MOPED DAT255

The development process of MOPED autonomous driving, how it was achieved with agile methodology and reflections about the process.

Edvin Meijer, Felix Nordén, Zack Lundqvist, Hampus Carlsson, Jesper Blidkvist, Anton Kimfors, Marcus Lindvärn, Kevin Solovjov, Anthony Kalcic, Emil Jansson, Elias Burström, Hugo Ekelund

Gothenburg, Sweden 2017

Contents

1 Introduction

1.1 Background

An increasing amount of modern cars and trucks use micro computers called *"ECUs"* in order to optimize their performance. Adaptive Cruise Control and Platooning are two features which can be developed using these micro computers. The reason - to have achieve autonomous vehicles for real-world traffic, with the intent of reducing transportation cost, improving traffic safety and mitigating climate influence. However, a system of this kind is highly complex, expensive and difficult to test. The MOPED (Mobile Open Platform for Experimental Development) is a small-scale replica of this system, which can be used as a tool to simulate cars or trucks. This enables developers and engineers to efficiently and safely test and develop desired features. The MOPED platform allows for small-scale simulation in order to find problems that can occur in larger scale. In order to develop both adaptive cruise control and platooning, the teams will adopt the SCRUM-methodology and associated agile tools to structure and reflect upon the development process.

1.2 Purpose

The main purpose of this project is to learn how implement an agile methodology and work with a product which, over time, might have its requirements changed. This includes forming strategies to efficiently tackle problems that may arise, and being flexible regarding feature prioritization to meet the Product Owners' requirements. During the project we will learn more about platooning, autonomous systems and utilizing existing APIs. We also have purposes inherited from the course's memo, which we strive to follow[1].

2 Theory

2.1 Application of SCRUM and agile development

When working on large projects, a plan for how work and decisions should be made needs to be decided. Within engineering, the usual approach for this is the *"Waterfall approach"*. However, this approach to engineering is, in many ways, inefficient while working with software. With software, things are constantly evolving, a more flexible and adapting approach is needed. This approach is known as *"Agile development"*.

2.1.1 Teams, roles and the social contract

In this section, we will introduce the setup and strategies we have chosen for maintaining a functioning SCRUM-workflow.

2.1.1.1 Teams

Before any development started, the team identified a number of topics and responsibilities which could be assigned to individual teams. During the first 3 sprints these were the following:

- *Architecture* - Basic software architecture, responsible for setting up the feedback loop and for integrating the modules implemented by other teams. Also, this team has the responsibility to fetch and broadcast data from the MOPED's sensors.
- *App/communication* - Adapting the already existing app, creating a client and server for transmitting and receiving data to and fro the app and the architecture.
- *Math and logic* - Implementing an algorithm[2] in order to keep the MOPED in safe distance from a vehicle in front. Also, they are responsible for ensuring that the reactions are smooth and developing general interfaces for "black boxing" the algorithm from the Architects.

After the third sprint, the scope of each team narrowed down and we realized that not enough user stories nor tasks remained for each team. Therefore, we decided to allocate more members to our bottleneck, the architecture. Seven developers formed a new team, the MOPED-team, with the broader responsibility of maintaining the existing codebase of the MOPED, finishing the architecture and integrating new modules with the feedback loop. The four developers who were left formed the

OpenCV-team, with image recognition as their responsibility for the completion of the platooning feature.

2.1.1.2 Roles

Roles that have been decided according to the roles in traditional SCRUM methodology are:

- *SCRUM Master(SM)* - A member with the responsibility to make sure that all members of the development team understand and utilize the processes inherent to SCRUM[3]. The Scrum Master functions as the maintainer of the team's group dynamics. He/she is there to thwart stagnation and to care for the team's well being. In our adoption of SCRUM the SM acted as a chairman during meetings.
- *Development teams* - Groups consisting of developers associated with the project[3]. Each team has to inform the project manager about structural changes as to make sure that the PM can make important final decision.
- *Product Owner (PO)* - The stakeholder of the product. He/she has the responsibility to explain to the development team what their product vision is and to clarify what needs to be prioritized. According to Scrum Guides, a PO is responsible for setting up a backlog and give constant feedback on implemented stories. This will optimize the output from the development team and ensure that the stakeholder value is reached[3].

Roles that are not a part of traditional SCRUM methodology, but are necessary for this project:

- *Project Manager/leader (PM)* - Whereas the SM has the responsibility to maintain group dynamics, the Project Manager is responsible for having the knowledge of technical aspects associated with the product. This gives the PM more authoritative power. This is useful for situations where teams can not reach a consensus. The PM then takes the final decision.
- *Group representatives* - Each team had a representative who was supposed to have more knowledge about what the team was currently doing. Their task was then to represent their team and inform other representatives during meetings. This was an effort to establish better communication between the teams.

