

UNIVERSITY OF LJUBLJANA  
FACULTY OF ECONOMICS

MASTER'S THESIS

MATEJ DOMADOVNIK



UNIVERSITY OF LJUBLJANA  
FACULTY OF ECONOMICS

MASTER'S THESIS

**AGILE SOFTWARE DEVELOPMENT METHODS WITH A CASE  
STUDY OF SCRUM METHOD AT RENAULT SAS**

Ljubljana, November 2017

MATEJ DOMADOVNIK

## AUTHORSHIP STATEMENT

The undersigned Matej Domadovnik a student at the University of Ljubljana, Faculty of Economics, (hereafter: FELU), author of this written final work of studies with the title Agile Software Development Methods with a Case Study of Scrum Method at Renault SAS, prepared under supervision of Talib Damij, PhD.

### DECLARE

1. this written final work of studies to be based on the results of my own research;
2. the printed form of this written final work of studies to be identical to its electronic form;
3. the text of this written final work of studies to be language-edited and technically in adherence with the FELU's Technical Guidelines for Written Works, which means that I cited and / or quoted works and opinions of other authors in this written final work of studies in accordance with the FELU's Technical Guidelines for Written Works;
4. to be aware of the fact that plagiarism (in written or graphical form) is a criminal offence and can be prosecuted in accordance with the Criminal Code of the Republic of Slovenia;
5. to be aware of the consequences a proven plagiarism charge based on the this written final work could have for my status at the FELU in accordance with the relevant FELU Rules;
6. to have obtained all the necessary permits to use the data and works of other authors which are (in written or graphical form) referred to in this written final work of studies and to have clearly marked them;
7. to have acted in accordance with ethical principles during the preparation of this written final work of studies and to have, where necessary, obtained permission of the Ethics Committee;
8. my consent to use the electronic form of this written final work of studies for the detection of content similarity with other written works, using similarity detection software that is connected with the FELU Study Information System;
9. to transfer to the University of Ljubljana free of charge, non-exclusively, geographically and time-wise unlimited the right of saving this written final work of studies in the electronic form, the right of its reproduction, as well as the right of making this written final work of studies available to the public on the World Wide Web via the Repository of the University of Ljubljana;
10. my consent to publication of my personal data that are included in this written final work of studies and in this declaration, when this written final work of studies is published.

Ljubljana, November 13<sup>th</sup>, 2017

Author's signature: 

## REFERENCE LIST

<b>INTRODUCTION .....</b>	<b>1</b>
<b>1 SOFTWARE DEVELOPMENT .....</b>	<b>3</b>
1.1 Software development methods .....	5
1.2 Agile Software Development.....	7
1.2.1 Overview of Agile Methods .....	8
1.3 Scrum Method.....	10
1.4 Traditional vs Agile .....	14
1.5 When Adopting Scrum .....	16
<b>2 SPARC SOFTWARE AT RENAULT SAS .....</b>	<b>17</b>
2.1 After Sales Department.....	17
2.2 SPARC software.....	19
2.2.1 Old SPARC.....	23
2.2.2 SPARC Evolution .....	24
2.3 SPARC Vision.....	28
2.4 SPARC Resources .....	28
2.5 SPARC Planning .....	29
<b>3 APPLYING SCRUM METHOD TO SPARC AT RENAULT SAS .....</b>	<b>30</b>
3.1 Kados tool for Scrum/Kanban development.....	30
3.2 SPARC Scrum Team .....	33
3.2.1 Team Roles .....	33
3.2.2 Communication.....	35
3.3 SPARC Evolution Frequency .....	37
3.4 Process of Defining User Stories.....	38
3.4.1 Setting Priority in Development Process.....	39
3.5 Product Backlog .....	40
3.5.1 Documentation.....	41
3.5.2 Features .....	42
3.5.3 Bugs.....	44
3.6.1 Sprints Backlog.....	46
3.6.2 Sprint Velocity of a Scrum Team .....	48
3.7 Testing and Validation of User Stories.....	49
3.8 Release Burndown.....	51
<b>CONCLUSION .....</b>	<b>53</b>
<b>POVZETEK V SLOVENSKEM JEZIKU .....</b>	<b>55</b>

<b>REFERENCE LIST .....</b>	<b>57</b>
-----------------------------	-----------

## **LIST OF TABLES**

Table 1 : Release for Previous 12 months .....	38
Table 2: Priority List Example.....	40
Table 3 : Sprint Backlog US499 .....	47
Table 4 : SPARC Team Velocity Table .....	49
Table 5: Realease Burndown Data.....	51

## **LIST OF FIGURES**

Figure 1 Extreme Programming.....	8
Figure 2 Feature Driven Development.....	10
Figure 3 Scrum Development Method .....	14
Figure 4 SPARC Functionalities.....	21
Figure 5 Information System SPARC .....	23
Figure 6 Old SPARC interface.....	24
Figure 7 Welcome Screen SPARC 7.....	25
Figure 8 SPARC 7 myRef Screen.....	27
Figure 9 Kados Welcome Screen.....	31
Figure 10 Sprints Roadmap Board.....	32
Figure 11 User Story .....	33
Figure 12 WorkPost Management.....	43
Figure 13 Count Of Familles .....	44
Figure 14 Design Bug.....	45
Figure 15 Velocity Chart for SPARC Team.....	49
Figure 16 Release Burndown Chart .....	52

## INTRODUCTION

Software development is a fast-moving process, which is in constant evolution. Therefore, methods that we are using in software development can vary significantly among companies. Usually it depends a lot on the company's culture. For instance, some companies are more careful and adopting new approaches can take longer while others like to be early adopters and are always looking for new methods. However, some methods are very specific so not all companies can adopt them.

In my master's thesis, I will be focusing on the Scrum software development method and will be illustrating a many of the practical problems Renault faces. It has been only 2 years since we started applying the Scrum method in our development process. Therefore, we can say that we are only rookies and we are still making a lot of errors throughout the process.

Communication type is very important when you are using Scrum. The problem at Renault is that we are still used to having to document everything in a very detailed manner like we used to do in the past, like in the waterfall method, in which you have to document every phase and every step. Sadly, we have kept quite a lot of the previous methods although we are trying to reduce them as much as possible. Another problem that comes with communication is the size of the team and the huge hierarchical chains that accompanies it. It is a long way for new user stories to come to developing a team. Besides documentation problems and hierarchical chains, we have problems communicating with the developer team, which is located in India. We have adjusted their working schedule in order to overlap with us as much as possible, but still on a daily basis we overlap only about 3 hours (Resnick, Bjork & Maza, 2010, p. 22).

The problem we face in every new release is that we do not manage our product backlog very well and eventually we do not finish all tasks in the planned time. Therefore, we have to move user stories to the next backlog that is planned for the next release. This way our entire yearly plan can become delayed since we are pushing more and more user stories to the next backlog. Eventually for the last backlog at the end of the year it will be overwhelmingly difficult to handle everything in time.

Team organization is rather different in Scrum than in older traditional software development methods. In traditional methods roles are quite specific such as developer, business analyst testers and engineers while in Scrum methods roles have to be set with wider responsibilities. When I look at our case at Renault, we are taking good steps towards this goal especially with new people who are joining the team. New people that join the project have a lot of responsibility in the project, however people who have been there for

more than 5 years are being designated to a specific post with very narrow responsibilities (Resnick et al., 2010, p. 25).

Another important aspect of our process is that we finally develop something that users need. This is the main point of our development. We do not develop software for ourselves but for final users, and we have to keep this in mind at all times. After that, we have to accurately prioritize the user stories based on whether they are urgent or are less needed. The current big problem is that we do not prioritize the product backlog items correctly. The priority list depends only on whether the item was created in our Scrum software. So basically, developers follow the timeline of user stories that were created. This is very wrong since we should definitely set certain priorities on our product backlog items in order to bring them to “life” safely.

The main purpose of my master’s thesis is to improve the entire Scrum process at Renault SAS (fr. *Société par actions simplifiée*) using literature and practical experimentation. Our SPARC team is still considered a recent adopter of the Scrum method so there are many things to change in order to reach a clean “flow” of each release. My role at Renault SAS is MOA (fr. *le maître d’ouvrage*) which could be translated as Business Analyst. In our current structure, I work on a daily basis with the SPARC application owner and Scrum master. Therefore, my position allows me to have an entire view of the SPARC evolution, from where I can spot difficulties in the process and search for solutions.

The goal of my master’s thesis is to identify the main problems of the current Scrum method that we are using for the development of SPARC at Renault. I will analyze the problems that were identified and solve them in order to improve the process. The improvements that I will suggest to the team will bring more stability to the project and increase the performance of the team members. Another goal is to make the transition from traditional methods to Scrum more detailed. This means that if we adopt Scrum we should adopt all aspects of Scrum, not only some of them while keeping other characteristics of the previous development method.

Finally, the main goal of my master’s thesis is that at the end we deliver better software to the end users more quickly. What I mean by having better software is a program of higher quality, and software quality from a user’s point of view involves, according to Chappell (2017, pp. 3-4):

- Meeting specific requirements,
- Creating software with few defects,
- Good performance,
- Ease of learning and ease of use.



My master's thesis will consist of a theoretical part and a practical part. Both of these parts will be analyzed critically and will be supported with examples from real cases. The theoretical part will be based on software development in general, for which I will collect literature from different authors. I will find the main differences between traditional development methods and agile development methods and identify when the application of Scrum is needed and when it is not.

The next part will be more practical. I will be comparing the current procedure that we use at Renault with the ones described in the literature. I will be looking for the differences between the two procedures and I will offer solutions for how we can implement these changes. I will use different metrics in my analysis, such as sprints velocity, release burndown and sprint burndown in order to get a better view of the progress that we make in Scrum methods.

My master's thesis will consist of three major chapters. The first chapter will consist of current software development methods. In the second section of the first chapter, agile software development will be presented and a more detailed presentation will be given on the Scrum method. In the next section there will be a comparison between traditional methods and agile methods where the main differences will be pointed out. The final section of the first chapter will be an analysis of Scrum application in practice, meaning when the Scrum method should be applied and when we should use another method. The second chapter will cover the software SPARC and outline the departments at Renault in which we use this software. First there will be a detailed description of a department where SPARC is used and afterwards there will be a more detailed description of the SPARC program itself. It is very important that we know the background in order to gain a better view of the Scrum method. In the last chapter, the Scrum method will be presented in a practical case which is used for the development of SPARC. At the same time problems will be indicated and solutions will be provided in order to improve the Scrum method in this particular case.

## **1 SOFTWARE DEVELOPMENT**

In my master's thesis I will be talking a lot about software development; therefore let us first clear up what software development is. Like in most cases there are multiple definitions. Each person can interpret software development in a different way and can also therefore apply it differently. However there are common points in all of the definitions.

According to Alexis (2017) software development is a process in which we have actions required for efficiently transforming a user's needs into an effective software solution. Efficiency means doing things in the right way, and effectiveness is doing the right things.

So, for the software development process to be successful, it is not enough to do the right things. We also have to pay attention to doing them in the right way.

In this process, usually we use multiple tools and methods in order to attain the highest level of efficiency and effectiveness. Methods and tools are chosen according to the nature of our project. This means that we have to find the best method and tools that correspond to our project which will ensure that project will bring expected results when it is finished. However, all software products have the same life cycle no matter which tools or methods we are using. There are two keystones in the software development life cycle:

- The starting point where the user has a need and we create a new product concept,
- The ending point where the product becomes outdated.

The ending point can also be defined differently since certain software can be outdated but is still usable. Therefore, maybe it is better to say that the ending point is when software is no longer available for use. During the life cycle of the software we go through various phases, and these phases are also present in most projects no matter which method we are choosing as our development method (Alexis, 2015, p. 16):

- Concept phase (users' needs are described),
- Requirements phase (who are the users, what data is needed, etc...),
- Design phase (software design system architecture, hardware and software required),
- Implementation phase (coding),
- Test phase (testing the code),
- Installation phase (deployment or making software available to users),
- Maintenance phase (solving problems that might occur).

Certain phases listed above might not be present and activities in each phase can also be missing if we are facing smaller projects. It really depends on the nature of a project. For instance, smaller projects might not have the requirements phase which can be left out or done in a very informal way.

However, we can have two projects which are the same nature and we would expect that the steps in the software development process are the same, but this is not necessarily true. It also depends on which method was chosen as a development method. The main difference between the development methods is how they apply these phases, such as in which order each of the phases is applied and how we manage each of the phases. Each of the phases has several sub-phases with multiple actions in each sub-phase, and these actions differ from method to method.

## 1.1 Software development methods

In the software development process, we have to choose a development method which we will follow for each project. Software development methods are frameworks used to structure the plan and control the process of developing an application or software. In each of the methods we have organized tasks that have to be performed or followed, and these tasks and the way we follow them is what makes each method unique. Since by nature all projects are different, there always has to be certain differences between them. Therefore development methods used in each project are changing. Here are the most used software development methods according to my personal experience:

- Agile development method,
- Waterfall development method,
- Lean Software Development,
- Spiral Model,
- Crystal Methods Methodology,
- PRINCE2,
- Unified Process Family.

At this point I would like to add that in practice no projects can fully follow the development method “by the book”. This means that if we have a project where we apply the agile development method it will never be fully implemented; there will very often be some activities that come from the Waterfall development method. In most teams that switched to the agile development method, their predecessor was waterfall. This means that they do not want to change some methods that worked well from the past and kept the same principles from the waterfall method while adopting some others from agile, which is also the case for other methods. Thus, there are always things that are overlapping with other methods.

The agile development method has multiple methods. Some of them are Scrum, KANBAN and dynamic system development method. Agile methods are very suitable for software development and are used worldwide by companies. Agile gained recognition in 2001 after the publication of the first book which described these methods which were already in use by the teams. The main ideas in the agile principle are real time communication, preferably face to face and not documentation, and having the whole team work on the project together, including business analysts, product managers who define the product, actual costumers and developers. Moreover, agile methods attempt to minimize the risk of the project by splitting the project into smaller “mini projects” which are called iterations. Each iteration consists of all the actions necessary to deliver small fractions of functionality and they usually last from 3 to 4 weeks (Software Development methodologies, 2017; Jongerius, 2014, pp. 13 - 14).

The Waterfall development method is also known as the traditional development method. The method had its first formal definition in the year 1970 and has not changed much since then. The method keeps the software development phases in a linear sequence where they overlap for a very small amount of time.

Lean software development is a very interesting approach where we do not focus on quality and bringing the project to its final stage. The main idea is to deliver a product with minimal requirements and features in order for the product to work. The main advantage of this method is that it can be fast and finished with minimal costs (Young, 2013).

The spiral model is a method which gives a lot of attention to potential risks during our project. It consists of 4 phases (Renaud, 2015):

- Planning,
- Risk analysis,
- Engineering phase,
- Evaluation phase.

Risk analysis is a phase in itself, which explains why it has more importance than other methods. In this phase, we have to identify the potential risks and solutions if failure occurs. This method is very useful for bigger projects, since identifying the risks and finding solutions before they happen can be very cost efficient. On the other hand, it does not have a significant cost difference if our project is small (Renaud, 2015).

The crystal method is based around the strengths and weaknesses of our team. There are 2 types of crystal method: one is light weight which is used for projects that are short and small. The heavy weight method is used for bigger projects and those which are very important where we cannot afford to have bugs. They all tend to have frequent deliveries, face to face communication and reflective improvement (Young, 2013).

The PRINCE2 development method is a worldwide method which was developed in 1989, and was later reworked in 1996. PRINCE means Project In Controlled Environment. Two main principles of PRINCE are:

- Each project needs to have a start date and end date.
- Each project needs to be managed in order to be successful.

In addition, it needs to be clear what we want to achieve with the new product, how we will achieve it, and the responsibilities in the project. The main benefits of using PRINCE are:

- Controlled start, middle and end of the project,
- Regular reviews of the progress,
- Better management since the project is divided into small stages,
- Regular progress reports.

