Post mortem report: Fleetspeak

Deadliest Trucks

By*: David Gustafsson, Patrik Haar, David Michaëlsson, Matz Larsson, Erik Pihl and Johan Segerlund*

Introduction

Fleetspeak is an Android application for VoIP communication specifically targeting truck drivers. The application allows multiple people to create or join rooms and talk to everyone in that room. Everything is designed with safety in mind and follows the NHTSA-guidelines to reduce driver distraction. This report concerns the work and processes that were made during the creation of Fleetspeak.

The processes and practices

We used scrum methodology when developing our application. For managing the scrum process, the web-application ScrumDesk was used. Weekly sprints were set up in the project, which meant a week for implementing features and tasks from the backlog.

The project consisted of two technical parts, an android application and a server. With this in mind we divided the workload into three parts. Two group members were responsible for the android GUI, two for implementing the server and two members worked with bridging the server and application.

We used GIT as a version control system to allow all project members to work simultaneously with the code. We worked with several branches each intending to contain the code of a specific feature.

Pair-programming was used when trying to find the various bugs and solve other problems that arose during development. It was also frequently used during merge of branches and git conflicts during the final stages of development.

We also had supervising meetings to simulate customer feedback and discuss implementation feasibility during development. These meetings were held with the assigned supervisor Viktor and Amir. Viktor gave us support with technical issues and Amir gave us advice on what features that was most feasible to focus on.

The time spent when developing Fleetspeak has not been evenly distributed throughout the project course. During the first weeks of the project the workflow was quite slow.
Part of it was due to the course planning, which did not give us much to work with in the beginning. There were also some issues with setting up the development environment, such as integrating AGA SDK.
The last weeks of the project was spent on coding and implementing the various features of the project, as well as fixing bugs. We did all our testing during the final stage of the project. We didn’t keep a log on how much time we spent working with the project. We

think 30 hours of work per week by each member is a reasonable estimate. There was more time spent in the last weeks than in the start of the project.

Requirements:

The first phase of the project was dedicated to finding an app idea that was achievable during the timeframe of the course and usable by a customer. Feedback from multiple sources such as course leaders and a truck driver was used to further develop the idea. As the app idea evolved the problem domain was defined. By the end of this phase we had a clear basis to move on to the next step, design.

Design:

In the second phase, the design of the project was defined from the requirements from the first phase. Some decisions on which tools and libraries to use was also decided, though some decisions were later changed due to time constraints and more effective alternatives.

Implementation:

Implementation was done in weeklong iterations where we wanted a runnable version after every iteration. In the beginning of every iteration we decided on which features were most important and began working on them.
During the implementation stage of the project the requirements of the application where discussed and changed. This was a result of the new input from supervisors and new ideas

Quality assurance:

During the last weeks of the project, time were spent on quality assurance. This meant bug fixing and writing various documentations, improving the structure and adding good comments in the code.

Techniques and processes

Pair programming

This technique helped to coordinate the knowledge of two members to a solving specific problem. That way the problem could be approached from different perspectives and an innovative solution could be found. This was often used when dealing with sound and network functionality issues. The pair programming also helped in the search for bugs and faulty code, since project members often realized what the error was when telling others about how their code worked. Utilising pair programming was also beneficial for sharing knowledge among the team members. This made merging branches from different areas easier.

There were a few disadvantages of this technique. The main one was that pair programming requires a considerable amount of time for explanation and discussion before coding. It also requires project members to stop development in their own specific area of development to help others. These were very small disadvantages since the pair programming was so effective at eliminating the problematic code and solve difficult tasks.

The majority of the time we spent during this project we worked in the same room. When issues occurred we could therefore easily move around and help each other and solve problems directly.

Even though the pair programming was a very effective tool for us, given a different situation it could be less effective. An example could be a project where group members have different schedules or work locations. It can also be less effective in earlier stages of development when the goal is to produce a large amount of code. Especially if the code concerns an area you have experience working with.