The roles are divided as such:
- Product Owner: Kenneth Lind
- SCRUM Master: Edvin Meijer
- Project leader/manager: Anthony Kalcic
- Group representatives; Emil Jansson, Zack Lundqvist, Edvin Meijer, Anthony Kalcic

2.1.1.3 The social contract

The social contract was written during the early stages of the project[4] and before the first sprint. After the first sprint, friction occurred between some teams and it was decided that alterations of the social contract were needed. A meeting with the entire team was held and the "Smaller group requirements"-section was added. The main addition was concerning communication and interaction between the smaller teams.

One concern was that a lot of team members did not read the social contract. This became apparent when set guidelines were forgotten. It did not result in any conflict, but did create some frustration. Had the product been developed over a longer time, these frustrations could potentially have become bigger problems and resulted in conflicts.

2.1.2 Adoption of the agile methodology (SCRUM)

We began by dividing our twelve team members into three smaller teams, each with four persons. Each team also then chose their representative.

We were striving to keep the core elements of the SCRUM methodology and adapt them to our needs. In essence, we did one-week sprints where we started with a sprint planning. Later on, a sprint review was held with the PO. Lastly, each sprint ended with a sprint retrospective. Additional SCRUM-tools used were stand-ups for our group representatives twice a week. The SM was attending all of the SCRUM-related meetings. There was a vague idea of daily stand-ups, but it was never realized as a regular part of our routine.

A sprint backlog was created and/or updated after each sprint planning. There was a common backlog containing the epics specifying the Product Owner's final vision.

Before development started, a Definition of Done (DoD) [5] was written. This was to prevent faulty code and to prevent incomplete features to be present during the demonstration. Our DoD focused on software quality and doing as much as possible to verify that the code quality held a high standard.

2.1.3 Other used practices

Except for adopting SCRUM into our development process, we have also used a handful of other good practices for assuring that we hold a certain code quality. The practices we have used are the following:

- Pair Programming
- Code Reviews
- Git Workflow
- Continuous integration
- Test-driven Development
- Behaviour-driven Development

- Software Versioning

Each of these practices are described more in-depth in our Code Quality document at the repository.

2.1.4 Time distribution

A table showing an approximation of the amount of time that each member put into each sprint.

	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5
Zack	24 h	18 h	25 h	14 h	12 h
Edvin	26 h	21 h	22 h	16 h	20 h
Hampus	24 h	16 h	23 h	20 h	15 h
Felix	37 h	43 h	28 h	23 h	25 h
Jesper	21 h	15 h	21 h	17 h	14 h
Anton	12 h	18 h	20 h	8 h	0 h
Marcus	19 h	20 h	13 h	21 h	23
Kevin	5 h	23 h	26 h	18 h	15 h
Emil	22 h	19 h	15 h	16 h	15 h
Elias	19 h	18 h	13 h	21 h	25 h
Hugo	22h	20h	14h	13h	14h

2.1.5 Velocity and task breakdown

Before the first sprint, a decision was made that each team should produce 100 points by the end of every sprint, as long as the four person teams remained. Instead of estimating a new value for each week, we settled on adjusting our story points instead. Since the learning experience was the most important aspect of the process, estimating story points in relation to the current conditions was regarded as a priority.

When teams needed to be more flexible and new team were created, an estimation was made that every pair of developers could produce about 50 points a sprint. This meant that a new seven-person team would produce approximately 175 points per sprint.

2.2 Tools, APIs and usage of existing technologies

In the early stages of the project, we had a very steep learning curve in regards to the platform. The first week was mainly used for researching the existing platform and the libraries which the MOPED depended on.

2.2.1 Android application

The app/communications team started by researching the Android application used for maneuvering the MOPED contained in the MOPED GitHub repository.

This application was made for phones utilizing the Android OS and serves as a simple remote control, sending data needed for velocity changes and steering of the MOPED.

The team needed to adapt to how Android applications are written, which can differ greatly from a "normal" Java program. Android applications are built around *Activities*, which force a developer to work with a *Model-View-View-model* (MVVM) structure. The team has mostly worked with the *Model-View-Controller* (MVC) structure in previous Java Projects. The difference is mainly in how you split up various classes, which initially caused slight confusion.

The UI creation tools provided by Android Studio proved to be incredibly useful since the original application was written for an outdated version of Android and the old UI needed to be replaced and updated in order to allow for easy changes.

2.2.2 Communication package

There is a communication layer between the application and MOPED architecture. The communication package is built upon Java's socket library which provides two-way communication between a client and a server. To keep the package separate from the others, a generic interface was written. This later turned out to be a good decision as we managed to easily transfer the already existing steering- and throttling features from the old app onto our own.