Generally speaking, PRINCE2 is a very particular method which can be tailored for any type of project; therefore it is also used in all types of companies (JISC, 2015).

The unified process family software development method is a unique method which is tailored to bureaucratic development teams. There is a high importance on choosing the best architecture for the project. The labor is used in a way where everyone participates in multiple roles such as development implementation and testing. The approach is also unique because the project will start with the riskiest tasks in order to see if it is doable. If it is not worth going forward, the project can be abandoned in time without additional costs. We have multiple branches of this method developed by different companies according to Young (2013):

- Rational unified process (created by IBM) is an adoptable process which can be tailored for specific organizations.
- Open unified process is an open source framework developed by Eclipse and is much simpler to use. Most of the elements have been merged or removed.
- Agile unified process keeps the same principles as the unified process family but tries to add certain agile principles.
- Enterprise unified process is made for very big projects which will last a long time.

## **1.2 Agile Software Development**

Agile software development is the most frequently used method today in companies. These methods started in the 1990s when companies saw a need to reduce the amount of failing projects. All the agile methods are based on customer collaboration, individuals and interactions, working software comes before comprehensive documentation and responsiveness to change. Being responsive to change is the biggest value that comes with agile methods since today we have a fast-moving economy. Having a plan that can constantly adapt to changes is fundamental in having successful software. The documentation and processes of agile methods tend to be just-enough, right sized and just in time. Agile methods were really created just to bring our project to the end, when we are delivering smaller pieces of working software without finishing the project we are contributing to the stability of the project and ensuring that project will be driven to the end. Within the team members the most important values are teamwork, interpersonal trust and communication quality. Communication is preferred to be face to face, or if it is not possible then at least by verbal

communication over documentation. Other attributes that come with agile methods are talent, skill, experience and programming ability. When we add properties to the individual we get someone who is extremely talented. This is not so easy to find all the time, but even if some of these properties are missing agile methods can still be applied (Rico & Sayani & Sone, 2009, p. 5).

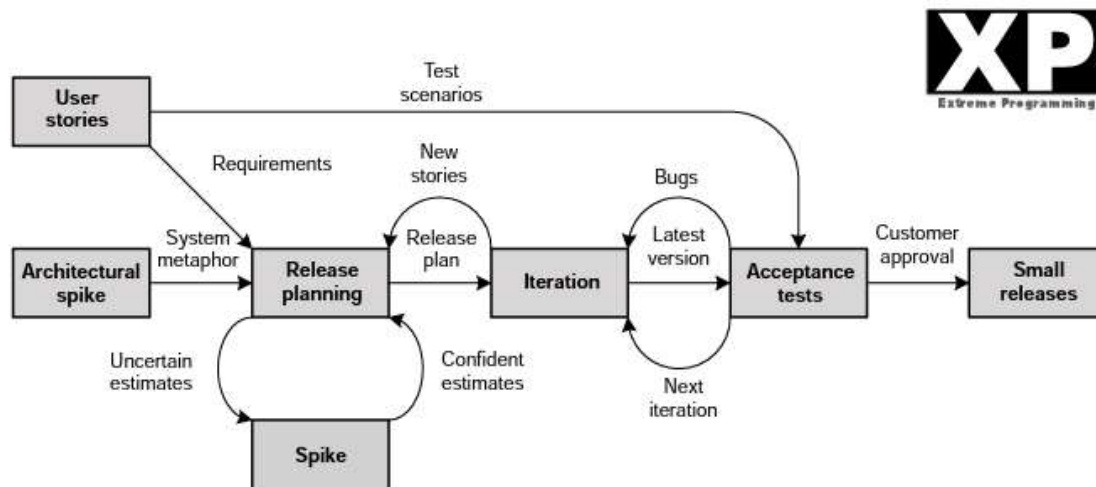
### 1.2.1 Overview of Agile Methods

The most used agile method is Scrum, which I will describe in the next section. At this point I will provide an overview of the other agile methods that are still in use and are applied by some companies. Here is the list of agile methods that are also very popular today besides Scrum:

- Extreme Programming,
- Dynamic System Development,
- Feature Driven Development.

Today, extreme programming is the second most used agile method just after Scrum.

*Figure 1. Extreme Programming*



Source: D.F. Rico et al., *The Business Value of Agile Software Methods*, 2009, p. 28.

Figure 1 shows us the first version of Extreme Programming. It consists of multiple steps and the first input that we have in the model is user stories. In user stories the requirements and the final result are explained. At the same time, there are ready test scenarios that our requirements have to pass in order to be validated. Sometimes user stories are created in a

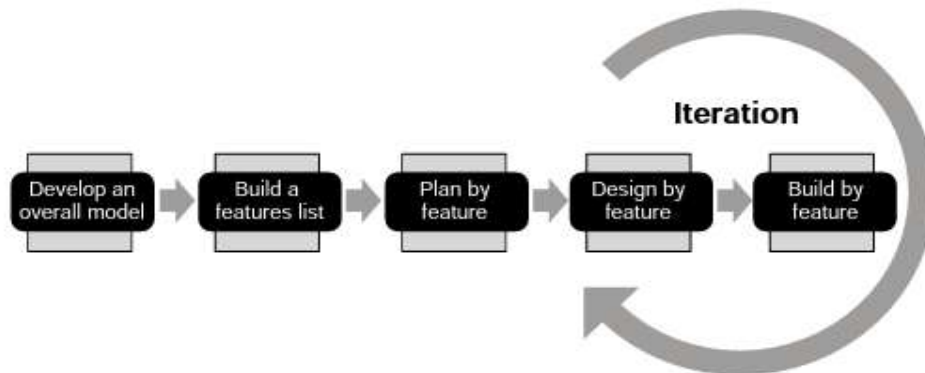
way where they cannot be estimated before the actual development begins. Therefore, we have an additional step in this model called spike which is a unique element amongst the agile development methods. The spike step means that developers have additional time to think of certain problems that have not been spotted before the actual development begins. During the spike, they have to find the most appropriate solution for a technical problem or a new requirement design. After the development, we have correction of bugs and testing with test scenarios that were already done. If everything seems in order the feature is accepted by the customer and we can release the feature. The initial version of extreme programming has changed over the years but the idea has remained the same.

The main principle of extreme programming is that we have short development cycles, which are sometimes considered too short. The shortest cycle for extreme programming is 1 week and it can go up to 3 weeks. In this way it is easy to track progress and adjust resources if needed.

Dynamic Systems Developments were created in the early 90s. Again, the goal was to create a new method that would take into consideration constant changes in customer requirements. Dynamic Systems Developments consist of three main factors: Communication between developers and users, stable and highly skilled developers, and flexible customer requirements. These factors must be fulfilled in order to run our project successfully in Dynamic System Developments Method. After we have the main factors, we can move on to the stages. There are five main stages in the method, which are: Feasibility, business study, functional model iteration, system design and build iteration, and implementation. The process is also divided into iterations, and with focus on communication and team work it proves that this method is also based on agile development values (Rico et al., 2009, pp. 27-30).

Another agile development method is Feature Driven Development, which was developed in Singapore when a certain bank had difficulties bringing a project to life. The project was too big for their current method and requirements were constantly changing, so they had to change their current method. They came up with a method that is easy to use by developers and users and is also very flexible and adaptable to constant changes (Rico et al., 2009, pp. 30-31).

Figure 2. Feature Driven Development



Source: D.F. Rico et al., *The Business Value of Agile Software Methods*, 2009, p. 30.

In Figure 2 we can see the 5 main phases of the Feature Driven Development method. Phase one is where we create a general model in which we define the scope of our project. In phase two we have to create a full list of features that we wish to have in our software. Each feature should not be longer than 2 weeks, otherwise they have to be cut down into small parts. Phase three is done feature by feature and is where we define who will be working on what. Phases 4 and 5 are cycle phases split into 2 to 3 weeks. A list of features being developed in this time depends on the resources available and complexity of the features. In the design phase, we finalize the way the features will work and how they will be developed. In the last phase, the features are being developed and tested. If everything works fine it is delivered to the client.

### 1.3 Scrum Method

In the previous section I spoke about various development methods from traditional to agile and described each method briefly in order to have a basic idea of development methods. I will dedicate this section entirely to the Scrum method and I will describe how Scrum works.

Scrum was formally described in 1995 by Ken Schwaber and Jeff Sutherland. They created this method because the methods that they were using at the time simply did not work. The Scrum method assumes that because software development is unpredictable long term plans will not work; therefore we need to come up with a method which can be flexible or adaptable. Scrum's primary goal is to bring a project to an end or success (Rico et al., 2009, pp. 25-26).



After the formal release, Schwaber and Sutherland wrote multiple books which clarified the entire process of the Scrum development method. As a main source for describing the Scrum process for this thesis, their latest document on Scrum method was used (Schwaber & Sutherland, 2013). There are many other sources on Scrum which are more recent, but using a document that was written by the founders of Scrum with more than 20 years of experience in Scrum is definitely a better choice.

Scrum consists of Scrum teams and their associated roles, events, artifacts (product backlog) and rules. Each of the components in Scrum has a specific purpose and is essential in order to be successful. The purpose of Scrum rules is to bind together events, roles and artifacts to achieve a relationship and interaction between them (Schwaber & Sutherland, 2013 p. 3).

Scrum theory is based on empirical process control, which means that knowledge comes from experience and situations where we make decisions on what is known. There is a special empirical approach in Scrum which optimizes predictability and reduces risks. This approach is based on three pillars:

- **Transparency:** All aspects of the process must be seen by everyone and everyone has to have the same understanding of the definition during the process. This means that if something is marked as “done” everyone in the process needs to have the same definition of “done”.
- **Inspection:** We have to constantly inspect Scrum artifacts and progress towards the goals.
- **Adaptation:** If during the inspection it is detected that certain aspects deviate from the acceptable limits correction must be made as soon as possible to avoid further deviation. In Scrum, inspection and adaptation is done formally with these events:

- Sprint planning,
- Daily Scrum,
- Sprint review,
- Sprint retrospective.

The development team at Renault uses Daily Scrum which is a 15-minute event where inspection of the work is done and adaptations are done if necessary (Schwaber & Sutherland, 2013, p. 4).

The Scrum team size depends on the size of the project but there are always 3 types of roles: The product owner, development team and Scrum master. The most important property is that the Scrum team is self-organized and cross functional, meaning that participants in the team know the best way to accomplish their work without any directives from others outside

the team. Cross functional means that team members have competencies to accomplish work without depending on someone outside the team (Schwaber & Sutherland, 2013, p. 4).

The role of the product owner is to maximize the value of the final product. The product owner is always one person and they are responsible for creating product backlog, and creating a priority list and business values of the items in the product backlog. They make sure that requirements are clear to the development team. Decisions taken by the product owner have to be respected and no one in the development team is allowed to develop something that was not requested by the product owner (Schwaber & Sutherland, 2013, p. 5).

The development team has to be small enough to remain agile and big enough to allow development of all the requirements in the sprints. In the team, there are professionals who do the work of delivering requirements that were set in each sprint. Development teams are self-organized and empowered by organization to manage their own work. No one tells them how to turn requests into releasable functionalities, and they have all the skill required to create each request (Schwaber & Sutherland, 2013, pp. 5-6).

The Scrum master is a key person in the method. They have to ensure that Scrum is understood and implemented. The Scrum master has a good technical knowledge and serves the product owner. Together they review the product log where they estimate the costs and technical difficulty of the requirements. In that way the Scrum master can help product owner to maximize the business value of each sprint. They also help the development team to understand the needs requested by the product owner. In addition, the Scrum master is the one who coaches the development team wherein certain aspects of the Scrum development method are not fully applied yet (Schwaber & Sutherland, 2013, pp. 6-7).

Sprint is the hearth of Scrum, and its duration can vary from 2 to 4 weeks. Once the duration of the sprint is set it cannot be changed. The development process is made of multiple sprints which are run in sequence and the next sprint will start as soon as the previous sprint finishes. Sprint is an event in Scrum which also contains other events such as the Daily Scrum. Each Sprint contains the requirements that were already defined by the product owner and Scrum. According to the complexity of the requirements we create a sprint that is feasible in its set duration. During the sprint duration, we cannot make changes that would endanger the goal of the sprint, changes can be done during the sprint, meaning we can change the scope but it has to be negotiated with the development team. Sprints can be considered small projects since at the end of each sprint we have some additions to the entire project which is ready for delivery (Schwaber & Sutherland, 2013, pp. 7-8).

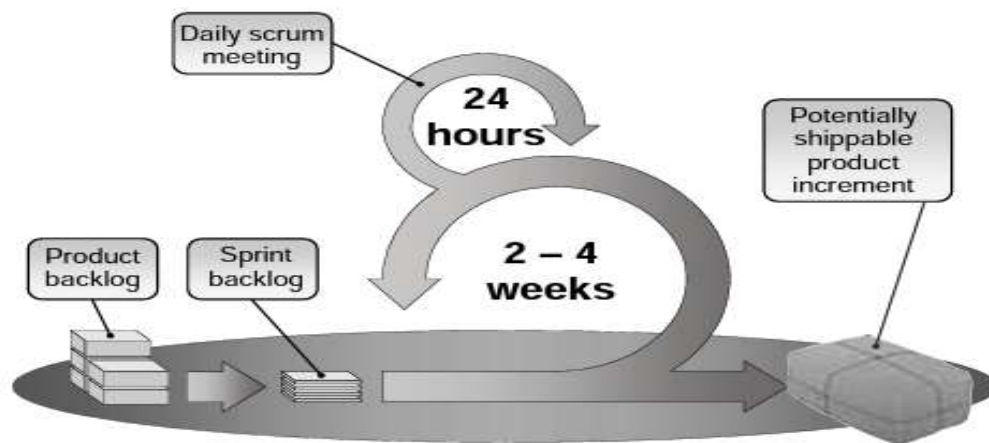
Sprint Planning is an event where a sprint is being created. It is where the entire Scrum team meets and together they select the work load for a sprint. The duration should never exceed 8 hours. The Scrum master ensures that the event takes place and that participants understand its purpose. In Sprint Planning, we prioritize the requirements based on business value and also from a technical point of view the team has to make sure that they will be able to finish the work load set in the next Sprint. The Scrum master with their experience can play an important role in this event to ensure that the workload is executed in time. The development team has to also evaluate the time needed for each of the requirements to help determine optimal duration of Sprints (Schwaber & Sutherland, 2013, pp. 8-9).

After sprint execution we have sprint review and sprint retrospective. These events come after the sprint is finished and the purpose of the two events is to review the sprint to find out how things were done, what went wrong and at the same time find the improvements in the process which will be implemented in the next sprint (Schwaber & Sutherland, 2013, p. 11).

Scrum artifacts provide key information that the Scrum team needs to have to understand the product and activities which potentially have to be done. Artifacts are designed to provide high transparency to the project so that everyone has a good understating of the project. Two main artifacts are the Product Backlog and Sprint Backlog. The Product Backlog is a list of requirements that might be needed in the product. The product owner is responsible for completing and ordering a Product Backlog. The list is never completed and it always evolves as the project is advancing; therefore it is very dynamic. On this list we have all the features, fixes, functions and requirements that we intend to implement in the future to our product. On the other hand, the Sprint Backlog is a shorter list of items that come from the Product Backlog. The Sprint Backlog is a forecast from the development team about what functionalities will be delivered in the next incrementation. The Sprint Backlog is detailed enough that changes can be seen on a daily basis in our Daily Scrum meetings. Only the development team can change the Sprint Backlog during the sprint duration (Schwaber & Sutherland, 2013, pp. 13-14).

Figure 3 shows the exact process I described in previous steps. We have a Product Backlog which was created by the product owner and contains list of features which he wishes to apply to his product. Then together with the Scrum master and development team they create the Sprint Backlog where we have a smaller list of features that came from the Product Backlog. When this list of features is validated by everyone, we create a sprint in which the development team will develop the features listed in the Sprint Backlog. A sprint's duration is usually 2 weeks but can go up to 4 weeks. Each day there is a Daily Scrum meeting to check on the progress of our Sprint Goal. After the completion of a sprint we need to have shippable product with new features included. When it is validated the process is repeated.

