

**Backlog**

We set up a product backlog early in the project which of course took some time but paid off later in the project since it was easy to know what was prioritized and what to do next once you had completed your current task.

Once the backlog was set up it was little to no work keeping it up to date and this allowed everyone to be up to speed on what was most important which saved a lot of time in the sense that there was always something left to do.

This strategy is useful in most projects no matter the size since it just helps keep track of the priority of different tasks.

**Sprint Review**

We also used sprint reviews to see what went well and what didn't during a sprint. These helped us to avoid repeating the same mistakes. The result was that the working process became more efficient throughout the project since more and more things that hindered our progress was discarded. The time consumed by these payed off in the sense that we became more effective after each of the reviews.

Reviewing is a good technique to use in all projects since it allows you to notice both the good and the bad parts resulting in a better work-process as you go along. This helps everyone in the group to evolve and embrace what they did good, boosting their self-confidence while also understanding what went wrong and how to avoid it. Which really is what learning is all about when you come to think of it and allows you to progress as a group.

# Project

Our ambition was to use Scrum during this project but Scrum or agile development can be hard to get used to if no one has worked with it before. However, our group has managed to adapt our style of developing with agile development.

Something that went well are our sprints because it was decided that our sprints should stretch over a week, which all of them did. However during the first sprints, we planned to do more during one sprint than we had time for. Due to that, much of the work went on to the next sprint. It worked better after a couple of sprints though when we became more familiar with the agile development process.

Our sprint planning and sprint reviews were quite short and not one hour as they should be according to one of the lecturers in agile development by the name of Thomas Luvö. We thought though that since our sprints are only one week and not one month as they usually are in the sprints at Ericsson which was his example, we didn't need that much time to plan our work. Another big issue which is closely related to this was that our sprints had a too large scope which is something that showed up early in our sprint reviews.

At the end of each sprint, we released a version of the application as scheduled which is something that worked very well in our project as almost all of our new features which we worked on during the sprint were included in the release and functioned as promised.

In the first iterations, we had a slight communication issue as parts of the group worked kind of isolated. This lead to that features were implemented which hadn't been discussed as a whole group how to do, yet. The code was mostly rewritten later and therefore lead to more work for the entire group because of a simple misunderstanding. The same problem also showed up in the sprint reviews quite early but fortunately we managed to keep it from reoccurring in the following sprints. Our great regular daily meetings was one factor that solved this issue.

The scope of our project were bigger than we could manage within the time limit. However our biggest and most important features works and the main purpose of the application has been fulfilled.

## Reflection on non productive decisions

The choice of working group-wise instead of splitting the responsibility between group members, where people would be on their own, appears to be an excellent decision when looking back. It allowed us to control the growth of the application really well while saving us a lot of time-consuming communication to update each other on how the progress went. On the other hand it did as well allow us to drift away a bit during our sessions when we would lose our focus. Resulting in time wasted on non-progressive activities but that is a part of the development as well. Everyone need breaks.

Having lunch-breaks allows you to take some time off and refresh your mind which should be a good thing. It differed from day to day though how long it would be. I suppose that we got back to work a bit late some of the times though which put us slightly behind schedule but we probably made up for the lost time the days we instead decided to start early.

Working same days every week is comfortable and turns it into a routine which probably is good as long as you do not get too comfortable, which often results in getting lazy. Therefore some slight changes is probably to prefer to not let your mind get stuck in routine.

Working at about the same locations every week made it easier for us to handle our assignments in a way that fit the environment. For example, it was quite essential to have a whiteboard close in order to be able to easily discuss difficulties as a group.

During the scheduled working periods, barely any breaks were taken which was extremely tiresome in the long run and resulted in the group being less effective and irritated. Whether or not this was a good decision is hard to judge since with breaks there would be less time spent on working, even though more progress would have been made during the time we actually worked. I suppose it is a tie but having breaks is most likely preferable in order to keep the spirit up.

A communication media which suits everyone in the group is certainly helpful when you need to reach someone, or several people of the group at once. At least by judging from this experience. Facebook Messenger was the mostly used media in this case which opened up for easy one-to-all messaging which worked excellently.

## Group Collaboration

Most of the work were performed during day-time at scheduled times. Using a shared Google calendar in order to keep everyone up to date with changes and making sure that everyone was free which worked really well overall. There wasn't a single moment where someone wasn't sure of whether there was something planned, or not. Everyone have been attending even though people tend to always be late.

The second week we made sure to book rooms for the rest of the development period in order to always have somewhere to go which has been working most of the time. It's a fierce battle otherwise.

A chat on Facebook helped us communicating, especially regarding when someone couldn't attend to a meeting or a scheduled group activity. Our every-day communication helped a lot too since we meet every day outside of the project as well.

Since we used pair programming we had to cooperate well with one-another and work really close. One side-effect from this was when one of us would continue the development on their own, at home for example which would put the other one behind. The next time we would work we would solve it by discussing what had been done and updating the other person on the new feature and how it was implemented. Unfortunately it can sometimes be quite difficult to explain an advanced implementation by words only. Especially if decisions were made regarding how to structure the code.

Communication overall has had its ups and downs, as mentioned earlier we ran into trouble the second week because of miscommunication where the word "task", in the context of our application structure, had different meaning to different people which lead to the application not functioning properly. In the long-run though we've been communicating just fine, despite a steep learning curve in the beginning.

# Conclusion

We've learned much as a developing team. In the beginning we commonly ran into trouble at the end of a sprint but fortunately we talked about it during the sprint review and made sure that it would not occur again.

A better and more structured way of documenting as well as less documentation would come in handy. It comes with experience though. We needed more time to develop a way of writing documentation that would feel natural to everyone and easy to understand.

Next time, we should work closer to the backlog and sprint log as well. Sometimes felt like we drifted away from it, implementing things that wasn't as essential. Can be compared to a company's way of looking at it where it will be an unnecessary cost to implement things that aren't requested by the buyer. Instead you should focus on only writing code that is of worth to the buyer and does specifically what have been requested in order to generate revenue.

You could look at the group as a whole and make sure that the skills among the group members are very different. This way everyone could contribute well with their certain expertise, no matter what type of project. For example you could have one with focus on testing while someone else is design-oriented and one handles the code-structure. I'm sure you get the point.

It wouldn't hurt to work in a more relaxed environment either where you had access to fresh air to keep your mind energized, easy access to refreshments and some close-by activities that you could attend to let your mind rest during lunch. Writing this I was thinking about the movie The Internship where they are showcasing how they do it at the headquarters of Google which is one of the top technology companies in the world for a reason.

Whiteboards are awesome too, imagine filling a room with them where you can keep your notes. Makes documentation a lot easier because you can write your ideas on the wall just as they pop up in your head. As of now, you have to take a picture of what you wrote and erase it since you move around from room to room. The solution would probably be to use online scrum tools to move the whiteboard to the cloud. That would in our case have centralized our Scrum management and made it more accessible. Resulting in increased productivity and minimized time on paperwork.