2.2.3 MOPED architecture

For the group's work to be as efficient and value-driven as possible, the architecture team wanted to reuse as much of the already existing platform as possible. However, we also wanted to ascertain that our architecture would adhere to our code quality key values. I.e that the code is *extensible*, *modular*, *readable*, *reusable* and *testable*.

Initially, we researched the existing platform using the provided documentation and used the first sprint to get an understanding of how the TCU, VCU and SCU interacted with each other and the basic concepts regarding the CAN-bus[6], the provided server and SQUAWK[7].

After reading through the whole documentation and also the SQUAWK-documentation[8], we found the python scripts that was used by the TCU to interact with the other ECUs. We first decided to have this as a backup plan if we would not be able to configure the server in the near future, but quickly changed direction to it as we saw another group being able to manipulate their MOPED during the first sprint review. This quick decision led to an array of positive outcomes later on, which can be found in chapter 4, *Reflections*.

As we now had found a way for the ECUs to communicate, the actual architecture had to be designed for the TCU. With our key values in mind, the team began

developing a core domain model and came up with a modular and robust solution that has not changed conceptually during the remainder of the project.
A more concrete description of the core domain can be found as a UML diagram in the appendices, but the core concept of it is to have the following:

- A behaviour focused state controller, which handles the states that the MOPED should be in
- Different actions for each behaviour to utilize depending on the surrounding circumstances
- A single communication port for incoming and outgoing data to and fro the core domain
- A car controller for setting the values of the vehicle's steering and throttle components
- Sensor components for parsing the data from the supplied sensors and then transmitting this data to the other modules
- A process runner for interacting with the VCU and SCU (or the Arduino later on) using JAVA-code

By having these distinct and separated modules, our architecture is very flexible. For instance, additional sensors can easily be implemented by implementing the used interfaces. The process runner module can also be switched out for another process runner if the way of communicating between the ECUs would change.

Our design decisions entail our architecture is adhering to the $S.O.L.I.D$ principles[9]. Each module has their single responsibility (S), and the components are open for extension, but closed for modification (O). Every extensible class has been extended if, and only if, the need for both polymorphism and code reuse has been present (L). Each module has different interfaces for others to depend on, only supplying tools that are needed for their specific task (I). Lastly, in conjunction with the previous principle, each module depends on interfaces rather than concrete implementations as to allow for modularity (D).

2.2.4 PID-controller

In order to create a controlled and smooth motion the team decided on developing what is known as PID[2] for the MOPED. Essentially this software is based on a feedback loop and is used in a variety of industrial control systems.

The main purpose of using a PID-controller in the MOPED is to avoid, or at least try to counter, oscillation that might occur in platooning. The oscillation is what appeared to be one of the largest challenges in getting platooning to work smoothly. The fact that the PID-controller is a general mathematical solution means that it an be applied in a variety of different situations, which was beneficial to our product since we wanted to control both the longitudinal and lateral axes of the MOPED when it is in the platooning-state.

PID is based on calculating an *error*, which is the difference between the actual value and the desired value. In our case, these values are the distance which we want to keep to the object in front, the desired value, and the actual distance to the object in front, the actual value. The controller then applies a correction which is determined by the three parts of PID, the **P**roportional, the **I**ntegral and the **D**erivative. These values are determined by the error value.

Firstly, the proportional component of the controller is based directly on the error, since it takes the error and multiplies it by a constant that is set to a fixed value. What this component means is that if the error (the difference between the desired value and the actual value) is large and positive, the resulting P-value will be large and positive.

Secondly, the integral component is determined by the history of the controlled movement. For our MOPED this would mean that for every time the feedback loop hits and the error exists(the actual distance to the object in front is not equal to the desired value), the integral value will grow in proportion to the error, getting bigger and bigger until the error is no longer present.

Lastly, the derivative component. This component is responsible for keeping the rate of change of the correction controlled. This is done by using the current speed of change, the derivative, to estimate how the error value will develop in the near future. What this means in reality is that large accelerations will be dampened which results in a smoother change of the correction.

In the end, these components are added together and the result is the correction which is then applied to what the PID is regulating. By using this technology for our product, we would able to achieve a smooth and controlled movement for the MOPED in its ACC-state.

2.2.5 Computer vision

In the context of this report, computer vision refers to interpreting the pixel data of an image. In our specific setup we use the Raspberry Pi camera-module and the library OpenCV[10] to do the actual image processing.

One common application of computer vision is to track and identify changes in an object present in the images. In our case, we track the movement of a specific object, such as a ball.

Our setup involves finding green areas in a given image, and then selecting the largest one in the shape of a circle. This is then processed and passed on to the PID in the form of an offset from the image center.