Figure 3. Scrum Development Method



Source: D.F. Rico et al., *The Business Value of Agile Software Methods*, 2009, p. 25.

At this point I would like to add one final thought which is very famous among Scrum users. The Scrum method is known to be lightweight and easy to understand but hard to master. You can easily understand the process and how it should work but applying it afterwards and having it work like it should is very hard. How well the process is adopted depends on everyone in the team, but the Scrum master has the biggest role.

## 1.4 Traditional vs Agile

This is the question that many companies still ask themselves: What is the actual difference between traditional and agile methods? In order to answer this question, I will compare traditional methods to the method described in previous section, Scrum.

The main differences between the two methods can be listed in 4 main topics:

- Requirements engineering,
- Stakeholder collaboration,
- Sprints vs phases,
- Team dynamics.

Engineering requirements is a list of requirements which our product is able to do at the end of the project. So basically, it is the features that we want to include in our final product. When we are using waterfall method we need to start the process with gathering

requirements and then gathering production of the approved requirements. This is a long document which serves later for all the phases in the traditional method. Once the document is validated it will not change; it has to stay as it is and should be followed until it is finished. This approach works very well when a project is stable and is unlikely to face any variations in the future. On the other hand, Scrum method allows changes during the development itself. Scrum's Product Backlog gives us the possibility to list our requirements during the entire time and they will be developed later on in sprints. Basically, we spent a lot less time in Scrum on creating requirements, therefore removing them from the Product Backlog will not cost us much. When we are looking at the situation today, the software environment is changing constantly and having a requirement list which is flexible is a huge plus. Taking into the account that some projects can have a duration of 3 years, it is impossible that our needs will not change during that time. The Scrum process is made for the current environment where needs are changing rapidly (Hayes, 2009, p. 19).

Stakeholder collaboration is very important when it comes to project management. Stakeholders like to have a look at the project and know how it is progressing. Scrum has an open relationship which allows stakeholders to see the features which are being developed and how the project is advancing. In addition, at the end of each sprint they get a functional incrementation of a product which allows them to use certain features already. Stakeholders are allowed to add additional requirements during the project and the priorities can be changed, while in waterfall method changes and plans can rarely be changed. Once the requirement's milestone is passed, new requirements have to go through official demand, which in certain projects is not even allowed. This difference allows Scrum to modify requirements and ensures that at the end the delivered product will be what end users would like to have. On the other hand, waterfall method can deliver a product where users changed their needs and the resulting features are not needed anymore (Hayes, 2009, pp. 20-21).

Waterfall method uses a linear approach which is managed in phases. Progressing to the next phase depends on the previous one; thus previous phases have to be in a certain stage which allows us to advance to the next stage. The problems with this method are that verification comes at the end and major failures are identified when it comes to testing and validation in the last phase, which means putting a lot of effort into something that was potentially not done in a proper way. On the other hand, Scrum approach works in sprints which are set off throughout all phases and are made of features that come from the Product Backlog which are then put into the Sprint Backlog. In the Sprint Backlog, we have more manageable tasks which are being developed in sprints. In Scrum, case problems can become visible on the same day since the communication in the team is done on a daily basis through the Daily Scrum and most of the problems are recognized there and solved at the same time. Testing and validation of the features comes much sooner than in waterfall method. Although sprints are short cycles they might not always go as planned. In certain cases, they can be canceled

due to technical or business issues, however the time wasted is significantly lower than in waterfall method. Time wasted in sprints can be a maximum of 30 days, which would be the minimal lost in the waterfall method (Hayes, 2009, pp. 21-22).

Scrum has a different approach when it comes to team members. When using Scrum method our team is small, which means there should be about 20 people in total. The team should be self-organizing and self-managing, which means that everyone in the team knows how to organize their work. In addition, each team member has to be cross functional, which means their position is described in general and each has a more global view of the entire process. This is a main difference between waterfall and Scrum when considering the team members. Waterfall method has members who are focused on their part of work and if someone is missing from the team it will be very hard to replace him. Management of the team is done by the project manager to ensure milestones are achieved. On the other hand the Scrum master does not manage the team; they only ensure they are not interrupted and do Daily Scrum meeting to encourage problem identification and solutions (Hayes, 2009, pp. 22).

## **1.5 When Adopting Scrum**

In the previous sections, different methods and more specifically Scrum method were introduced. The question that remains is when a company should know if the Scrum method is the right choice for its needs. Here I will provide the main characteristics that your project needs to have, where adopting Scrum would make sense.

- If our project requires a lot of reworking and we do not have a clear picture of our needs yet, then using Scrum method would be a great choice.
- If our projects are often exceeding budgets or are late, Scrum method has great monitoring of the current progress of the project.
- If our team members do not understand each other or blame one another, Scrum method emphasizes team work with everyone in the same room. This really means that we have to work together in order to finish the project.
- If there are people in our company who always have something to say although they are not part of the project and they are constantly slowing down things, Scrum is an excellent choice. In Scrum, people outside the project have no right to intervene.
- If it takes a long time for your project to get delivered even though it is not a long project and is technically fairly easy, Scrum will help you remove all unnecessary documentations which will ensure a shorter time to man

These are the main factors that should be present in your project to seriously consider switching to Scrum method. However, for some projects Scrum method is not recommended (Jongerius, 2014, p. 16).

In certain organizations where the work environment is slow, very central and formal, Scrum would not work. Since Scrum tends to be very fast, especially when it comes to decision making, decisions have to be made quickly. Since time is lost by waiting for multiple people responsible in the hierarchy chain to authorize a change, this will make our sprint not fully completed. Informal communication is one of the keys of being successful in Scrum. The main benefit of informal communication is that we make decisions quickly. Since most of the quick decisions are made face to face, this means that we do not have a formal document which will indicate the decision that was made. Therefore, for organizations where formal documentation is required at every step will simply not work with Scrum method (Jongerius, 2014, p. 17).

Scrum is not going to be useful to the projects where we have to put a lot of effort into thinking and realization, since Scrum is a fast method. Thus, highly complex and very long projects would have difficulty working with the Scrum method. If our team is not experienced enough we would have a hard time using the Scrum method since it is all about improvising and making choices, which will be much easier if we already have experience. If a product owner does not have enough decision making power and every suggestion has to go through the back office in order to reach a decision, this would slow down sprints and thus the Scrum method would be slowed down (Jongerius, 2014, p. 17).

## **2 SPARC SOFTWARE AT RENAULT SAS**

### **2.1 After Sales Department**

The after sales department has a big role today in the entire Renault Groupe. The pricing policy of the group is to sell cars at breaking point to set the price of the car where all expenses are covered. Therefore, the profit is coming from aftersales where the prices of spare parts are usually much higher than the breaking point.

The corporate after sales department is responsible for all the subsidiaries of Renault around the world. Currently we have subsidiaries in 33 countries, 34 if we count Australia which is following current procedures to become a new subsidiary of Renault in July 2017. In addition to the subsidiaries we have importers which are not part of Renault but still buy spare parts from us.

The corporate after sales department has multiple smaller units. Each of the units is responsible for certain activities, such as (Renault SAS, 2017b, p. 15):

- Performance unit,

- Accessories unit,
- Pricing unit,
- Marketing unit,
- Quality unit,
- Tires unit,
- Etc.

The entire after sales department has around 200 people at the corporate level; people who work in subsidiaries are not accounted here. Each of the units listed above has around 10-15 people and all of the units are cooperating with each other as much as possible. The unit which I know the best is the pricing unit, as I am part of this unit. This unit is responsible for setting the prices for all of the spare parts that exist in our information system. In the past the entire team was located in Paris but since the takeover of Dacia by Renault some of the activities are being transferred to the Romania team as it is cheaper. Currently the pricing unit is split into two teams: One is in Paris and the other is in Romania. The Paris team contains 9 people and the Romania team is about the same size. The key members of the team are called “Chef de Marché”, which means Market Manager. These people are responsible for certain regions and they do the pricing surveys and apply new prices to the subsidiaries for which they are responsible. Currently our team has 4 Market Managers who are responsible for their own regions (Renault SAS, 2017b, p. 15):

- G9 (Europe),
- Region Eurasia,
- Region (France),
- Region AMI, Asia, America.

The Romania team is responsible for all the importers, so setting new and adjusting the prices of spare parts for importers. In addition, they all provide support to the France team in various tasks.

On average each market manager will apply two tariffs per year, meaning that he will do a complete analysis of different indicators and apply the new price to spare parts for each country two times per year. This process requires a lot of software support since we have 1 million spare parts and each country has different prices. Managing this without software support would be just impossible (Renault SAS, 2017b, p.15).

My role in scrum team is MOA. I am a product owner of a SPARC software program which all the market managers are using on a daily basis in order to create new tariffs and consult about spare parts. The software which I am talking about is being used also by the entire Romania team and corporate after sales department. I am responsible for the evolution of the



software and to reach customer expectations. I will describe SPARC in more detail in the next section.

## **2.2 SPARC software**

The greatest need from the pricing department was to have software support for all their activities. Since there were too many spare parts, countries and economic changes which required frequent tariff changes, all the information that we had just too much for each market manager to handle in excel files. The procedure of creating new tariffs at the time required a lot of time from market managers.

- All the data that they analyzed was stored in excel files.
- For the price evolution study, they had to look in all the previous excel files.
- From the historical files, they created a new excel file with a list of parts still active.
- They simulated new prices by applying simple coefficients which was not always the best way to calculate new prices.
- The new list of prices was sent to the countries where they integrated new prices into their local files.

The procedure that we were using had many negative effects, such as:

- It required a lot of manual work.
- Historical data that was used for analysis was not reliable; therefore the prices were sometimes very off.
- It was very hard to observe the application of new prices that were sent to subsidiaries. In some cases, new prices from the corporate level were not respected on the country level.
- Car parts that did not even exist in a country's warehouse were sometimes tariffed which resulted in confusion and loss of time.
- An opposite case is where we had no prices for certain car parts which were present in our subsidiaries. Therefore, those car parts could not be sold since they had no price.

Because of all these difficulties, market managers had a need to modernize their process. They wanted to have one software program which would be the master of prices of spare parts in Renault. The idea of having such software was identified already in 1997. However, things at Renault do not move fast and everything takes time. Therefore, the first version of such software came out in 1999. It was called SPARC and is widely known today in the Renault network. SPARC stands for Spare Parts Catalogue and it is the second most used internal software at Renault. SPARC in 1999 had around 700 unique users that were connecting on SPARC on a daily basis (Renault SAS, 2017c, p. 18).

The main rule of SPARC when it was created was that all spare parts that exist in Renault IS had to be known and documented in SPARC. No matter the origin of the spare part or how it was created they all have to be in SPARC, otherwise they cannot be commercialized. It is the rule that was built at the time and today we follow the same principle (Renault SAS, 2017c, p. 18).

Whatever the origin of the spare part, it has to be known in SPARC:

- Original Parts references (created by Engineering),
- Central Specific Offer references (created by Engineering),
- Local Specific Offer references (created by Local After-sales Marketing).

At the time SPARC was created we did not have as many unique spare parts as we do today. In the past, it was managing about 350,000 car parts. 200,000 of those spare parts were central, meaning they were created by the corporate office and we had another 150,000 spare parts which were local and created by local marketing teams. But they were all managed in SPARC. When SPARC began we had 28 subsidiaries, which has now grown to 34. We had around 100 importers and 4 main partners. The most important partners for Renault at the time were:

- Nissan,
- General Motors,
- Daimler,
- Renault Truck.

In order to manage the huge number of spare parts, specific segmentation was created. We had 14 segments which are alphabetical:

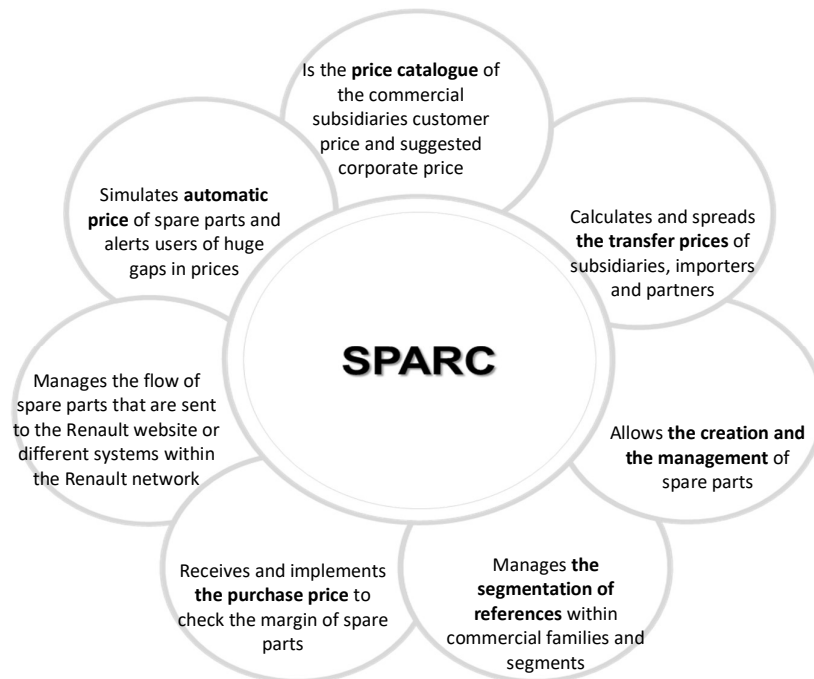
- A (fr. *Accessories*),
- C (fr. *Carrosserie*) or Bodywork,
- E (fr. *Entretien*) or Maintenance,
- P (fr. *Pneus*) or Tires,
- R (fr. *Echange Standard*) or Remanufacturing.

These segments are used in order to have better view of the car parts and their business indicators, such as turnover by segment and price evolution by segment. After segment classification, we have commercial family classification which uses three digits. Each segment has about 100 commercial families which group spare parts together with even more similar characteristics:

- 002C (rear window),
- 007C (shock absorber),
- 010C (handle).

This is the main segmentation that SPARC uses and nowadays it helps market managers to follow evolutions by family and to quickly apply price modification to the entire commercial family which is not selling well. The main functionalities of SPARC:

*Figure 4. SPARC Functionalities*



Source: Renault SAS, *New SPARC Evolution*, 2017a, page 2

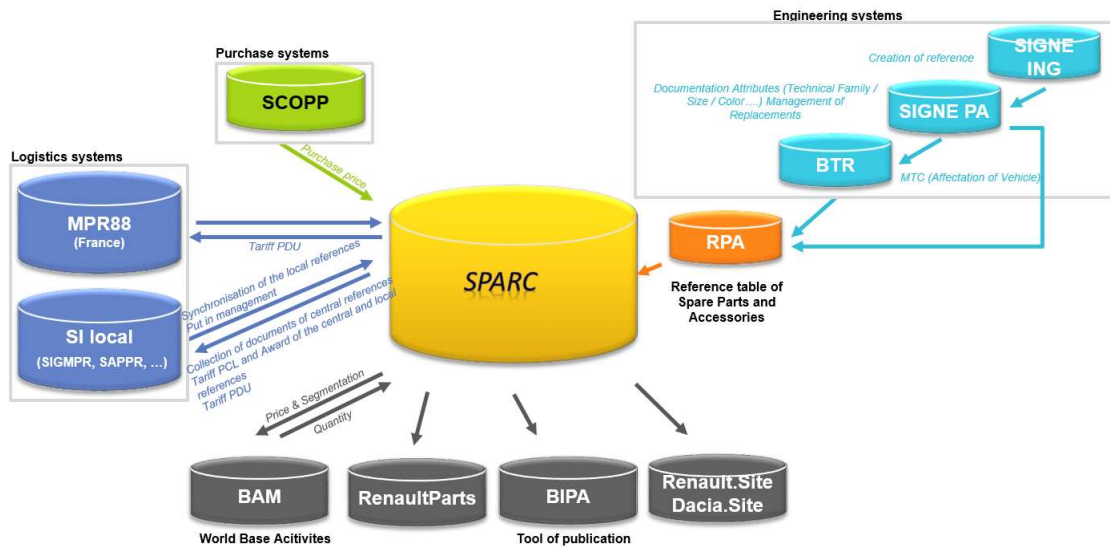
SPARC has a very important role in the information system of Renault. It is software which manages all the prices in one place. In order to clearly see the role that SPARC has in the IS of Renault I have created figure that shows all the possible flows and where the data comes from.

