Post-mortem Report
Krogkollen

By: Per Thoresson, Filip Carlén, Johan Backman, Oskar Kärrman,
Linnéa Otterlind, Albin Garpetun, Jonathan Nilsfors

Supervisor: Andreas Berggren

Chalmers tekniska högskola
Informationsteknik
DAT255

Introduction

This report contains a reflection of the development process
of Krogkollen. We will go through the different practices and techniques we have used and how we found them useful.

Krogkollen is an application developed by seven students from the Software Engineering program at Chalmers School of Technology. During our second year we've been reading a course called "Software Engineering", which teaches how to run a software project in an effective way.

Our ambition was to create an application that will help people to find a place where they can grab a glass of beer and hang out among friends. To help people with this, we wanted to show a map of available places, and give information such as price, age restriction and queue time. The focus on queue time was something that would be unique for our application.x

Analysis of time spent

Approximately every member spent 140 hours on this project. This translates to 20 hours each week. The last week was spent fixing bugs and polishing the application. In total this project took 980 hours divided between seven members.

The admin application, which took almost all of one developers time clocked out at 85 hours. This meant developing the application from scratch. Design decisions and testing also took time from other developers in the project.

In the main application there was several steps which required their allotted time as well. The detailed view was the first real challenge with android UI-development we met, thus taking 115 hours to complete. The last added feature, the list view, took 100 hours splitted between two developers. The detailed view presented some interesting issues with information encapsulation that took some time to solve. In order for the application to work as intended we also needed a server connection. This took 95 hours, including creating an interface for the android application to work with.

The thing that took the most time however was the map view of the main application. This is natural of course, since this is the main feature of the application. Most of the developers has had something to do with this view, and in total 205 hours was spent on this part of the application.

380 hours was spent on miscellaneous administrative tasks. Some of this time was spent programming all other aspects of our application than the ones mentioned above. Mostly though, it was spent on documentation. We discussed solutions and we had meetings (this includes the weekly Scrum). Other then that we also needed time to learn android programming, write tests and such. Of course preparing the weekly releases took its fair share of time as well.

Processes and practices

SCRUM

In the development of 'Krogkollen' we used the agile project model Scrum. Scrum was a new concept to all of the project members when the project started, but now it is something that we are all accustomed to now. Sometimes it felt a little bit forced, but we followed many of the methods of Scrum anyway and customized them so that they would work well with our own development process.

At the start of the project, we had a long and intense start up meeting where we discussed what features the application would contain and briefly planned the whole development phase. Questions like 'what should be done in which week', 'which priority does the different features have' and 'how many hours do we estimate that each step will take', were discussed and drawn up on a white board. This would ultimately be our project backlog.

We continued to return to this backlog during the whole process but obviously we had to change things when we for example realized that something would take a lot more time than originally estimated. Though the backlog changed over time it filled an important purpose, at any given time it clearly defined the goal of the project.

We wanted to have a weekly scrum meeting and we quickly decided that Mondays at 15.00 would be the time that suited us best. This meeting came to be a mix of a scrum review and planning the next sprint. We discussed the events of the previous week, what went well and what didn't, and also planned who would do what in the next sprint. We discussed problems that we had found and if we needed to change something because of it. If anyone had thought of a new feature or a change to an existing one during the previous week, we discussed if this was something that could take the project forward. This weekly meeting gave all group members a chance to catch up with with everyone else's work from the previous week and proved to be a great way to work.

The daily scrum meeting, that is frequently used in scrum is something that we chose not to use. This due to our regular programming sessions in which all members usually participated. Daily meetings proved unnecessary since we almost worked together all of the time. This gave us all a good overview over the whole process, and transparency was never a problem.

One of the big advantages of using Scrum according to us is that you will have a running product through your entire project. This is something you won't necessarily have otherwise. We can't really find a disadvantage (given that you modify it to your needs) of using Scrum so the next time we start working on a project of this type, we will use it again!

Pair programming

During this project we have used something that is called pair programming. This is just what it sounds like, you program together with a partner. Since our group had seven members it was not possible to use pair programming at all times. So what we did was that we combined it with

the scrum backlog and compared which of the features would be easiest doing by yourself and which would really need two people working on it. This was something that worked very well and if the one who was alone had any problems a person from one of the pairs could help him for a while instead.

During the project we found out that pair programming is a very comfortable and easy way to work. Problems and bugs are fixed much faster since there are two people focused on solving them instead of just one. You have the opportunity to quickly exchange ideas and you might also learn something from your partner along the way. It might seem unnecessary to use this method when we have been together as much as we have, because you could just discuss with the other people around the table. But the difference is that in pair programming the person you are working with is way more familiar with what you are doing and you don't have to explain the code to someone else, which saves a lot of time.

Overall we think that pair programming saves a lot of time since you have two people looking at the code, which for example makes it less likely for bugs to sneak into the program and you are optimizing the code directly instead of someone having a look at the code after you are done. To summarize, we would definitely use this method again in a similar project.

Technical details

Technical decisions

In the beginning of the project we knew that we would have a big server side of the application and some of the group members had heard that Grails.com would generate a homepage automatically, so we decided to try and use that. When we tried to implement this it turned out to be much more complicated than anticipated. Because of this, we decided to not use Grails.com and instead use Parse.com. Parse provides the user with a database. The database can store just the values that were needed in this project and the connection between the client and the server was easy to establish. It only required one line of code and helped us focus on the client instead of spending a lot of time working on the server side. This left us without a generated homepage though, and we thus had no way to handle new information about the pubs.