2.2.6 MOPED hardware

The MOPED was originally controlled by three Raspberry Pis, each controlling a specific system. This was later changed for a setup consisting of an Arduino and one Raspberry Pi.

The main difference between a Raspberry Pi and an Arduino is that the latter is a micro controller and thus does not have an operating system. As such it can run code more efficiently without having to deal with OS interrupts.The Raspberry on the other hand is a micro computer and has the advantages of more computing power and since it has an OS the ability to run complex software such as OpenCV.

3 Results

During the presentation the MOPED was able to perform as expected by following the MOPED in front of it and keeping a reasonable distance.

3.1 Results retrospectives

These are the things that we came up with on our retrospectives: https://github.com/felixnorden/moppepojkar/blob/master/documentation/retrospectives-results.md

3.2 Results KPI:s

The results from our KPI "Produced velocity" can be found at: https://docs.google.com/spreadsheets/d/1D-e4eu4Ox63KY1qPNM5qqKBDRvf7SXSrMOx7200cRGw/edit?usp=sharing".

Since the KPI for our individual velocity was for the team and to reflect and not for measuring it will be kept secret. The SM knew of the stress levels of each member and this was the only data that was supervised. The stress level was high the first two weeks but then went down to normal levels.

On the KPI for Product owner satisfaction we only got one number in sprint three: a seven.

The results from our KPI for code coverage can be found at: https://github.com/felixnorden/moppepojkar/blob/master/documentation/code-coverage.md

4
Reflections

Our reflections and things learned from and during the development of the MOPED utilizing SCRUM and an agile development process.

4.1 General reflections

This section covers reflections regarding topics which are not directly related to a specific section of the report, but instead brings up topics which are related to the prototype and the decisions we have made in order to produce the demoed product.

4.1.1 Hardware swap

Due to unforeseen circumstances, the CAN network on the MOPED stopped working during the third sprint. This was a huge setback as it limited code testing on the hardware, which then halted the development process. A temporary hardware fix was implemented with an Arduino acting as the VCU, instead of a Raspberry Pi. Exchanging the two Raspberry Pis for one Arduino both reduced weight and the power consumption of the MOPED.

Our hardware fix was promptly made into a permanent solution as the Arduino gave a significant performance increase compared to the old VCU. It had better stability as well, resulting in no major hardware issues in the following sprints. This not only made our development progress faster, but also resulted in better performance of the developed features. In conjunction with this improvement, the stakeholder's value also increase, as the features of the product had improved significantly.

4.1.2 Computer vision reflection

One of the biggest hurdles to overcome was installing OpenCV[10] itself on a Raspberry Pi. We did however thanks to the large Raspberry Pi and OpenCV community find a precompiled solution which we were able to use. One of the issues that remained was compatibility with the ISO provided at the start of the project, we were however able to upgrade it to a newer version.

During the project we tested several different tracking methods and objects to track. It was concluded early on that QR-codes while easy to read using existing libraries would not be useful in our specific scenario. For example there is the issue that when a vehicle is moving fast the code gets distorted and hard to read, this is in part due to the frame rate of the PI camera used to take the pictures being relatively low.

A green circle turned out to be much easier to track when moving. This approach did however suffer from similar problems as the QR-code. At weird angles the circle warped and the calculated distance and offset from the camera turned out to be incorrect. Therefore the flat paper was exchanged for a green ball which in a 2D picture would be seen as a green circle from any angle. However, this turned out to have some problems as the ultrasound sensor still required something flat behind the ball to track accurately. This was resolved by placing the ball inside a stack of papers. The preferred option would have been a transparent surface in front of the ball as this temporary solution occludes the ball from some angels.

There are different ways of tracking circles and we tried two of them. One was through circles, which worked really well. However, this used a bit to much of the Raspberry Pi's processor. The other method, and the one later used, was to separate an image by color and then run it through multiple filters to later estimate the biggest circle within the remaining image.

The tracked color was decided to be green, as cameras[11] often contain more sensors for detecting green colors. A range specified in HSV values were specified in which the color of the ball was presumed to reside. These values were hard to decide since depending on the lighting, the ball could be a different shade of green.

Another important thing to balance was the frame rate versus the resolution of the image. While higher resolution gives better tracking, it reduces the frame rate in the sense that it requires more processing for the Raspberry Pi, which delays the tracking.

4.1.3 Raspberry Pi as a platform

This section reflects on the benefits and trade-offs regarding the usage of the three Raspberry Pis for the different ECUs.

4.1.3.1 Performance

The group noted early on when working with Raspberry Pis, that utilizing its four cores is critical to achieving good performance. Instead of running everything on one thread, parts of the program could be split into multiple. For example, running image processing and video capturing on two separate threads, instead of one, improved the performance by a considerable amount.