From Figure 5 we can see the complete workflow of a spare part all the way from its creation to the distribution into other information systems. To explain the workflow, we will imagine that engineers have created a new car engine which has to be documented.

- The origin of this engine is Engineering, meaning that a reference has to be created in the Engineering IS first. The reference creation is done in SIGNE ING where it gets assigned a unique 10-digit code which never changes in our information system.
- The reference is sent to the SIGNE PA where it is specified according to technical family (three digit code used for segmentation), size (volume, weight), color and possible references that could replace this engine.
- BTR is another system that defines in which cars this engine can be placed in.
- All information from the previous 3 systems is sent to RPA, which is like a database of all the engineering parts. It is also a filter which verifies that the reference has all the required information in order to be sent to SPARC.
- Every night SPARC receives the data from RPA in a flow delta, meaning that once our engine reaches all the needed requirements it is sent to SPARC. Then it is in SPARC to document this product's different requirements. Requirements such as security file have to be done by the quality team, so every reference needs agreement which will allow our engine to be sold as a spare part.
- Once per week we receive purchasing prices from SCOPP. In the information system of SCOPP they know exactly what is needed by the engineering department for this engine since they have flow from RPA.
- With purchasing prices we know SPARC can set simulated corporate price for this engine. In addition, product managers can start setting subsidiaries prices for this engine.
- When our engine has all the prices, SPARC starts sending the reference to other systems such as MPR88 which manages logistic costs and local systems where we send the final customer prices and suggested corporate prices to BIPA, which is a catalogue of spare parts, and it needs prices for each spare part.
- Import flow is BAM, and SPARC sends prices and references that are available for sales to BAM, and BAM sends us all the references that were sold. This flow allows our product managers to follow turnover and it provides them with real feedback on prices that they are setting.

The entire process that I have described above can take in the best scenario 2 weeks and in the worst scenario it can take up to 6 months.

Figure 5. Information System SPARC



Source: Renault SAS, *New SPARC Evolution*, 2017a, Page 7

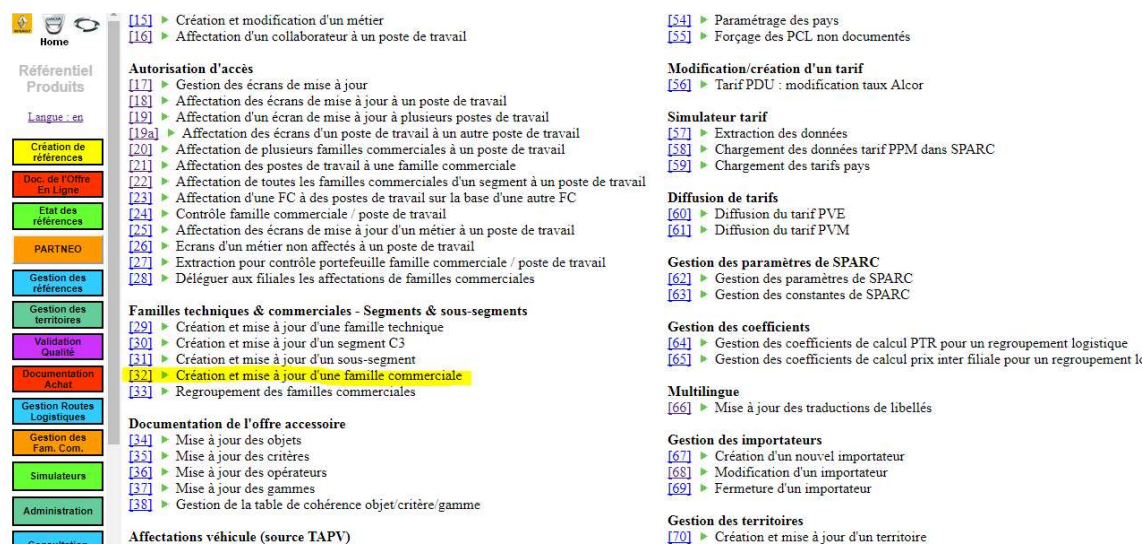
### 2.2.1 Old SPARC

Old SPARC, or nowadays we call it “SPARTAKUS,” was first launched in 1999 and was under constant development until 2015. The technology used for the development of the interface was mainly HTML which was connected to an ORACLE database. The technology and interface was modern at the time, however in the next year it became rusty. The development and evolution over 16 years were always focused only on backend changes. The main goal was to improve data quality in SPARC. Alerts were one of the main things that prevented having the wrong data in SPARC and improving its quality in general.

Figure 6 shows one of the screens in old SPARC. At the start, it was working well since we had only consultation. However, users started wanting to modify the data in the database. Therefore, new screens had to be made, to allow users to modify data directly in old SPARC without asking administrator to force changes into the database directly. With the addition of more and more screens, old SPARC started to become overwhelming and badly organized. Even admins did not know where certain information was shown and the functionalities of screens. There were many things that were developed where we had no business value of the evolution. At the time, the development of old SPARC was primarily led by IT where the business had nobody to represent them when features were under development. As you can see below, in just the administration tab we have around 70 sub menus which then have around 10 sub-sub menus. Some of these menus were only used

once and they were never used after but they were kept on the screen, which just caused confusion among users.

Figure 6. Old SPARC interface



The number of these menus was just getting out of hand. A negative experience that came out of this confusion was where we had a screen already developed but we could not find it; therefore we developed it again. At the end, we had 2 features which basically did the same task, which meant there was a huge loss of time and money.

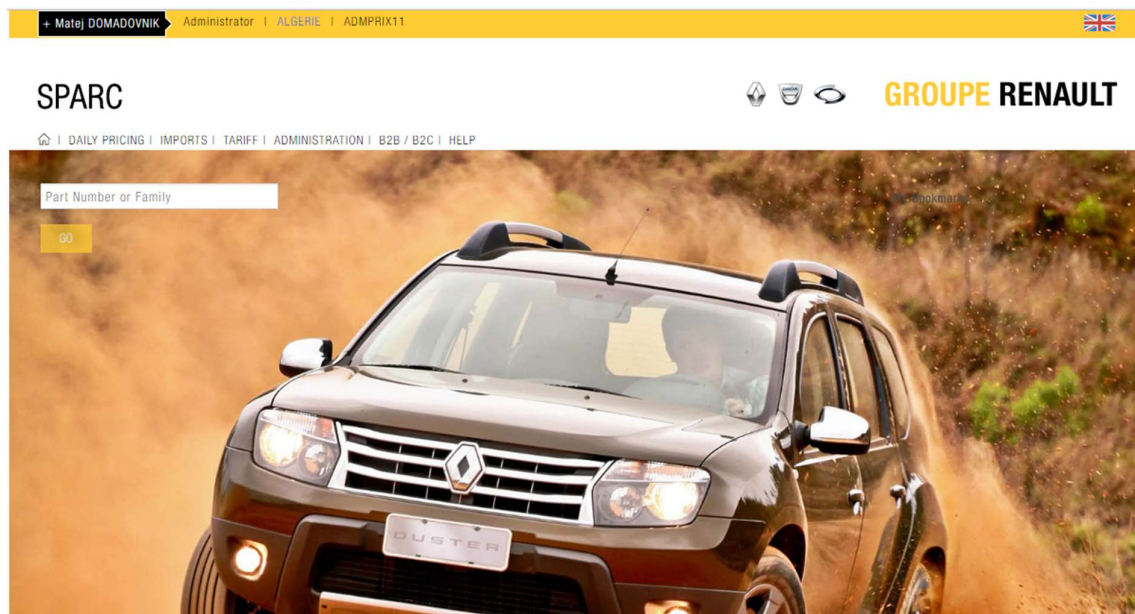
## 2.2.2 SPARC Evolution

After 16 years of using old SPARC there was a need to make a big step forward. When the project was presented there were a lot of doubts that it would succeed. Even the person who suggested the change and who presented the new SPARC change doubted the project. Top management had big reservations as well since there were many large-scale projects that were unsuccessful at Renault. Finally, the resources were granted for the project and SPARC 7 (this is what the project was originally called) was approved. The main evolution of the project is that we will completely replace the old interface with a new interface, which will use modern technologies. In addition to the new interface, a new feature was to be added called the PCL simulator. This is basically the simulator which market managers would use to simulate final customer prices for each country they are in charge of.

After 6 months of development the new SPARC interface was completed, which was created in HTML/CSS/JavaScript with help of AngularJS and jQuery. The database was still an ORACLE database which was connected to the interface with the help of SQL developers

and PL/SQL. In the first evolution there were only a couple of main features from the old SPARC that were involved. Since June 2015, we have been transferring all the other features that were in old SPARC to new SPARC. In addition, we are adding new features in every new update in order to improve data quality and increase revenues created by the after sales department. Just three months after the launch of SPARC we launched a new PCL simulator which did not exist in old SPARC. This simulator allows market managers to create new prices for spare parts and to analyse the price effect through history all in one place. This simulator has helped us reduce the number of market managers needed in the after sales department since now market managers can easily manage more countries at the same time. For example, the entire Europe region can be managed by one market manager. The simulator reduces the time needed to apply new tariffs for each country. Now on average it takes about 2 weeks to apply new tariffs for a country, and the prices that are simulated are much more coherent to the situation in the market.

*Figure 7. Welcome Screen SPARC 7*



The objective set was to completely migrate old SPARC features to new SPARC until December 2016, which would allow us to completely remove access to old SPARC. However, the objective was not reached and now in 2017 we are still migrating certain features to new SPARC. Currently I am responsible for what will be developed and what will not be developed in new SPARC. The features that are left to migrate do not represent great business value to me; therefore I am not including these changes in new SPARC. However, they are still used by some users so they will have to be migrated one day, and we

cannot remove access to old SPARC since the users are still obliged to use certain screens from old SPARC.

The feature that is most used in SPARC is consultation of a reference (car part). Each user consults around 50 different references per day. In Figure 7 we can see the search box in which a 10 – digit code is entered which represents a unique car part. After typing the reference code, you arrive at the most used screen in SPARC, which is the myRef Screen which can be seen in Figure 8. On this screen we see unique data values which are linked to specifics for this car part. This is also the main data that SPARC keeps in its data base.

The main data shown by SPARC is shown in myRef screen. On the example bellow I will list most crucial information that can be seen in SPARC. Screen is divided into 4 categories:

- Details
  - Designation is name of the reference. Our case is (fr. Attelage) Coupling in English. It is a device which connect parts together.
  - Created by corporate. Which means part is in our full control (fr. Créée par).
  - When was it created (fr. Créée le).
  - Where the part is located (fr. Mis En Gestion) our part is in Cergy which is the biggest warehouse of Renault.
  - TMO which tells us how long it takes to create this part. This case 5 hours.
  - Affectation MTC means which car can use this part. Our case is for multiple cars.
  - Security File is not needed for this part (FDS obligatoire).
  - Replacements, which spare part can replace this one.
- Segmentation
  - Segment (C) Bodywork,
  - Commercial family,
  - Marketing Segment,
  - Strategic spare part.
- Tarification Corporate
  - In this category, we have various prices.
    - Brut Price,
    - Net Price,
    - Purchasing Price.
- Tarification Argentine
  - In this category since I am connected as argentine subsidiary I will see the tarification of argentine.
    - Costumer final Brut Price,
    - Costumer Net Price,
    - Transfer Price (internal price of Renault network).



Figure 8. SPARC 7 myRef Screen

+ Matej DOMADOVNIK
Administrateur | ARGENTINE | ADMATEJ
FR

SPARC
GROUPE RENAULT

TARIFICATION QUOTIDIENNE | PAYS | TARIF | IMPORTS | ADMINISTRATION | B2B / B2C

Saisissez une Référence
7711223344
GO

Détails de la référence 7711223344 - ATTELAGE MONOBLOC KANGOO - 424C

Désignations Marketing (Court/Long)  
Testing\_1231@#\$%^&\*  
Testing\_Reference\_Ermöglicht condições32  
Désignation  
**ATTELAGE MONOBLOC KANGOO**  
Créée par  
**CORPORATE**  
Type  
**MARKETING (OSN)**  
Créée le  
17/12/2002 (CPM19)

Mise en gestion par Cergy  
**OUI (11/03/2003)**  
Référence Interdite  
**NON**  
Accord CAEI  
**NON (06/06/2017)**  
FDS obligatoire ?  
**NON**

Marque  
**RENAULT (01)**  
Constructeur  
**CERGY (1)**  
Catégorie  
**OSN**  
Origine  
**PNEU (7)**  
Remplacement  
Remplacée par la référence 7711213066

Références Liées  
1  
TMO  
5.00 heure(s)  
Unité de vente  
1 pièce(s) de 11 cartouche  
Composition  
-  
Affectation MTC  
Plusieurs Affectations

Segmentation

Segment  
**CARROSSERIE (C)**  
Sous-segment  
**SECURITE PASSIVE (C5)**  
Famille Commerciale  
**BENV CDE SERRURE (424C)**

Segment Marketing  
**ATTELAGE FAISCEAUX (911A00)**  
Echelon  
-  
Modèle de Produit  
**ATTELAGES (911A0000)**

Famille Active  
**OUI (01-JAN-40)**  
Chef de Produit Corporate  
**CATHERINE MAILLOT**  
Famille autorisée à votre poste de travail  
**OUI**  
Stratégique pour Corporate  
**NON**

Valeur Perçue  
**NON**  
Mesures Systématiques  
**NON**  
Animation FDV  
**NON**

Tarification Corporate

PPM Brut  
**543.01 EUR (7 677.19 ARS)**  
PPM Net  
**380.11 EUR (5 374.03 ARS)**  
Remise  
**30% (Code 6)**  
Information sur le PPM  
-  
CA PNC Commercial Monde 12 mois glissants  
**140 EUR**

Créé par  
**ADMORG (MyRef Screen)**  
Début  
05/07/2017  
Mode de calcul (Référence)  
**Automatique (A)**  
Mode de calcul (Famille)  
**Manuel (M)**  
PNC Minimum  
-

PAM  
PAM pour PPM  
**45.61 EUR**

PBC  
France  
**45.61 EUR (45.61 EUR)**  
PBC pour PPM  
**45.61 EUR**

Tarification Argentine

PCL  
**0.62 ARS**  
PNC  
**0.40 ARS**  
Remise  
**35% (Code B)**  
Créé par  
**ISDC04 (MyRef Screen)**  
Début  
21/04/2017  
Mode de calcul (Référence)  
**Manuel (M)**  
Taxe  
**21% (Code C1)**

PDUS  
**58.52 108**  
Prix de Transfert créé par  
**F192**  
Début  
12/12/2003  
Prix Concurrents  
-  
CA PNC commercial Pays 12 mois  
**200**  
Quantités commerciales Pays 12 mois  
**24**  
PNC Moyen (CA/GTE)  
-

Sourcing  
**FRANCE**  
PBC  
**0.03 EUR**  
PAM  
**0.04 EUR**  
Marge / PBC  
**25%**  
Marge / PAM  
-

Existe dans le SI du pays  
**OUI**  
Mise en gestion  
**OUI**  
Epuisée  
**NON**  
Commercialisable  
**OUI (TECOP)**  
Stratégique  
**NON**  
Chef de Produit Pays  
Plusieurs Affectations  
Top Diffusion  
MODIFIER LE PRIX

Liens

Historique des prix  
Prix des autres pays  
Affectation (MTC)  
Remplacements

Envoi vers AOC

Prix d'achat

All the data shown in figure 8 are needed in order to analyse the problems that might occur for certain spare parts. On the same screen, we can see also the price history of each spare part, why it was changed and by who and when. The information that is shown in myRef has the biggest value in SPARC, therefore developments occur on this screen very often.