When deciding how to handle the information on the pubs, we chose not to create a website. Initially we thought that it wouldn't be a problem to create a simple website but what we didn't know was that the external library from Parse.com did not have direct support for PHP. Nobody in the project had any experience with Javascript, so it was decided that the time needed to both learn Javascript and develop another connection with the database was too much. We instead chose something that took very little time to develop, but would yield similar results. A web form made on wufoo.com. This meant we saved a lot of time not developing the website, but instead we have to put the information into the server manually.

Technical problems

At the very beginning of the project we decided to use Android Studio. It was supposed to be the preferred IDE for developing Android applications but it was still in a test phase and we had a really difficult time trying to get certain features of it to work. When we met with our supervisor and discussed this we came to the conclusion that we shouldn't use Android Studio, instead we should use IDE:s we were already accustomed to such as Eclipse and IntelliJ IDEA.

In our first sprint the most prioritized goal was to be able to see a Google Map in the application. This was supposed to be easy but took several days to get to work. It was a pretty serious problem but we never really thought of finding a replacement or changing our application since it was such a necessary feature, instead we just worked until the problem was solved. This was solved by downgrading our Google Maps API. The latest version had problems on Android. This is a great example of a problem which is really hard to foresee, and required extensive testing before we finally figured out what was wrong.

Over all, when we had any problems we made an assessment to see if it was worth the work to fix it or if we needed to rethink our choices. Since we used an agile working method it was easy for us to make changes if we needed to. Most of the times it resulted in us simply fixing the problem in a reasonable amount of time. Some things, like the last feature of being able to ask for a song at a pub through the application, was dropped. This was no big deal however, since we even stated it to be an extra feature in the backlog.

Evaluation

What went well

The weekly scrum meeting was definitely something we all found to be a good habit. It gave all of us a good overview of what we had already done and what we needed to do. Everybody could, and did, contribute to our process and we knew what our responsibilities were every week.

The team work in this project was almost flawless. All group members contributed to a good and open-minded atmosphere. It was fun working together which resulted in a product that everyone could feel proud of. Another factor that matter when talking about a good team work is transparency. Even if we didn't use tools that is frequently used in scrum (such as the scrum board), everyone knew what other group members worked with a the time. Probably this had to do with our ambition to always work together. Even conflicts and disagreements were solved easily. Through a democratic process, where everyone had the chance to voice his or her thoughts, we came up with the proper solution. Thankfully we were an odd number of members, which always resulted in a majority in situations where votes brought down the final determination.

What would we do differently next time

With a little perspective to the project, most things have worked out fantastic. No major conflicts, good planning and a great application as a product. It feels like everyone have taken their bad project experiences and tried hard not repeat it. No more panic programming the last week.

No more missed out features due to bad estimations. Even if we are overall satisfied with the working flow there has definitely occurred problems that could have been prevented.

Discussing scrum, there is a lot of different parts that we left out as you can read in the section named scrum. Next time we would use the burndown chart more frequently. We left this part out because we couldn't find the need for it in this project, since we sat together most of the time. Thus we did have a really good overview of the project without the need of the burndown chart.

To verify that the code worked as it should we used test driven development. During the project we developed test cases parallel to the actual application. These helped us a lot and we would probably have a higher focus on creating tests if we started a new project today. We would also try to develop fully functional tests before implementing code. This would benefit us because we would know directly when something was incorrectly implemented without even having to think. A large part of our application was pretty hard to write code tests for and therefore we made a lot of informal tests. What we didn't do was create a documentation standard for the informal tests to provide a structured environment for them. We would have done that if we were to redo the project today.

In this project we often referred to the bus factor. The bus factor basically means that all project members should have knowledge of every part the project contains. Having a low bus factor makes sure that the development of the project can continue even if a member, for some reason, cannot proceed. To minimize this we probably would have tried to vary the task between the sprints. By that we mean that everyone should make lesser contribution on every feature but contribute to more features. That would have given everyone a better understanding for each part and therefore not risk being dependent of a person to get a certain part done.

In the beginning of the project we made very little research on what tools to use. That gave us some early problems that easily could have been prevented. For example we tried out using Android Studio for the development process. We had some big issues using this and decided, after two or three days, that we would use Eclipse and IntelliJ IDEA instead. The problem was that we assumed Android Studio would work well without any specific research and this lost us useful time. So basically we would probably spend about a day to evaluate possible tools before actually starting to use them if we started a similar project today.

To be able to plan our weeks we made time estimations on how much time different parts of the project would take to implement. In the beginning we experienced that we had made some miscalculations, which was expected because we were all new to this method. We tried to reestimate how much time the upcoming features would take based on previous weeks experiences. We probably didn't spend enough time on this "reestimation" because the actual work we got done almost never met our plans. Sometimes we were able to do more and sometimes less. So we would give this part a little more focus to be able to make more realistic plans for each week.

Summary

To sum up this project we are very pleased with everything from the start, during the process and to the actual release product. We had our doubts that it would be hard to maintain a good communication between all group members because we were seven people. This turned out not to be the case. We are also very satisfied with the process where we used scrum for the first time and it actually turned out to work really good for our group. We didn't follow the scrum rules to a 100 percent but would probably try to if we would do this project again. We have earned a lot of valuable experiences about developing a product in a, for us, larger development team.