4.1.3.2 SSH and remote desktop

During development it was very helpful to be able to access the Pi. While this could have been done by connecting and disconnecting a screen or a keyboard, a much simpler way is by using Secure Shell (SSH) and setting up a remote desktop. If performance is not the highest priority there is also the possibility of using remote desktop to directly view the desktop of the Pi, which was very helpful when setting up the computer vision part of the project as this enabled us to see all the imaged directly.

4.1.4 Conflict

At an early stage of the project, a conflict occurred between two of the development teams. This was due to lack of communication. However, thanks to our good retrospectives and the SM's mediation between the teams, we had a quick and painless resolution. Instead of having people grumbling behind each others backs, the period after the conflict was very productive and the mood has been extremely positive since.

The important lesson from this is that having friction and conflicts in a group can be a good thing, because it forces the team to cooperate and actually helps the team building process. We have also noted that it is good to bring disagreements to the surface as early as possible, as this will allow the team to work efficiently as quickly as possible. Having these conflicts at later stages could have had worse consequences as the frustration in the group could have increased over time. The atmosphere of the team could have deteriorated, since unresolved disagreements tend to create a negative perception of the one(s) you disagree with.

Furthermore, as the disagreement was between two teams, an "Us vs. Them"-mentality was starting to develop. This had the potential to reinforce negative thoughts within the teams about the "adversary" in the conflict. This was avoided by improved communication between both parts of the conflict, leading to a better work environment.

After this incident, we learned how to work better as a group and to take everyone's opinion into account. By sharing our thoughts and feelings about issues as they emerged, further conflict was avoided.

4.1.5 Code reviews

The group's Definition of Done included a section that stated that all code had to be reviewed by at least one group member which had not worked on the code previously. It was established early in the project that this was to be done through pull requests on GitHub or in person. This methodology was the official way to handle code reviews for the entire project but it was not always strictly followed.

Some group members expressed that the code was reviewed to slowly as other team members were busy with their own sprint tasks and could not take the time to sit down and look at the already produced code. As the deadline came closer this pattern was reinforced and code reviewing was not being applied in practice anymore. This allowed the development of new functionality to be developed more quickly but might have also played a part in allowing bugs to slip into the code base.

4.2 Reflections KPIs

In this section, we reflect on the results which we have gathered from our three KPIs which we set in the beginning of the project.

4.2.1 Velocity and burn up/down

This KPI is the one with the most clear results. We did not deliver what was expected of us in a majority of the sprints in according to the burn-up charts. There were a lot of factors for this, the main reason being our Definition of Done (DoD). It was decided that for being marked as "Done" it needed to be verified as working in the MOPED/GUI which was harder than we thought. Because of architecture being a bottleneck it took over three weeks before some tasks could be verified on the MOPED. Though our DoD was hard to meet we agree that it was necessary to keep the quality high and assure that faulty code was kept out of the master branch. However, depending on how communication with the project owner is handled, this could potentially make it look like very little progress is being made.

The sprint planning also played a big part in our low velocity. Since we had our points fixed at 100 we should have concentrated on correcting our story points and the amount of stories for each sprint. This is something that happened to some degree, but it felt like the deadline we had was close by. To even arrive at the point which we had set out to reach, we could not slow down the pace even if we wanted to. When slowing down, we should have given our stories higher points, but we tried to be realistic and maybe underestimated some stories relatively to the time that we had. We did make the mistake to compare what we had to a full time job even if we knew that we only had 20 hours a week.

4.2.2 Code coverage

Because of bad guidelines on how to properly measure code coverage, and less than optimal test driven development, we only took one measurement of code coverage. This measurement was taken only after the development was completed.

This results in code coverage not being used properly as a KPI, as there are nothing to compare our only measurement to. The use of TDD would mean that an optimal development process should strive for high percentages of code coverage through all sprints.

This KPI would also show how much TDD was actually used, because TDD being correctly used results in high code coverage. When used correctly, code coverage would most likely be a great indicator for the completeness of the current software.

We also realized that we should have used stricter guidelines on how and what to measure. The remote control application is a good example on something we should have excluded from the KPI at the beginning, because of difficulties testing code coverage in an android app.

4.2.3 Individual survey

The individual surveys was meant as a weekly reflection concerning stress, quality, effort and general thoughts. Results from the survey were collected by the SM, and was not shown to the rest of the group.

This survey did not generate a lot of visible data. Therefore, it is hard to give an evaluation of how "good" it was depending on the data. When this KPI was decided, we wanted a way to determine the mental state of the group. There was a discussion about whether or not the data should be looked upon and be evaluated but we came up with the conclusion that there were no reason for this.

We feel like this KPI has been useful since each member could gather data on the effort put down each week and compare to earlier week. This helped to know when for example the quality worsened and the reason behind it.