## **2.3 SPARC Vision**

SPARC's vision from 2015 was that we would transfer all the content from old SPARC to new SPARC and add new features which would increase the total turnover of the after sales department at Renault. The complete closure of old SPARC was planned to be at the end of 2017, however this will not be the case. The latest analysis shows that the closure of old SPARC will have to be done in 2018 since there are still many things left to be transferred. In particular, the administration tab is where we have the most features which are not available in new SPARC. However, the majority of actions which are crucial to business are all available in new SPARC, which means that our users are using new SPARC for their day to day activities.

In section 2.1 I have presented the structure of Information Systems from which you can see that there are many different systems that are depending on each other. Each of the systems has its own process, as I described. One of them is for warranty while others are responsible for logistics or prices. The vision that we have for year 2022 is that we will create one major software program which will bind all those systems together into one, which will result in better data quality. This is a project with a big budget which has already started in 2017; therefore part of our resources is being sacrificed for the sake of this project. Completing it would mean a great change for the entire after sales department team.

At the end of year 2017 everyone who works at Renault should get their own tablet and also laptops with touch screens. Currently SPARC runs only in browsers and it is not responsive; therefore accessing it from a tablet or smartphone is completely unusable. The future of SPARC is that it is to be made friendly for smaller devices such as tablets and also launch a mobile application which would contain some of the main SPARC features.

## **2.4 SPARC Resources**

The budget for SPARC is created for each year according to the demands that the application owner has planned for the next year. Each bigger evolution is valued according to the difficulty of the evolution and time consumption. This is done by the application owner and application leader. At the end the budget has to be validated by top management. The budget which is directed to SPARC is reducing since the biggest evolutions were already done and new SPARC is already operational; therefore top management believes that added value is low on the money invested. However by comparing the budget for year 2018 with the budget from 2017 it had slightly increased.

The budget granted for each year depends a lot on in the development team. The yearly budget will determine number of developers that are attributed to SPARC team. Currently we have 15 developers that are assigned to SPARC and are all located in India, and they are also part of the Renault – Nissan Group. The team in France is also the same, but two senior workers were replaced by two juniors which might have an impact on 2017/2018 development.

## **2.5 SPARC Planning**

Planning in SPARC is done on a yearly basis. After the validation of the budget the application owner can start creating a time line for next year with all the evolutions that were decided to be implemented next year.

The first step is to determine with the application leader how many patches we will have next year and how many incriminations we will split all the evolutions into.

The second step is to arrange all the evolutions into relevant topics. The idea here is that we all work on similar topics in the current patch. For example, we work on improvements for the price simulator tool in patch 8.2. In this way we can be more efficient since we do not jump from topic to topic and sometimes you can do multiple evolutions at the same time.

The third step is to arrange evolution priorities inside the current patch. The main criteria on the evolution list are business value and complexity. The evolutions that bring the most business value with the least complexity come first. After that there are evolutions that are more complex but still important. We also always have some filler evolutions which can be recognized on Monday and are already completed on Friday, meaning easy evolutions with quite a big impact on the business side.

The fourth step is to present the plan to the French IT team and explain exactly what has to be done to each of the evolution listed. At the same time we choose an evolution owner. This is the person who will be in direct contact with the development team.

The goal of this schedule is to have a global view on development for next year, but it is never fixed since we are working in an agile environment and some of the evolutions can easily be dropped and others can be added.

### **3 APPLYING SCRUM METHOD TO SPARC AT RENAULT SAS**

We started using Scrum method in the year 2015, which is when we started the development of new SPARC. Since then we had difficulties applying all the Scrum guidelines into our process. However, we are evolving from sprint to sprint which is making our team more and more agile. In this chapter, I will present exactly how we are using Scrum method and the difficulties that we are facing. At the same time, I will come up with solutions for our problems and suggest improvements that we could implement to improve Scrum implementation in our team.

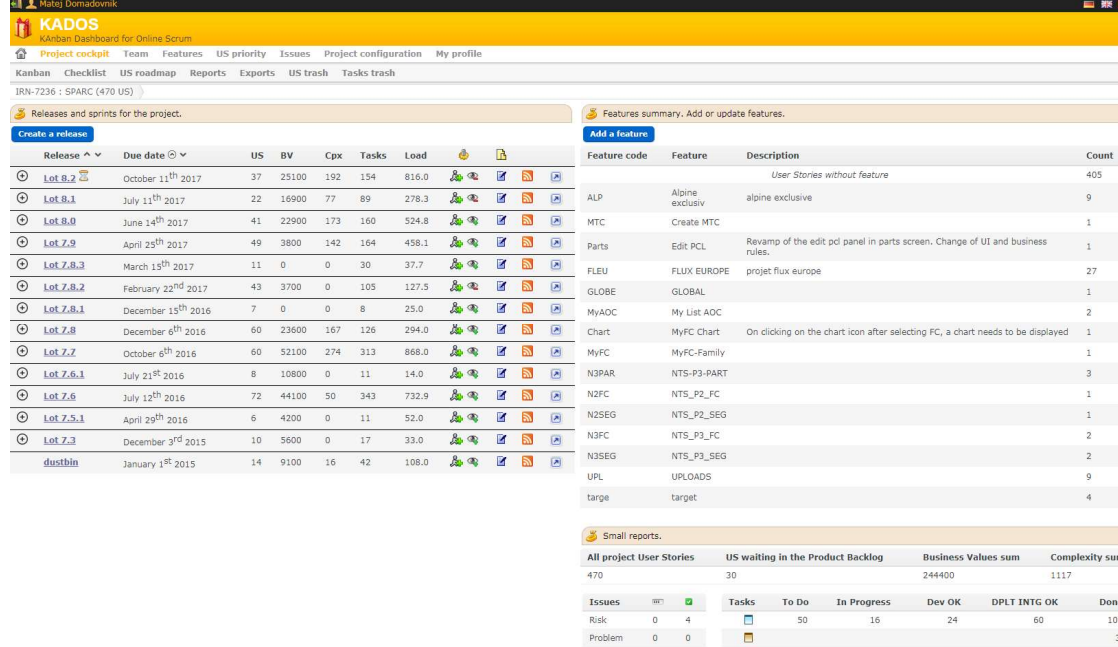
#### **3.1 Kados tool for Scrum/Kanban development**

Our team is considered a Scrum method team; however, we do have a few good features that come from another method called Kanban. The main reason for this is because we are using a tool which is supposed to be a Scrum and Kanban development method tool. We started using this tool in 2015, and it is very well accepted within our team. All the team members use the tool on a daily basis. Some of us checks the progress of certain user stories others report on their development progress.

Figure 9 shows the welcome screen of Kados. This is the screen where we can see a history of the patches that were applied and the version of the release. Since the very beginning, we have launched 13 new versions or increments to new SPARC. Within each version, we can find all the sprints that were done. If we go even deeper, we can see all the US (user stories) that were done in each sprint. The tool keeps the history of all US that we did since the start of our project. Also, it tracks the people who demanded the change who were working on it and what exactly was changed. This can be very useful if we are trying to understand certain features. Since we know the people who were working on it, we can talk with them directly to clarify certain features.

Each version has its release date as well which should never change. According to Scrum method, the sprint's duration is fixed and must never change. For us, this rule was not respected once, in the version where we were implementing new car brand which recently joined Group Renault. The last sprint was extended to have brand integration completed. The extended sprint led to the release date of the version to be delayed. The reason for this is that the development team could not deliver the required changes in time, meaning that workload was badly planned by the Scrum master and leader at the development side. The entire version was roughly delayed for about 3 weeks.

Figure 9. Kados Welcome Screen



The screen that is most used in Kados is shown in Figure 10. This is the screen which appears when we click on the specific release. In our case, I had clicked on release 8.2 which is planned to be delivered in the operational environment on October 10. Once the release is started, the backlog is filled with US which the application owner had provided. Once the Scrum master has created a sprint with the help of the application owner he chooses the US that will be done in the first sprint and sets the duration of the sprint. From Chapter 1 we learned that a sprint's duration should never be longer than 3 weeks and we are definitely respecting this limit. I can confirm that our sprints are always less than 3 weeks.

During the release, the application owner can add new US into the release backlog, but we can never add new US into the sprint which is already running. This last rule is sadly not always respected. Very often there are certain US that will be entered into already running sprints and sometimes some of them will be taken out. This is a huge violation against Scrum method, and it should never be the case.

From Figure 10 we can also see the number of US and their descriptions. Each box in that screen represents a US, which is attributed to one of the sprints that was created or it can still

be in the release backlog. In this same screen, Kados allows us to have different views of the release. We can have a more compact view which will have less information about each US. There is an option as well to sort all US by a business value which allows us to see more important US at the start of each sprint. This screen offers a very visual overview of the entire release and provides enough information for both developers and managers.

Figure 10. Sprints Roadmap Board

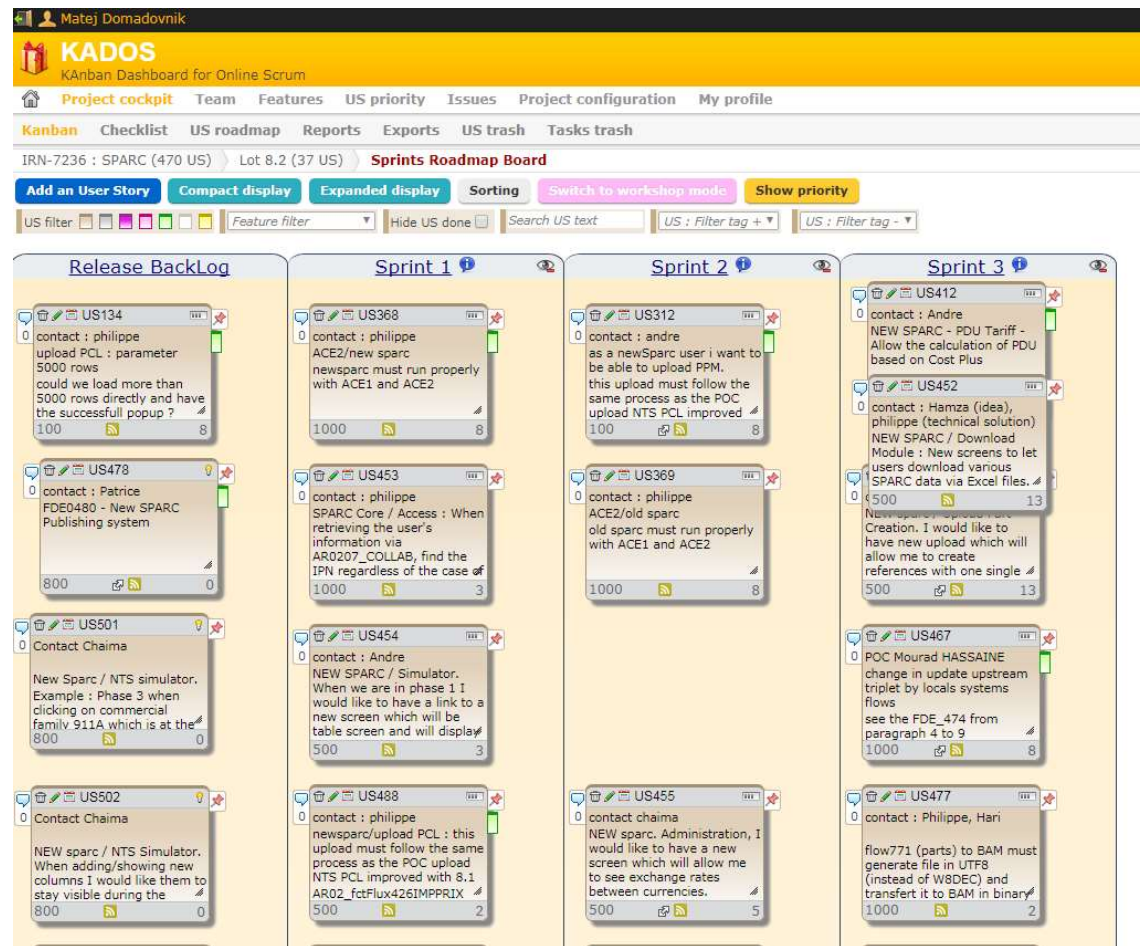
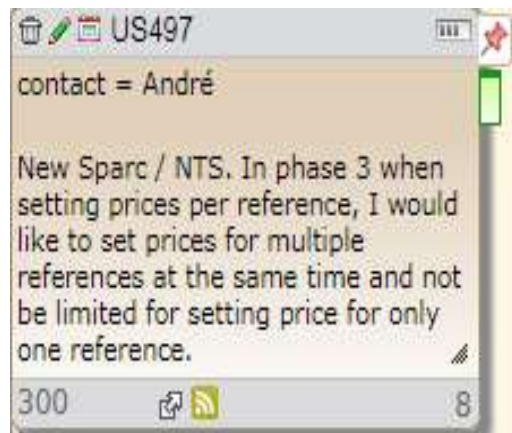


Figure 11 shows us one of the US that the application owner had created. Each US has its IT owner. The IT owner is a person who will have direct contact with the development team. If he is unsure of how to do something he talks to the application owner to make sure that at the end the features delivered will be accepted by the application owner. At the top, we can see the US number, which makes it easier for us to communicate. We always use this unique US number when we are writing an email or when we talk face to face. Inside the box is a short description of the US. This description is really enough for any developer to develop the new features out from the description. The point of the description is to understand the

context. Developers work closely with the IT owners, and the IT owners give developers specific information on what has to be done. But before the IT owner can talk to the developer he must meet with the application owner to understand the need. Together with the application owner, they come up with the solution which is later transmitted to the development team.

Each user story has a progress bar in the right corner as well. This progress bar tells us how far we are from finishing the US. This user story has a green bar on the right side which means that it has been developed, but it is not done yet since it still has to be tested first by the IT owner and then with the application owner. At the bottom left side we can see the business value of the US and on the bottom right side is the development complexity. In the middle, there are buttons to which we can attach certain files that will explain the US more.

*Figure 11. User Story*



## **3.2 SPARC Scrum Team**

### **3.2.1 Team Roles**

According to the Scrum method, there should be 3 roles in the team: product owner, Scrum master, and development team. We follow a very similar structure ourselves with one exception. The Scrum method says that the development team is self-manageable and they do not need a supervisor, which is not the case for us. Our development team has a team leader or manager who leads the team and distributes the tasks among the developers. He also supervises the progress of the development. This characteristic is against the original Scrum method which I described in the first chapter. The consequences of this is that the developers do not think broadly enough; they are very focused on the tasks that are given to

them, and after completing them they move on to the next one. This principle has some similarities with the waterfall method which says just care about your business.

Our Scrum master is the most experienced person on the SPARC team. He has been working on SPARC for about 15 years. Before the new SPARC project, he was working on old SPARC. He has been the Scrum master since 2015. In our team, the Scrum master confirms the releases of each patch. He works closely with the application owner and the development team leader. The Scrum master also ensures that US are progressing towards the release date and that the release will not be delayed. In certain cases, he helps understand how certain features should be implemented. He is the one who keeps the team together and ensures that progress is being made.

In December 2016 I took over the role of being application owner. This role in our team ensures that business demands are recognized and then transmitted to the Scrum master and development team. The application owner must have a list of future evolutions which he wants to implement in the next release. Each of the evolutions is carefully thought through, and they all need to bring added value to the business. He also has contact with the application users and can identify needed evolutions easier. The main responsibilities are:

- Create a list of possible evolutions.
- Analyze the feasibility and development difficulty of such evolutions.
- Analyze the business impact of such evolutions.
- Present evolutions to the Scrum master and IT owners.
- Creation of US.
- Support IT owners in case of doubts.
- In certain cases direct contact with developers.
- Final validation of US.

All of the tasks above are in coherence with the Scrum method except having direct contact with developers for certain US. This means that the application owner has to spend time giving very technical details to the developers to get the job done.