In future projects we believe strongly in using pair programming when handling faulty code and solving complex tasks when coding.

Scrum:

Scrum was a useful technique as it gave us a good understanding of what we had to do each week. Having a group conversation about these problems increased the understanding of current tasks and updated each member on the progress of the different parts of the project. Compared to the waterfall model, scrum did not require a huge pile of documentation before we could start coding.

One disadvantage was that the time given was too short to use scrum effectively. After the initial planning there was about five weeks of time coding divided into four sprints. Since most of the areas within the project was new stuff for most of the team members, it was hard to judge if tasks would take one or three days which, due to the short sprints, messed up the planning for that week.

The tool that we used to track our scrum progress, Scrumdesk, had some issues. For instance, there was some bugs and confusing management methods concerning the tool itself. Still, those problems could be due to that Scrumdesk was in beta-state when we worked with it. It should also be noted that Scrumdesk's web support was really helpful, and often fixed the issues that we had or gave us a good response. Nevertheless, using Scrumdesk for this project was helpful, despite the above listed issues.

Scrum was efficient since it gave a better structure to the weeks. In that way, people had a better view of what they should work with and do during the week. In fact, this especially helped us during the earlier weeks with motivation and completing what we set up as that weeks goals. Although, during the later weeks, the planning fell due to limited time.
Scrum worked well enough to be our software development method of choice in future projects, but it would probably work better with two-week sprints instead one-week sprints. So for an optimal usage for this technique we feel would be about 3-6 members working (with implementation) for 6+ weeks.

In the future, smaller projects would not be worth the planning and setup of scrum even though it's one of the lighter techniques to use. Projects with more people would be very hard to handle without breaking it up into smaller groups and having someone dedicated to organizing the different groups.

Git:

Using Git was an effective way of handling the various version of the programs. Utilising the branching features in git made it easier to work on the different parts of the code simultaneously. This way the server and android sides could divide their work and commit syncs in a simple way without too many code conflicts.

There were some minor disadvantages though. Firstly, Git is a tool that requires some knowledge to use to its full potential. Secondly, some merging conflicts occurred that needed to be solved in order to make things work properly. This was not time efficient.

Overall, Git is an efficient tool to make sure everyone has access to dynamically changing work. Git is very suitable for the coding, but also for images and documents describing the work process. It is also excellent for backups and sharing data with others. These features makes Git a useful tool in future projects.

Git might not be as useful when working alone on a project, compared to working as a team. Though, there are still several useful commands which could be used when working alone, such as the revert and branching features.

Supervising meetings:

Meetings with a supervisor had a positive effect since one could get external opinions and views on the project. Scrum made it easy to use the input from our supervisors to adjust our project design. We also got motivated from this meetings, since progress had to be made each week.

Similarly to the disadvantage with scrum, supervising meetings requires significant progress every week to be rewarding.They also require to be scheduled in advance to make sure all group members and the supervisor can attend. Therefore it was not always time effective.

Supervisor meetings are suitable in most situations as they are able to give vital input towards development of the project from outside of the project group. Meetings can be useful to give a different perspective on ideas and problems. However, in projects with smaller scope, meetings would be less suitable as they require time to plan.

Workflow

Working with Scrum and meetings with supervisors were beneficial as they were rewarding and suitable to our situation. The idea to split up the group members to specialize in different areas worked out well. This led to group members becoming more knowledgeable in their own assigned areas and improving the group's expertise. This was important as most of the group members did not have prior knowledge in app development, network programming or sound mixing.

To improve our way of working we could have had a more thorough discussion of how to to connect the android application and the server. The lack of interfaces between the server and app made the integration in the later parts of the project more time consuming.

Reflections over non-process specific decisions

Efflux

For handling the RTP data, i.e. the sound packets, we used the open source java library Efflux. As with many other tools and processes, Efflux was in beta. This showed during development as there were trouble getting it to work with the other parts of the server. It also contained a lot of insecure code parts with bad exception handling. This required extra work and bugfixes, but in the end it worked as intended.