It could have been clearer what some questions really meant, it was easy for each member to interpret them in their own way. The questions could also have been more directed towards the specific parts of the project. We used a scale from one to ten when the rating stress level or sprint difficulty. This was somewhat limiting, an addition could have been that the number could have been accompanied by a short comment to improve the reflection and reason for the chosen number.

4.3 Retrospectives

The teams used weekly retrospectives as a means to communicate their needs and opinions to the rest of the team. Here follows the group's reflection on the subject.

4.3.1 SCRUM Master's reflection

As the SCRUM master I felt that the first few retrospectives were shaky and that there was no structure. For the first three sprints each member answered these four questions on the whiteboard; "What went well?", "What didn't go as well?", "What have you learned?" and "What is puzzling you?". The results differed a lot from the teams, some were open and could have a discussion right from the start but others had a harder time opening up.

For each sprint I became more comfortable in my role and the structure became better and we could hold more efficient retrospectives. What I realized was that a lot of the remarks on the whiteboard was related to the technical aspects of the project. Seldom answers were regarding the group dynamic or the methodology and from my interpretation of SCRUM this is the most important aspect of the retrospective. I read an article about how to hold more efficient retrospectives [12] that recommended to let the team members answer to what they should start-, continue-, stop with to improve the team's work but focusing on the process and the group dynamic. I tried to make it clear that this type of retrospective is more focused on agile methodology and group dynamic. What happened was that most of the technical remarks disappeared and that the focus became the process instead. What was worse with the new approach was that it needed reflection and engagement in the process also what SCRUM is and should be. Not a a lot of members had any interest in this and, therefore, there were not a lot of remarks on the way we worked. This led to the retrospectives falling a little short. Even so, when things came up it was in the right direction and became clear improvements in the way we work. These may have come up on the first type of retrospective but since the start/stop/continue approach was more time efficient, I preferred the latter and would have continued with it.

4.3.2 Group's reflection

Retrospectives quickly developed a good structure at an early stage and many members feel like this part of SCRUM is the part which is showing the best results. The methodology emerged within the group in a natural fashion, which implies that all members of the group has the same appreciation of having retrospectives. A lot of good topics regarding the atmosphere in the group were brought up, which cascaded into a lot of discussion and reflection regarding the communication within and between the smaller teams.

However, while the retrospectives of the early sprints were greatly appreciated, some members felt that the meetings in later sprints did not quite hold the same quality. When we joined two of the teams into one, the retrospectives stepped away from the personal aspect of the retrospective and more into a team-focused one. When the groups were smaller, the members felt that they could describe and discuss their personal experiences from the sprint more thoroughly, which led to more personal development. With larger groups attending the retrospectives, the focus shifted from the individual's experience to the group's.

Furthermore, the structure of the meetings is one thing that could have been more dynamic. This would be advantageous since the teams have had very different tasks for which the general formula of the retrospective might not have been the best for every specific team.

In addition, the structure of the retrospectives changed midway through the project, which some members felt had a negative impact. As stated earlier, the first version of the structure had a more personal touch with questions like "What did **I** learn this sprint?" and "What confused **me**?". This iteration of the retrospective led to a more personal development which was appreciated by many. In contrary to this, the latter approach to the retrospectives had more group-focused questions, such as "What should **we** continue doing?" and "What should **we** start doing?". This approach is focused more working as a group and trying to get work done. While this might have led to a more efficient work environment, a discussion with the group concerning changing the structure of the retrospectives would have been appreciated, considering the importance of the retrospectives.

4.4 Sprint reviews at Lindholmen

While the PO was not present on the majority of the reviews, the times when he was were very meaningful and productive. Just like what Christian talked about from his guest lecture by IGDB, we realized how much clearer the goal became when we spoke with the PO directly. Especially as we discussed what we planned to do and got immediate feedback as he spoke his mind. Without this opportunity we could have gotten stuck in the development process. Or worse, we could have focused on features that we as developers wanted to have on the MOPED, which may not correspond with what the user would see as useful features in the finished product. Since the PO could not attend all of the reviews, it could have been better to have sprints of a two week duration since there would be more available time to plan and administrate the sprint. Additionally, it would have made it easier to find time for retrospectives and reflection upon the sprint. More on this can be found in section 4.5.3.3.

The best part about the reviews was that a lot of our classmates were present at the same location and time. This opened up for discussions with the other groups, which sometimes entailed us realizing that our current problems had already been solved by another group which then shared their solution with us, or vice versa. One thing that we will take with us for the future is the advantage of working together in the same location since this makes exchanging information and helping each other solve problems exponentially easier.