For example, there was an US which would give users a table of syntheses at the end of their price simulation. This US required SQL coding to show users the correct data at the end. The job of the application owner is to tell us which data he wants to have for which country and for which year. However, since there was no IT owner for this US, it was up to me to give developers the exact table names where certain data should be taken from and with which filters. Things like this take a lot of time, and I could not focus on my other activates which were far more important.



The example above is appearing more and more often. The reason is that the IT team in France that works on SPARC is not big enough. Therefore if I want certain evolutions to be implemented, I have to take over the ownership of them.

The development team is located in India, and there are about 15 people who are working on SPARC. One person on the development team is the development team leader he is responsible for the work being done and keeping it on time. There is another part of the IT team which is located in France. The France IT team currently contains 4 people, and they are the so-called IT owners of US, which as we saw is not always the case. The weakness of the IT team is that they do not have wide knowledge in IT. Their knowledge is primarily focused on certain programming languages; therefore we have many developers as well since each one of them is doing a specific US which requires their language. The Scrum mentality is that developers have wide knowledge of IT and can easily replace another developer and can complete their work even if they are not here. This is not the case for us. There are multiple features that are being developed by many different developers. This is because features require the usage of SQL at first and to show the results the developer must use JAVA or JavaScript language. It is very rare that one developer can complete the task alone. This is also the reason why we are changing the members of the development team so often because their skills are not wide enough.

In general, the Scrum master is facing a lot of problems with people leaving the team and new people that are joining. Usually it is young people who join the team, and a lot of training is required, but on the other hand, they learn a lot faster and work much harder than experienced members.

### **3.2.2 Communication**

Communication is another big problem that we are facing in our team. The Scrum method emphasizes the usage of face to face communication over emails and video calls. Usually, solutions happen much faster and are more accurate. Also, you get immediate feedback when you are talking face to face, while with written communication it can take hours/days to get a response. All demands should be done face to face, and if this is not possible, we should use video conversation. The last form of communication that should be used is written communication.

Our Scrum team has 22 people, and there are 5 different nationalities in the team which represents the first problem that we are facing, and this is communication language. Sadly the official language at Renault SAS is still French, making it even more difficult for everyone. The Development Team does not speak French at all, and other team members are not very fluent in French either. French is used between the team members that are working

in France, and when we are talking with the development team, we use English. Therefore almost all requests are to be translated into English for the development team to understand them, which is time-consuming and impacts our productivity. For me, it represents a big problem to explain certain features in French to the IT staff, and it is also time-consuming. From my point of view, the best solution, in this case, is to have English as the official language in our team. For other non-French members it would be much easier to communicate in English all the time, but for the French natives, it might present a problem.

The second problem with communication that we are facing is that our Scrum team is not located in the same place. We are very scattered around which has a big impact on the way we are communicating and on our relationship. We are located at three different sites:

- Renault Technocentre,
- Renault Plessis,
- Renault – Nissan India.

The development team is located in India, meaning that we almost never have physical contact with the team. Two times per year one developer comes and stays with us for 2 weeks. There is also a 4-hour time zone difference which basically means that we can communicate only in the mornings since in the afternoon they are already finished their work. In Scrum communication by Skype is an alternative communication and should not be used all the time, however in our case we are limited to this method. A drawback is that it takes much longer to explain evolutions and it is less accurate. Also, the development team is less likely to ask questions if they must contact someone by Skype then if they are sitting next to them. This has a significant impact on our feature validation process. Sometimes features that are delivered at the end are far from what had been requested.

Another problem but less severe is that I and the Scrum master (and the French IT team) are not located at the same site. The IT team and Scrum master are always located at Renault Plessis which is located 10 km south of Paris. I am located at Renault Technocentre which is located 15 km west of Paris. To reduce the problems that come from communication and to have face to face communication I go to the Renault Plessis site twice per week. When we are together, we do most of the work needed for things to move forward. For the rest of the time we use Skype to communicate, but it would be much better and easier if we were together all the time.

Having English and French as official team languages and not being physically in the same place as a team has an impact on our progress. I will explain the impact with a practical case that has happened multiple times already. I had a new feature which I wanted to develop. The feature is to have a new administration screen that will allow the administrator to grant

the user access. I created a prototype of how the screen should look and what the actions should be. I met the IT team and presented to them face to face how it should look and what the functionality should be. The IT team took notes and tried to memorize as much as possible. After the meeting, the person responsible for the feature creates a detailed text file which contained all the major information that the developers need to know. Once it is done, they sent it to the development team and they also communicated by Skype to make everything clear. When it was finished, they asked me to validate the feature. In this particular case, the feature was developed not exactly as I demanded. Some of the key parts were missing because of bad communication. Because I did not validate the feature, it had to go back to development.

This represents the huge difficulty that we are not located in the same place all the time. We could avoid unneeded documentation, and we could have direct meetings with the entire team if we were located in one place. This would reduce such type of errors where features are not being developed as initially demanded.

### **3.3 SPARC Evolution Frequency**

Since the implementation of the Scrum method and new SPARC project, we have launched 13 releases. Here we include all the major releases and also the minor releases. It is up to the Scrum master to set the next release date. On average we have 5 major releases every year with 2 minor releases. In Table 1 we can see the previous releases for the past 12 months. All the major releases have a duration of about 2 months, and they have from 3 to 4 sprints. After each or sometimes during the last sprint we identify certain improvements or bugs. We do not have enough time to fix the bugs or implement new suggestions without moving the release date. Therefore, we complete the release but after that, we start the minor release which has only one sprint, and the point of this release is to fix bugs in the last release and to apply additional modifications to the integrated features.

The frequency of releases is very important for the application owner. There are certain evolutions that are very important from the business side and make us lose money. Therefore when such evolutions are identified, I want to see them delivered as soon as possible. The maximum waiting time for that evolution to be delivered is about 3 months. Thus, if we are currently in the last sprint in a release a new evolution identified will have to wait for this release to complete. Adding evolutions in the last release might cause a delay of delivery into production which we cannot afford. Therefore, we have to wait for the next release which will be completed within 2 months. This is the delay time in the worst case scenario. Usually, such an evolution is applied to the same release when it is identified. It takes us about 1 month to implement an urgent evolution.

Table 1. Release for Previous 12 months

Start Date	Release	Delivery Date	Number of Sprints	Number of Features	Type
October 6th 2016	7.8	December 6th 2016	4	60	Normal Release
December 7th 2016	7.8.1	December 15th 2016	1	7	Bug fix
December 16th 2016	7.8.2	February 22nd 2017	3	43	Normal Release
February 23rd 2017	7.8.3	March 15th 2017	1	11	Bug Fix
March 16th 2017	7.9	April 25th 2017	3	49	Normal Release
April 26th 2017	8	June 14th 2017	4	41	Normal Release
June 15th 2017	8.1	July 11th 2017	2	22	Normal Release
July 11th 2017	8.2	October 11th 2017	4	37	Normal Release

But sadly in certain cases, reaction time for very simple things can be very long, which is worrying and makes our users dissatisfied. For example, we had an evolution about rounding prices in Switzerland. Renault Switzerland wanted to have certain spare parts rounded to zero decimals. So the filter was to round only spare parts that were bodywork (door, bumper, headlamp, etc.). On April 25<sup>th</sup> that change was delivered but we rounded all spare parts that were not bodywork, so our filter was the opposite. It took about 1 week for Switzerland to notify us that the rounding was not good. We had already started with the next release that had a delivery date of June 14<sup>th</sup>, which is almost 2 months later. We could not change the plan already for next release as we wanted to keep to the schedule, therefore the bug fix was delivered on June 14<sup>th</sup>. In this case, I felt ashamed to explain to Switzerland that they had to wait for another 2 months to have the correct rounding.

### 3.4 Process of Defining User Stories

The application owner identifies user needs, and then they create a US which is a key element in the Scrum development method. The application owner must understand the user needs exactly to come up with a US that will have a positive impact on the application. The evolutions must have a positive impact for users and eventually positive financial results. To ensure that, the application owner must use multiple approaches to identify user needs.

As an application owner of SPARC I use the following approaches for user needs identification:

- Workshops,
- Brainstorming,
- Personal experience.

I organize workshops with a few key application users. Key application users are corporate market managers. There are currently 4 corporate market managers with who I work very closely during the workshop. There are usually 2 workshops per year, and the duration of each workshop is about 2 hours. In this workshop, we all use the application's different features, and we look for improvements and problems. I carefully note down all of these problems and improvements, and at the end, I have a nice list of suggestions that can be implemented in future releases. This list is later filtered by me if I believe that certain evolutions are not worth the time or if we have a better solution for them.

The brainstorming approach is used to get more diverse evolutions. In this case, I have meetings with all the users that work closely with me on the business side, which is around 10 people. These evolutions are not necessarily linked to existing content, or they can be entirely new features that will revolutionize SPARC. After the meeting, all the ideas are filtered based on business value and complexity.

The personal experience approach is also where I get a lot of ideas for new improvements for the application. Since SPARC has a user support team which is responsible for giving access and helping users having difficulty using SPARC, this is the first level of support. When the helpdesk cannot answer the question, they ask me to help them out. I am the only person who knows SPARC applications from the technical side and the business side. Therefore while giving user support, I can see what problems users around the world are experiencing and I can come up with new evolutions that will clarify certain problems in the future. Also certain users are brave enough to suggest evolutions by themselves which is welcome as well.

#### **3.4.1 Setting Priority in Development Process**

After I have collected future evolutions, it is time to set the priority for each of them. This is done with the help of the director of the pricing department. In a meeting, I present the future evolutions of SPARC and I explain the difficulties of each evolution to the director. Then we analyze the business benefits of each of the evolutions listed together. We use a scale from 1 to 10 for complexity, and the same scale is for business value. Total value is calculated by subtracting complexity from business value. The final output looks like Table 2.

Table 2. Priority List Example

Title	Business Value	Complexity	Total	Priority number
Priority References	9	4	5	1
Prix mini verification imports	9	4	5	2
Prix mini	9	4	5	3
Hide Columns	8	4	4	4
Multiple Selection	7	4	3	5
Impprix	5	2	3	6
Highlight Option	6	3	3	7
Additional Panel	10	8	2	8
New Tariff Type "Simulation"	9	8	1	9
Modification phase 1 special families	4	4	0	10
Chrome Tariff Friendly	4	5	-1	11
Timeout function on excel button	1	3	-2	12
Exclude integration of FC Designations	2	7	-5	13

This meeting validates and sets the business priority for all the US that are planned for the next release. According to these priorities, all US in Kados have to be updated and given a business value in this order.

This prioritization is very important since sometimes developers will not be able to develop all the US which were set for the specific release. This action reduces the impact on the business side if evolution is not completed during the release.

This is a very new approach for us; it has been practiced since mid-2017. Before we did not set any priority to the US which leads to cases where certain features were developed that were not so necessary and others which were crucial were left for the next release. The development followed the timeline which depended on when the US was created. From the 8.0 release, we started having meetings on the business side to set priorities during the development process.

### 3.5 Product Backlog

The product backlog is a Scrum artifact which we use more or less correctly as it is described in Scrum theory. We use the product backlog to store all the ideas/evolutions that might be entered into the Sprint backlog one day. Beginning in April I also started adding the business values of items that are in the product backlog; however, the business values are set according to my own beliefs without consulting the business side. Setting the business values helps me sort all the evolutions that we have in the product backlog based on the business impact that evolutions can have. The product backlog helps the entire team see how the

SPARC is going to evolve in the future. By reading the product backlog, everyone can see in which direction SPARC is going. Currently, there are evolutions such as a mobile application or responsive design which indicate to developers that they should start thinking of how we can accomplish availability on tablets or mobile phones in the future of SPARC.

There is only one case which we do not respect when it comes to the product backlog. According to Scrum theory, only the application owner should be the person who adds items to the product backlog. However, this is not the case with us. Often the Scrum master adds certain evolutions which are very technical. These evolutions are of an IT nature and have no direct impact on the business. Such evolutions involve cleaning the database or fixing certain information flows with other systems. In certain cases I am not even consulted when those demands are made, meaning that there are things being developed which I am not aware of, which is very wrong according to Scrum method.

The reason for this is because the team is used to working with the previous method when there was no application owner; there was only the development team which decided what will be developed themselves and business needs were not respected at all. However, the case where the Scrum master creates evolutions himself are very rare, but they still exist.

The solution to this problem is that the Scrum master talks to the application owner and then the application owner has to decide if this suggested evolution has enough business value to enter next release.

### **3.5.1 Documentation**

The agile product backlog in Scrum is a list of features that contains short descriptions of all the desired functions of the product. In Scrum method, to start a project it is not necessary to have upfront documentation which describes all the features and requirements. The application owner should provide information which allows other team members to create a basic picture of the features. A typical Scrum backlog consists of features, bugs, technical work and knowledge acquisition.

In our SPARC team, features are by far the most dominant in the product backlog, being followed by technical work and bugs. Features and bugs that are being entered into the product backlog come from the application owner. Features are prioritized and split into multiple US which is then developed in sprints. Technical work is considered IT demands that do not have a direct impact on the application users. These items do not come from the application owner but the Scrum master. In our case, we do not prioritize these items, since they are automatically considered high priority. Another item in our backlog are bugs that are prioritized and completed according to the prioritization.

Knowledge acquisition, which according to Scrum guides is also part of the product backlog, is not present in our SPARC team. Knowledge acquisition is an item which allows developers to learn about a certain technology/language. For example: "Positive impacts of AngularJS on SPARC." This is an official demand from the application owner / Scrum master who would like to know more about certain technology. However, if we are not using this approach in an official way it does not mean that our development team does not do such research. It is only demanded in a less official way but much less often. Therefore integrating such items into the product backlog would possibly motivate developers to learn more about new technologies since they would know that it might get implemented.

### **3.5.2 Features**

One feature is decomposed in multiple user stories. One story is planned in a sprint, and one feature is planned in a release. The number of stories that each feature is decomposed into varies depending greatly on the size and complexity of the feature. Delivering one US usually should not give value, but when delivering a set of US which form one feature then it represents a value. When planning each release, all the US which complete one feature are attributed to sprints within the release. The estimated effort for each feature is the effort that was given for the completion of the US within the feature. Even though all US together make a feature, we cannot do our testing on the feature itself; we have to test each US separately (Aubry, 2010).

I will present a real example of our latest feature. A new feature in SPARC is a screen where administrators can modify user access to commercial families and to countries. This feature is a heavy weight; therefore I needed to split the feature into multiple US.

Features were split into three main US. Figure 12 shows us the first result of the first US which was to create a screen where administrators can see current user access. Besides username, their workpost, description, and function we can also see the count of commercial families and count of countries that the user has access to. With this first US completed I had one part of the requested feature completed as well. This screen alone without modification possibility is only for consulting, and it does not represent added value to me. The main idea of the feature is to grant or remove access to users. Therefore 2 new US were created which allowed administrators to modify access.