Server sound handling

Early in the project we decided to mix all audio streams on the server. This gave a positive effect on the network traffic since each client only needed one datastream at all times. Late in the project we realized that good sound mixing is a tough challenge which takes time to implement. The knowledge of this might have affected the above mentioned choice.

Sound mixing and audio codec

As mentioned before, the sound mixing was a tricky part. Right now, it mixes sound with very few filters to improve the sound quality. Naturally, this affects the result dramatically. Due to time constraints, we had to prioritize other things.
One way to improve this is to check for packet loss and calculate proper values for the lost packages. Another way could be to use other sound mixing algorithms or other audio codecs (Speex, Opus etc.).

Protocols

For this project, we used two protocols for transferring data: TCP and RTP.

TCP was used for sending the various commands between clients and the server. For example, if a user would like to change their name it's possible to send a command to the server, with the server changing the requested name. One useful TCP attribute, is that it is built around confirmed transfers. So that if a sent command is incorrect, the server will ask the client to send the command again. This makes TCP a reliable way of sending commands.
For the audio side, we used RTP to transfer sound data to the server. RTP is a useful protocol for sending large amounts of data that tolerates some packet loss, as long as the original order of the packets is preserved. This works well with our project since we want to transfer packets of PCMU encoded audio where packet loss a relatively small problem.

Refactor commandhandler to use a design structure.

As it is now we have a long “else if” series which identifies incoming commands and handles them right there after identification. This takes up a lot of space and makes it more confusing to add new commands. In the beginning it worked just fine, but when the amount of commands increased it became more and more apparent that we needed some kind of design pattern utilizing polymorphism. That would imply that each command is created as an own class according to the template pattern.

Group Interaction

The interactions of the group members were different during the project phases. At the start of the project we all worked on the same task: specifying the app idea. Decisions in this stage required discussions and input from all group members.

When the actual coding began, the members were split into groups with different responsibilities. As the project continued, more work and decisions were solved internally. The progress of the groups were presented on the weekly scrum meetings. Closer to the deadline of the project, more and more cooperation between groups was required.

The interactions between group members worked well in the discussions and the scrum meetings. It often led to productive outcomes and let the group members gather different input concerning design and implementation choices. The scrum meetings were good as they divided the workload between all members and made it easy to track progress. This resulted in every group member always having a work to do and a clear goal during the project.

Something that could have been improved was communicating the requirements of functionality between different areas of the application. In practical terms this would mean thorough and extensive interfaces between the server, application services and activities.

Future projects

We have learned several things that we could do better in similar future projects. Such as more frequent build releases, more consistent usage of scrum and testing the GUI with people outside of the project group.

Releasing a version of the project each week is something that would be helpful for both customers and developers. At first this would mean more stress on the developer, to code and release a build that they are content with. But with the less distance between releases, it would become easier to be content with smaller changes from the previous build. This would earlier bring issues and faulty design to light as well as the security of having an up-to-date and stable version.

Larger usage of scrum during the early stage of the project would enable a better workflow through the phase. It would give us a clear picture of what needs to be done and how to go about it. This was somewhat lacking in the beginning of our project as considerable amounts of time was spent on on planning and discussion on what to do next.

GUI testing with people outside of the group would have given us a clear picture if the application is easy to use and what design scheme is most fitting the application. In our project the we didn't have the time to make these GUI tests which made spotting user unfriendly design harder.

If we were to write a similar application concerning the same technical area we would also have a different situation than the one in this project. In the beginning of the course no one knew anything about sound-, network- or Android programming. Nowadays that has changed, we have at least basic knowledge about these components and will be able to handle them much better in other projects and development situations.

Conclusion

So what is our final conclusion of this entire project? Would we do it again? Would we tear down our bodies and brake our minds just to finsih this? The answer is yes. The awnser is yesh.