The Scrum of Scrums-meetings (SoS) were appreciated and made it easier for groups to communicate what they were working on and avoiding unnecessary duplicate work. Since the SoS:s gathered the teams' communication leaders, decisions about standards regarding the demonstration could easily be made. However, there could be improvements. One way to make the SoS:s better would be to have all attending SMs try to understand more of the issue and then coming to a unified consensus before ending the meeting. Some meetings ended with the decision being postponed until the following week, which made some details in the development uncertain and relevant information sometimes got forgotten.

The testing phases also gave useful information as we got to know which groups we could test our MOPED with. Additionally, we could exchange knowledge about, and give possible solutions for, the current issues faced by both groups.

4.5 What was, what is and why this outcome?

In this section, we compare our top three conclusions from the LEGO exercise and their hardships to how we tackle them in mid-project and after final product.

4.5.1 Learnings from the LEGO Exercise

After our first encounter with SCRUM during the LEGO exercise, our group had quite a few different learnings that we took with from all the impressions that we had been exposed to. Three of the most prevalent topics which we reflected upon was:

- Organization and work delegation is key to produce value efficiently, especially under pressure
- Retrospectives are crucial for having value producing sprints in succession
- Estimating the required resources by velocity is very difficult

4.5.2 Half-time reflection

When we reached the halfway mark of the course, a lot had happened. We all felt that we had gathered much more experience under our belts and that we had learned to make better use of the tools which we had been given during the course's introduction. In terms of the aforementioned three learnings, we had the following headlines in our half-time reflection which touch on the same topics:

- *"Dynamic teamwork"* and *"The grand conflict"*
- *"Retrospectives"*
- *"Sprint Planning"* with the subsections *"Fibonacci's sequence and Planning Poker"* and *"INVEST"*

4.5.3 Today's experience and conclusions

4.5.3.1 Teamwork and task delegation

From the very beginning, we all had a consensus that working 12 people in a single group would be quite difficult. The LEGO exercise proved this further, as we had a hard time adapting to the sprints' duration and could not produce any value until the second to last sprint. However, this also pointed out that we were able to cope with the pressure and adapt quite fast to the overhanging pressure. From that point on, we continued our work with the mindset that our teamwork would improve as we built up more experience.

This initial mindset became quite difficult to work with and quickly came to a change as we hit *"The grand conflict"*, (section 4.1.4), which also is described further in the half-time reflection. To think in this way was a bit childish and naive, as problems are not solved by themselves, but by hard work and effort from all involved parties.

Although we have learned quite a bit from this, there are two prominent learnings that we all have taken with us. Firstly, hardships and conflicts are very effective for group dynamics to evolve quickly and for the team to create and affirm clearer boundaries. In our case, communication became more streamlined and everybody had a more professional mindset regarding the project and the team. Nevertheless, having these conflicts are quite risky and could make the group fall apart completely if they were to be too grand. The important thing to remember is that it is not only one part's fault when a conflict arises and that both parties need to understand what they did wrong in order for a resolution occur. If, and only if, the resolution is beneficial to both parties, then the group will evolve at a faster rate and grow deeper bonds.

Secondly, splitting the group into smaller teams were effective for developing in parallel, but was also detrimental to the group's overall health. By this we mean that the decision to divide the group into three teams in the beginning was great in the aspect of delivering more value to the PO. However, when we merged two teams together to work on the MOPED, the old teams remained subconsciously, which led to the members not feeling up for the task of learning what the others had worked with. This led to a big bottleneck during sprint four and five, which could have been avoided if we had worked more dynamically earlier on as we discuss in our half-time reflection at *"Dynamic teamwork"*. Had we instead incrementally shared each teams' code base with the others and maybe had looser boundaries between the teams, we would have had more efficiency during these last two sprints as the learning curve would not have been as steep and frightening to the unfamiliar eye.

4.5.3.2 Retrospectives, the good and the bad parts

After the LEGO exercise, we all had come to the conclusion that retrospectives were a vital component in order to develop an efficient and evolving workflow in SCRUM. After each sprint, we became more and more organized and, as we mentioned in "Teamwork and task delegation" 4.5.3.1, managed to produce value efficiently after a few tries. This was probably due to us gathering up resourceful information from the PO at the sprint reviews, which we then used in our retrospectives to develop better tactics for the next sprint.

Midway of the project we had continued to have good retrospectives, which is described more in-depth in section 4.3, "Retrospectives". However, as mentioned above, in conjunction with the merge of two teams, the retrospectives became less efficient as we switched to the new approach. The reason for this change was time. As we had doubled our members, the old retrospectives would take twice as long and time was one thing we felt we were already short on.