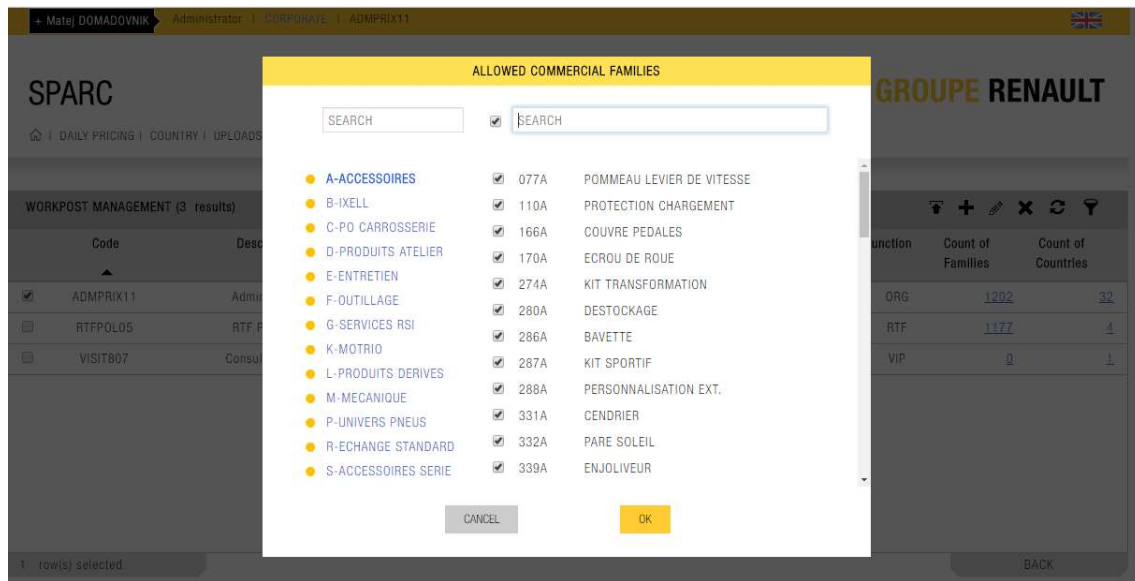
Figure 12. WorkPost Management

Code	Description	Profile	User	IPN	Function	Count of Families	Count of Countries
ADM48	ADM Base de données BIPA	SPARC IT	FRANCOIS DELION	a501350	ADM	<a href="#">1064</a>	<a href="#">0</a>
ADMATEJ	RNPO SLR & Q virtuel L. Bonnot	Administrator	MATEJ DOMADOVNIK	p084658	ADM	<a href="#">1077</a>	<a href="#">54</a>
CPA01	CPA PR Pneumatiques	Corporate Consultation			CPA	<a href="#">41</a>	<a href="#">0</a>
CPD11	CPD DTDP	Corporate Consultation	PHILIPPE MATHIEU	a182289	CPD	<a href="#">0</a>	<a href="#">0</a>
CPMROU04	Formation	Administrator	ELENA NITU	ax04600	RTF	<a href="#">1071</a>	<a href="#">1</a>
CPMROU05	Formation	Administrator	IRINA RISTOIU	ay11932	RTF	<a href="#">1071</a>	<a href="#">1</a>
ISDC01	Admin SPARC	Corporate Consultation	BALDWIN MATHIAS	z000041	ADM	<a href="#">1069</a>	<a href="#">0</a>
PBT93	PBT93	SPARC IT				<a href="#">0</a>	<a href="#">0</a>
PBTTE1	PBTTE1	SPARC IT				<a href="#">0</a>	<a href="#">0</a>

By clicking on the hyperlink in the columns "Count of Families" and "Count of Countries," we get a popup window which allows us to modify the access for the selected user. One US was created for the count of families and another for the count of countries popup. In Figure 13 we can see an example of a screen for a count of families. We separated 2 US since both of the popups are specific, and the same design could not be applied. In the popup, we can see which commercial families the selected user has access to. The actions on this screen are linked directly to the database, and the effects are visible in real time.

This agile approach where we divide features into US is very convenient; for instance, in our case, if we somehow run out of time without completing the last US we would still have a feature which has utility, meaning we could only modify commercial family access without country access modification. There would still be added value, and it also makes it easier for testing and developing where we make smaller steps rather than making the entire feature at the same time.

Figure 13. Count Of Families



### 3.5.3 Bugs

In a perfect world, every time that we deliver our code into production at the end of each release it would work perfectly. This would mean that there are never any bugs in the code after the code has been created. Since we do not have a perfect world and these cases, do not exist we use Scrum method to reduce the impact of bugs, which are still present even in Scrum method. There are multiple methods that can be applied when a bug is discovered. These options depend on the structure of your company, how critical the bug is, and what matters to the application owner the most (Green, 2016).

The cleanest approach in Scrum method to deal with bugs that were discovered during the sprint or release is to put those bugs right back in the release backlog and include them in the next sprint. Since the context is fresh in the developers' minds, the fixing costs will be low. However, if the bugs are found in the production code, we can add those bugs as an US in the next release backlog, and at the same time, the impact of the bug should be estimated to determine priority (Prakash, 2013).

Our SPARC team uses a similar approach, but we have one disadvantage, which normally it is not present in Scrum method. If we have discovered a bug in the production code it can take some time to get it fixed since we are bound to the official release date, and we do not change those dates; therefore bug discovered in the production code might take 2 months to be fixed. It will be fixed sooner in the test environment, but it will get delivered with the next release. However, these cases where we find bugs in the production code are rare

enough and usually do not have a crucial impact. When there is a way to "work around" the bug, then it means it is not crucial. If the bug is discovered in the production code, a new US is created with the same IT owner and the same developer will work on it to be more efficient. In most cases, this US will be applied in the next sprint of the current release.

Most of the bugs are discovered during the sprint phase. Each IT owner during each sprint validates the US that they are responsible for by looking at the code and testing the functionality. This is where we find the first bugs that can later lead to new US or a quick fix without US creation. When an IT owner validates the US, it is the application owner that will do the final testing. When the application owner finds the bug, they will create a new US in the release backlog that will usually be fixed in the same release. The most common bugs are the following:

- Design bugs,
- Coding errors.

Figure 14 shows an example of a design bug, which is currently in the production of SPARC. This bug appears only if the value in the cell exceeds a certain number of characters. In this case, it will overwrite the existing value below. Coding errors that we discover often are wrong data selection from the database. For example, instead of showing the country name such as "Russia" we show the country ID that is in the database such as "27384957".

Figure 14. Design Bug

🏠 | DAILY PRICING | COUNTRY | **TARIFF** | UPLOADS | ADMINISTRATION | B2B / B2C | Extractions

My current PCL tariffs (35 results)							
Search in this table							
Commercial Entity	Tariff Type	PNC Target	Application Date	Chosen Version Code	Consolidated PNC Effect	Progress	Status
<input type="checkbox"/> ROUMANIE (DACIA)	Marketing	2.00 %	04/01/2016				Phase 1 - Initialisation
<input type="checkbox"/> ROUMANIE (DACIA)	Fiat Raise	5.00 %	19/08/2015				Phase 1 - Initialisation
<input type="checkbox"/> ROYAUME-UNI	Marketing	10.00 %	24/04/2017				Phase 1 - Initialisation
<input type="checkbox"/> RUSSIE	Fiat Raise						Phase 1 - Initialisation
<input type="checkbox"/> RUSSIE	Marketing	127.80 %	03/04/2017	RU03	127.54 %	0 %	Phase 1 - Initialisation
<input type="checkbox"/> SLOVAQUIE	Marketing	5.00 %	12/04/2017	SK07	0.16 %	0 %	Phase 1 - Initialisation
<input type="checkbox"/> SUEDE	Marketing	0.00 %	12/07/2017	SE02	0.00 %	0 %	Phase 1 - Initialisation
<input type="checkbox"/> SUISSE	Marketing	0.00 %	01/01/2017	CH01	53.85 %	0 %	Phase 1 - Initialisation
<input type="checkbox"/> TURQUIE (MAIS)	Fiat Raise	3.00 %	27/01/2017				Phase 1 - Initialisation
<input type="checkbox"/> UKRAINE	Fiat Raise	5.00 %	04/05/2016				Phase 1 - Initialisation

0 row(s) selected

BACK

## 3.6 Sprints

Sprints are the core of the Scrum method, and they also present the core activity of the SPARC team. Sprints are actually where work is being completed. In the latest release, we

have 3 sprints. The duration of the first two sprints was 10 days, and the duration of the third sprint was 12 days. In a meeting with the application owner, Scrum master and development leader we carefully select a list of the US which will enter the next sprint. The list of US comes from the release backlog which was created in the previous step. The development leader and Scrum master must ensure that all the US that are in a specific sprint will be finished in the time that was set for the sprint. The application owner tends always to include as many US as possible into each sprint while the Scrum master and development leader set sprint loads more rationally to make sure that sprint load will be completed (Cohn, 2017).

We have never had any problems with sprints, and the duration of sprints is never extended. Sometimes certain US in the sprint were not tested in time because we lack testing team members; therefore the US is validated with a delay of a few days. The experienced Scrum master that we have in our team helps a lot in sprint planning since he can estimate the development duration very well.

Part of each Sprint is also the Daily Scrum which we do not practice in our team. This is a very crucial part of Scrum method. The Daily Scrum helps us see daily progress without connecting to our Scrum tool Kados. By having a Daily Scrum, we can avoid one of the major issues that we face in our sprints. When a developer is working for a few days on a certain US, he communicates with the IT owner at the end, and they realize that this is not how it should be done. Therefore, in our process, we lose a few days which were spent on developing something that was not needed. By having Daily Scrums where each developer briefly talks about his work we can identify these types of problems much faster and avoid further loss of time.

Having Daily Scrums in the SPARC team was already suggested but never tried. The biggest constraint is that the development team is located in India. Therefore the suggestion was to have a daily 15 minute Skype meeting with the entire team but as I said it was never applied; it is just an idea for now.

### **3.6.1 Sprints Backlog**

The sprint backlog is a list of tasks that have to be completed during each sprint. During the sprint planning for each US that will be part of the next sprint, we have to identify tasks that are needed for developers to complete the US. Most of the teams also estimate the time needed to complete each of the tasks (Cohn, 2017).

In the table below we can see the example of the US backlog which is part of the sprint. This is a real example of an US that had entered a freshly created sprint. The table below is seen by the entire team and is very dynamic and updated by the developer himself. US499 has its


status in the first row which can be tasks "running" or "done." Done is when it is validated by the application owner. Just under the US progress, we can see a short description of the US. This description helps the rest of the team understand what this US is about.

The tasks that are shown below are completed by the development team. The first step in every US is to talk with the IT owner about that US. In Table 3 we can see that developing team talked already with an IT owner. This first contact with the IT owner and developer help the developer understand the US more deeply and to clarify any possible questions that they might have. When this task is done, they can then modify the backlog and add additional tasks that they might have identified during the conversation with the IT owner. For this specific US, the developer has 8 tasks, and each of the tasks has a short description such "development – ratio calculation," "code review" etc. For each of the tasks, he also notes the time required to complete the task.

Each of the tasks has multiple possible stages. For instance, for the discussion task, André notes that the task was completed and it took 30 minutes to complete it; therefore it is in the stage of "done." The next task that the developer has to do is an analysis of the existing correlation flow. This means that he has to analyse the current correlation and how and where it is it calculated. This task is in the stage of "to be done." All possible stages of a task are:

- To be done (not started yet),
- In progress (started),
- Dev OK (development is done),
- DPLT ING OK (testing from the development team),
- Done (validated by the development team, IT owner, and application owner).

Table 3. Sprint Backlog US499

US	US499		Tasks running	
<p>Contact André</p> <p>New Sparc / NTS simulator. Phase 3. The current calculation mode is to be changed. Both correlation future and correlation actual should be modified.</p>				
Dev	T499.1	Discussion Task André	Done	0,5 h
Dev	T499.2	Analysis of existing correlation flow	To Do	3 h
CA-Test	T499.3	PAT Testing -US 499	To Do	2 h
Dev	T499.4	Development - Ratio calculation	To Do	2 h
Dev	T499.5	Development - Ratio/Median calculation	To Do	2 h

Dev	T499.6	Development - STD calculation	To Do	2 h
Dev	T499.7	Development - Correlation calculation	To Do	2 h
Dev	T499.8	Unit Testing	To Do	3 h
Dev	T499.9	Code Review	To Do	2 h

### 3.6.2 Sprint Velocity of a Scrum Team

A velocity chart shows the value delivered in each sprint. This allows us to plan the workload for future sprints. Based on the analysis of previous sprints we can better predict the work that the team can accomplish in the next sprint (Atlassian, 2013).

According to Kumar (2017), to calculate story points, we need to consider the size, effort, and complexity of each US. Some of those values will give us story points for each US. The sum of those points in each sprint will give us the total story points per sprint.

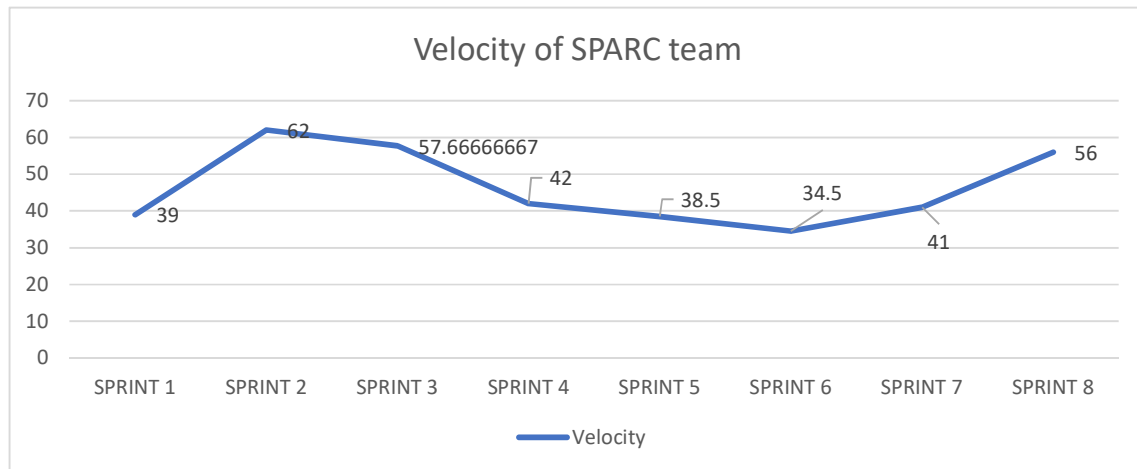
In the practical case shown in Table 4 and Figure 15, I used complexity and size to calculate story points for each US. I could not use effort since we do not document the effort needed for our US. The data presented is from the past 8 sprints that we had in our development process. The duration of all 8 sprints is the same if we are only counting working days. The duration in working days for all 8 sprints is 10 days. In my analysis, the start of the first sprint was on May 5<sup>th</sup>, 2017, and sprint 8 started on the 14<sup>th</sup> of August. The velocity that is visible from our example shows a huge deviation in the first sprint and sprints 4, 5 and 6. After thinking about the reason for the low amount of story points, I found two answers.

The reason for only 39 story points in sprint1 was the recent replacement of the development team leader. This was the first sprint with new leadership in the development team. The team was still adapting, and it is completely normal that we had low velocity. However, sprints 5,6 and 7 do not have the same origin of low story points. The most probable reason for them is that this was the time for summer vacation for our team. Those three sprints were planned from mid-July to mid-August when most people plan their vacations. The reason why the velocity had fallen was because the SPARC team for those sprints was not complete.

Table 4. SPARC Team Velocity Table

Sprint Number	Number Of US	Total Story Points	Velocity
SPRINT 1	10	39	39
SPRINT 2	19	85	62
SPRINT 3	10	49	57,6
SPRINT 4	12	42	42
SPRINT 5	10	35	38,5
SPRINT 6	9	34	34,5
SPRINT 7	8	48	41
SPRINT 8	9	64	56

Figure 15. Velocity Chart for SPARC Team



The SPARC team does not use velocity analysis. We plan our next sprints without historical data support, and they are only based on experience which comes from the Scrum master. This tool could help us better estimate our development capabilities and plan our sprints in a much more accurate way. The calculation of this report is very simple, easily understandable, and we already have everything to implement the usage of it.

### 3.7 Testing and Validation of User Stories

The development stage of each US starts on day one of the sprint, and it is very likely that by day 3 there will be enough developed that the US start the testing phase. The point is to start testing as soon as possible. By this time the testing team has already prepared the test case scenarios. Some of those test scenarios can be automated while others can be manual depending on the nature of the US. The testing team has to be part of the Sprint Planning

Meeting to understand the US's acceptance criteria. After the testing team has finished their job, then the application owner has to take a look at the US and share their opinion on it. If there are any minor changes, they can share them with the developers, but they have to be minor enough that they can still be completed in the current sprint (Mehra, 2017).