In retrospect, this new approach was not very efficient at informing the team of each member's problems and insights. However, it was good for focusing on how the team should cooperate as a whole. The former approach was of great use, but demanded a lot of time. While we had the old approach, we also seemed to take each others' thoughts more to heart and really try to uphold our set team goals for the upcoming sprint. When we instead did the new approach, the goals were more thrown out there with the focus on just setting goals and not on upholding them.

However, this could be due to other factors, such as the group doubling in size and being under constant time pressure.

As a conclusion, combining the two would probably make a pretty efficient approach for retrospectives as it would focus on both the individual member's insights and the team dynamics as a whole. Nevertheless, for this to be efficient, time needs to be available. Therefore, this combined approach would be a better fit for sprints of a longer duration, such as two weeks or longer. In addition, we still think that holding retrospectives is an, if not the most, important part of SCRUM. Doing this combined approach could probably have entailed better group dynamics during the project and if we would have been given the opportunity to set our sprint lengths ourselves, such as the teams at Zenuity, then bumping up the length to two weeks and having these longer retrospectives would not be a bad idea.

4.5.3.3 Planning, stand ups and being realistic

Our most prevalent learning from the LEGO exercise was something along the terms with *"Estimating is hard"*. There were so many different factors that needed to be taken into account for an accurate estimation. Instead of thinking about factors such as how we would get the required LEGOs and whom to run off to to get it, we more or less only estimated the difficulty of producing the elicited item.

At the half-time mark of the project, we had changed our view on estimation from *"Estimating is hard"* to *"Estimating is easy, doing it accurately is the hard part"*. We had begun to hit our estimations more accurately and felt like we had gained some momentum. One of the main factors behind our improving results was our ability to split our user stories more vertically and making them smaller and more tangible. Another was simply us building up more experience to back our reasoning with. More on this topic can be found in our half-time reflection under *"Sprint planning"*.

As for what we bring with us for the future, there are a two primary things we see as very valuable. The first is that estimations are just that, estimations. It is impossible to always accurately estimate the resources needed for tasks that depend on multiple factors, especially if some factors are human. Therefore, it is better to overestimate rather than underestimating, as there probably are factors that have not been taken into account. Also, by overestimating a task by a bit, the risk of not finishing the task and having a sprint backloglog full of unfinished items decreases, which then mitigates the risk of dejecting the team.

A second thing is that researching and adopting more of best practices that are available for a certain tool, such as daily stand ups, the INVEST-criteria and Planning poker in this case, is a good thing to do at an early stage. This became especially clear when we had the guest lecture about configuration management by Volvo. During the lecture, we found out that we had adopted a very efficient workflow of best practices when regarding the actual development process, which can be found at section 2.1.3. What we did lack, however, was a more efficient way for holding the sprint plannings. Had we instead taken Planning poker or INVEST more to heart and spent the time necessary to plan more in-depth, then maybe our product's outcome would have been different, for better or for worse. Nevertheless, one thing that is for certain is that if we would have had better estimations, which these tools encourage, then the stress levels that pressured some of our members during the sprints could have been mitigated significantly and therefore made the work environment more enjoyable.

Bibliography

[1] H. Burden and J.-P. Steghöfer, "Course pm for software engineering project (dat255/dit543) 7.5 hec, autumn 2017." [Online]. Available: https://github.com/hburden/DAT255

[2] T. S. Moppepojkar, "Pid-regulator." [Online]. Available: https://sv.wikipedia.org/wiki/PID-regulator

[3] Scrum.Org and ScrumInc, "The scrum guide™ - the scrum team." [Online]. Available: http://www.scrumguides.org/scrum-guide.html#team-sm

[4] T. S. Moppepojkar, "The social contract." [Online]. Available: https://github.com/felixnorden/moppepojkar/blob/master/documentation/social-contract.md

[5] ——, "Definition of done." [Online]. Available: https://github.com/felixnorden/moppepojkar/blob/master/documentation/definition-of-done.md

[6] W. community, "Can bus." [Online]. Available: https://en.wikipedia.org/wiki/CAN_bus

[7] S. Microsystems, "Squawk virtual machine." [Online]. Available: https://en.wikipedia.org/wiki/Squawk_virtual_machine

[8] ——, "The squawk system." [Online]. Available: https://github.com/sics-sse/moped/blob/master/squawk/doc/TheSquawkSystem-Sep02.pdf

[9] M. Feathers, "S.o.l.i.d principles of oo-design." [Online]. Available: https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)

[10] O. team, "Opencv library." [Online]. Available: https://opencv.org/

[11] Wikipedia, "Bayer filter." [Online]. Available: https://en.wikipedia.org/wiki/Bayer_filter

[12] M. Cohn, "A simple way to run a sprint retrospective." [Online]. Available: https://www.mountaingoatsoftware.com/blog/a-simple-way-to-run-a-sprint-retrospective