In the SPARC team, we have 3 lines of testing. There is a testing team in India that creates testing scenarios, and during the development, they do their tests according to the scenarios they have written down. Most of the errors and corrections are found at that stage and since they have direct contact with the development team they can explain and integrate changes extremely quickly. In most cases, the US is delivered to the testing environment on day 2 of the sprint. This is the first prototype of the US which is not yet finished, but certain test cases can be done already. When the development team had finished with the US and the testing team has validated their tests, then the IT owner is informed that they can do their final control before the US is passed to done. The IT owner verifies the functionality of each US and also checks the developer's code. If in doubt, the IT owner talks to the application owner to clarify business demands. When the IT owner confirms the US, then they inform the application owner that they can do the final validation of the US. In certain cases, the application owner comes with a few minor suggestions that are quickly implemented. After the validation of the application owner the US can be moved to "done."

We are currently facing many problems with our testing procedure. The main problem is that we are falling behind in testing. The development team completes their work, and the US is tested by the India test team; however, it cannot be tested by the IT owner and application owner. This means the US cannot pass to done, which obliges us to move developing the US to the next sprint because it has not been tested. The main reason why we are lagging behind development is that our team in France is much smaller than the team in India. The IT owners that have to validate each of the US are just overloaded.

Another major problem is that the testing team in India does not know exactly what the acceptance criteria are for each US since they do not participate in sprint meetings and during the sprint planning the application owner never defines acceptance criteria. If the testing team had a better idea of what to test and what is requested for each US, it would be much easier for them to do relative tests and increase the chances of finding and blocking bugs. This would also reduce the pressure on the IT owners who could invest much less time in testing.

To improve testing and US validation, we could include the testing team in our sprint meetings. In the same meeting, the application owner should list the acceptance criteria for the testing team which would help a lot. By applying this in future sprints, we could improve the US quality of each future sprint.



### 3.8 Release Burndown

Another very useful tool that we can use in our Scrum method is release burndown. This is a chart which is the most common sprint tracking mechanism used in Scrum. The usage and application of such a chart does vary among the users; some use story points others use task count to measure values of US. The tool is very useful for the entire Scrum team since they can see the current progress of each release. By looking at the chart, they can identify if they are behind in development. This chart should be managed by the Scrum master, and it should be updated at the end of each sprint (Mittal, 2013).

Table 5 is the data that I collected from our latest release which is not finished yet. Release burndown is a tool that is known to everyone in the team; however, nobody in our team has ever created a release burndown chart. This tool could be very useful for the management team and the Scrum master himself. Currently, the Scrum master is counting the US and giving the total number of US that is left to be completed in the current release. The report itself is simple to create; the only thing that has to be done is to document the complexity and effort of each US. After that, we must sum the total number of story points and subtract the sum of story points completed in each sprint. The number of story points should be approaching 0 with every sprint, and with the completion of the last sprint, it should be 0.

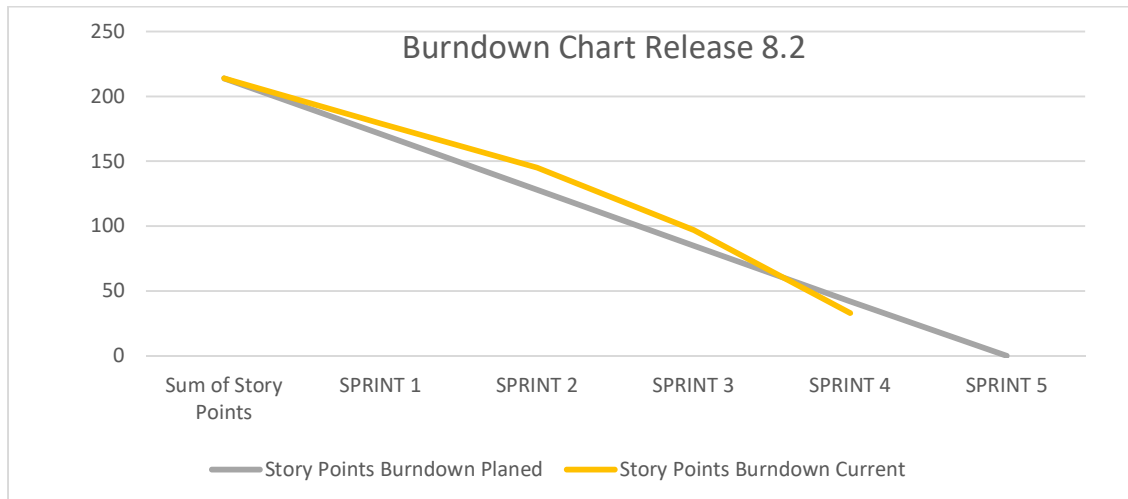
Table 5. Realease Burndown Data

<b>Sprint Number</b>	<b>Number Of US</b>	<b>SP Completed</b>	<b>SP Burndown Planed</b>	<b>SP Burndown Current</b>
Sum of Story Points	42	214	214	214
SPRINT 1	10	35	171	179
SPRINT 2	9	34	128	145
SPRINT 3	8	48	85	97
SPRINT 4	9	64	42	33
SPRINT 5	6		0	

Figure 16 is an example of a release burndown chart which is based on the data gathered from SPARC release 8.2. When we look at the graph, we can see the same problem that we had with the velocity. During sprints 1 and 2 we were behind schedule, meaning that if we continue with the same pace then at the last sprint 5 we would not finish all the US planned in 8.2. In sprints 3 and 4 we had increased the pace, and at the end of sprint 4, we were even a bit ahead of schedule. The reason for the slower start is because people were on vacation and many team members were missing. For sprints 3 and 4 the team was complete again.

Currently, we are working on sprint 5 which seems to be on track to finish every US in the current release.

*Figure 16. Release Burndown Chart*



## CONCLUSION

In my master's thesis, I analyzed different agile methods that appear today in development teams. All agile methods have a very similar approach; however, they do have some key differences that help us distinguish between them. The differences between the methods are mostly because the nature of each development team is not the same; therefore new methods were adopted to satisfy the needs of each team.

After the comparison, I made between traditional and agile methods I came to the conclusion that today agile methods are used much more often and are more suitable for most companies. Business needs simply cannot be defined so far in the future, since businesses change very rapidly. The use of an agile method helps the development team to be more responsive to business needs, which is crucial today for a company to be competitive.

After the analysis of existing theories on agile methods, I started to work on a practical case. My goal was to find differences between Scrum method in theory and the Scrum method that we are using in the SPARC team at Renault. Since we are quite new to Scrum method, the differences were significant, and the usage of Scrum method is still very sloppy. There are multiple key features that we are not using on our team such as Daily Scrum, face to face communication over written communication, reporting tools such as velocity calculation and release burndown chart, etc. Implementing these key elements would help us improve our development process and make us even more agile. However, the main problem that we have will be very difficult to overcome. The main issue is that our team is very scattered around the world, which makes it almost impossible to fully implement Scrum method.

Actions taken by the Scrum master that can have an impact on our current procedure is a software change that we use for our Scrum method. Currently, we are using Kados, which is very good software and the entire team is used to it. It has good visual support and therefore does not require too much effort to use. The Scrum master suggests that we move forward to different software called JIRA. It is a program that is being widely promoted globally and more teams at Renault are adopting it in their daily process. It is more sophisticated, and it also allows certain automated reports that can be very useful for us. For instance, burndown and velocity charts are automatically generated in JIRA which would encourage our team to use those reports to plan future sprints or check on current progress.

The subject that I have not discussed in my master's thesis is current trend at Renault where we are outsourcing more and more resources to India where the workforce is much cheaper. Therefore I fear that the idea of having the entire Scrum team located in France is unrealistic. It is more likely that the entire team will be moved to India to cut the company's costs. This move would be good for Scrum method quality since the team would no longer have

language differences and any cultural differences. They would all be located at the same site and could apply all aspects of Scrum much more easily. However, by doing so, the team would be far from their business users since the tool is mostly used on a corporate level and our corporate level is located in France. So I do not see this drastic change taking place for at least another 5 years. However, it is more and more likely since labor costs are becoming unsustainable.

## POVZETEK V SLOVENSKEM JEZIKU

Moja magistrska naloga je sestavljena iz treh ključnih delov, ki se nanašajo na uporabo agilnih metod. Naloga je sestavljena iz teoretičnega in praktičnega dela. Praktični del temelji na delovni izkušnji, kjer uporabljamo Scrum metodo na ključni aplikaciji pri Renault SAS.

V prvem delu ki je teoretični del sem zbral literaturo iz katere sem povzel ključne misli avtorjev za vsako metodo ki je danes v uporabi. Vse metode ki sem opisoval so agilne metode in zato imajo precej skupnih značilnosti. Je pa sicer nekaj ključnih razlik, ki so nastale zaradi različnih potreb razvijalnih ekip. Podrobneje sem opisal agilno metodo Scrum, saj je to metoda katero poskusimo implementirati v celoti v nas delovni proces. Zadnji element v teoretičnem delu je primerjava agilne metode z tradicionalno metodo. S tem sem želel doseči pogled na razliko v razvijalnem procesu, saj je naša predhodna metoda bila prav Waterfall metoda in sem želel identificirati elemente waterfall metode, ki so se zmeraj prisotni v našem procesu.

Drugi del magistrske naloge je predstavitev delovnega okolja v Renault. Sprva sem predstavil prodajni oddelek z ključnimi delovnimi mesti, ki so tudi končni uporabniki SPARC aplikacije. V nadaljevanju sem predstavil SPARC aplikacijo ki je ključni predmet naseljevanja magistrske naloge. Celotna Scrum metoda temelji prav na SPARC aplikaciji. Predstavil sem vlogo aplikacije v celotnem Renault Informacijske Sistemu. Glede na to da SPARC obstaja že 16 let sem predstavil tudi OLD SPARC in NEW SPARC katerega razvijamo z uporabo agilne metode Scrum. Predstavil sem vizijo naše aplikacije in tudi vire ki jih imamo na razpolago.

V tretjem delu magistrske naloge sem pa podrobneje analiziral metodo Scrum, ki jo uporabljamo v SPARC ekipi. Začel sem z programsko opremo Kados, ki nam pomaga pri izvajanju Scrum metode. Predstavil sem ključne funkcije in pomanjkljivosti orodja. V nadaljevanju sem predstavil Scrum ekipo. Tukaj sem navedel nase vloge v ekipi in komunikacijske probleme do katerih prihaja. V naslednji temi sem predstavil ključno delo Lastnika Aplikacije. Kot eno izmed njegovih glavnih opravil je identifikacija novih orodij / sprememb. Predstavil sem način s katerim Lastnik Aplikacije pridobiva relativne ideje za izboljšavo aplikacije. V naslednjih poglavjih sem predstavil ključno delovanje razvoja, ko so zahteve že opredeljene z strani Lastnika Aplikacije. Velik poudarek sem dal na »Product Backlog« in »Sprint«. Ta dva elementa predstavljata srce Scrum metode, brez katerih Scrum ne bi obstajal. Pri vsakem sem izpostavil probleme ki nastajajo v našem procesu za katere sem nato poskušal s pomočjo teorije iz prvega dela poiskati rešitev. V zadnjih dveh poglavjih sem tudi predstavil dva praktična orodja, ki jih trenutno ne uporabljamo v naši ekipi bi pa bila oba zelo dobrodošla, saj bi pripomogla k boljšemu planiranju in zmanjšanju negotovosti v razvojnem procesu.



## REFERENCE LIST

1. Agile Modeling (n.d.). Retrieved 4 March 2017 from <http://agilemodeling.com/essays/communication.htm>
2. Alexis, L. (2015). *Software Configuration Management Handbook 3<sup>rd</sup> ed.* Narwood, MA: Artech House.
3. Atlassian. (2015). *Viewing the Velocity Chart*. Retrieved 30 August 2017 from <https://confluence.atlassian.com/agile/jira-agile-user-s-guide/using-a-board/using-reports/viewing-the-velocity-chart>
4. Aubry, C. (2010). *Scrum Agilité et Rock'n Roll*. Retrieved 28 August 2017 from <http://www.aubryconseil.com/post/Features-et-stories>
5. Chappelle, D. (2016) *The three aspects of software quality: functional, structural, and process*. Retrieved 4 March 2017 from [http://www.davidchappell.com/writing/white\\_papers/The\\_Three\\_Aspects\\_of\\_Software\\_Quality\\_v1.0-Chappell.pdf](http://www.davidchappell.com/writing/white_papers/The_Three_Aspects_of_Software_Quality_v1.0-Chappell.pdf)
6. Cohn, M. (2017). *Scrum Product Backlog*. Retrieved 29 August 2017 from <https://www.mountingoatsoftware.com/agile/scrum/scrum-tools/product-backlog>
7. Green, M.D. (2016). *4 agile Ways To Handle Bugs in Production*. Retrieved 29 August 2017 from <https://www.sitepoint.com/4-agile-ways-to-handle-bugs-in-production/>
8. Hayes, S. (2009). *Agile Development in the Irish Software Industry: Models for Change*. Newcastle: Cambridge Scholars Publishing.
9. Hughes, R. (2013). *Agile Data Warehousing Project Management: Business Intelligence Systems Using Scrum*. Waltham: Morgan Kaufmann.
10. James, M. (n.d.). *Scrum Methodology*. Retrieved 4 March 2017 from <http://scrummethodology.com/>
11. JISC. (2015). *An Introduction To PRINCE*. Retrieved 22 April 2017 from <http://www.prince2trainingen.nl/PRINCE2Websitefiles/Prince2%20introduction.pdf>
12. Jongerius, P. (2014). *Get Agile Scrum for UX, Design & Development*. Amsterdam: BIS Publishers.
13. Kumar, S.J. (2017). *Story Point Estimations in Sprints*. Retrieved 30 August 2017 from <https://www.scrumalliance.org/community/articles/2017/january/story-point-estimations-in-sprints>

14. Mehra, A.J. (2017). *Story Point Estimations in Sprints*. Retrieved 30 August 2017 from <https://www.scrumalliance.org/community/articles/2016/july/a-testing-strategy-in-scrum>
15. Mittal, N. (2013). *The Burndown Chart: An Effective Planning and Tracking Tool*. Retrieved 2 September 2017 from <https://www.scrumalliance.org/community/articles/2013/august/burn-down-chart-%E2%80%93an-effective-planning-and-tracki>
16. Prakash, A. (2013). *Dealing with Bugs in the Product Backlog*. Retrieved 29 August 2017 from <https://www.scrumalliance.org/community/articles/2013/october/dealing-with-bugs-in-the-product-backlog>
17. Renaud, M.G. (2015). *Les différents cycles de développement en informatique*. Retrieved 7 May 2017 from <http://www.responsive-mind.fr/cycles-developpement-informatique/>
18. Renault SAS. (2017a). *New SPARC Evolution* (internal material). Paris: Renault SAS.
19. Renault SAS. (2017b). *DPP Département documentation* (internal material). Paris: Renault SAS.
20. Renault SAS. (2017c). *Documentation de SPARC* (internal material). Paris: Renault SAS.
21. Resnick, S., Bjork, A. & Maza, M. (2010). *Profesional Scrum with Team Foundation Server 2010*. Indianapolis: Wiley Publishing, Inc.
22. Rico, D.F., Sayani, H.H. & Sone, S. (2009). *The Business Value of Agile Software Methods*. FL: J. Ross Publishing.
23. Robson, S. (2013). *Agile Sap Introducing Flexibility, Transparency and Speed to SAP Implementations*. Cambridgeshire: IT Governance Publishing.
24. Ruler, B. (2014). *Reflective Communication Scrum*. Hague: Eleven international publishing.
25. Schwaber, K. & Sutherland, J. (2013). *The Scrum Guide*. Retrieved 4 March 2017 from <http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf>
26. *Software Development methodologies*. Retrieved 22 March 2017 from <http://www.itinfo.am/eng/software-development-methodologies/>
27. Young, D. (2013). *Software Development Methodologies*. Retrieved 22 April 2017 from [https://www.researchgate.net/publication/255710396\\_Software\\_Development\\_Methodologies](https://www.researchgate.net/publication/255710396_Software_Development_Methodologies)