CHALMERS
UNIVERSITY OF TECHNOLOGY

Challenges and Strategies for Balancing Plan-Driven and Agile Requirement Engineering

Master of Science Thesis in Software Engineering

Swathi Gopakumar

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Gothenburg, Sweden 2016

Challenges and Strategies for Balancing Plan-Driven and Agile Requirement Engineering

SWATHI GOPAKUMAR

Supervisor: Dr. ERIC KNAUSS
Examiner: Dr. REGINA HEBIG

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Gothenburg
Sweden
Telephone + 46 733595151

Acknowledgements

Firstly, the author would like to thank Dr. Eric Knauss, the academic supervisor of this study for his guidance and invaluable feedback throughout the study. The author would also like to extend special thanks to the company contact, Lars Ljungberg for collaboration, guidance and support throughout the study. Finally, the author would like to thank the examiner Regina Hebig for agreeing to be the examiner of the study.

Swathi Gopakumar, March to October 2016, Gothenburg

I dedicate this work to my loving husband, my adorable son, family and in-laws, for their endless love, support and encouragement.

Abstract

The main characteristic of traditional software methodologies is following a series of sequential steps like requirement specification, design, development and testing and deployment. Client requirements are documented to the maximum possible extent. This is followed with visualising the general architecture of the software which is later followed by coding. Then comes the different testing process and the final deployment. Basically, there is deep planning and visualization of the project even before building the project, and then working one’s way through to the planned finished structure. The agile method of software development is a contrast to the former. The focus here is on iterative and incremental development, where all the phases of the process are periodically revisited. A lot of importance is given to adaptability. Agile Methods sprung up as a reaction to traditional ways of developing software and acknowledge the need for alternative ways to the old documentation driven, heavyweight software development processes.

Both traditional and agile approaches imply different ways of handling requirements. Each of them have their specific benefits and challenges. Some of the challenges of traditional RE are lengthy time consuming documentation, irrelevant requirements leading to requirement waste, not accommodative of late requirement changes, minimalistic customer interaction etc. While agile RE challenges are related to insufficient documentation, lack of budget and schedule estimation, neglect of non-functional requirements, no support for requirement traceability issues, requirements volatility etc. With several challenges of traditional/ plan driven requirements and agile/ change driven requirements mentioned in literature, it would be beneficial if organisations could find a way to work with requirements by taking the best of both worlds. A good idea is to find a balance in working with both forms of requirement engineering.

With software intensive industries gradually shifting from traditional software development process to adopting several agile methods, it is important to explore how this balance can be attained. To explore this balance, we conducted case study at a premium automotive company that works with plan driven requirements as well as agile requirements or user stories. The case company has been successful in gaining the benefits of both requirement engineering forms by achieving to find balance in working with both traditional and agile methods. For our case study, we conducted eleven semi-structured interviews within this automotive company to explore how they worked together with both types of requirements handling. After extracting the challenges found in the company’s requirement model, we conducted a focus group discussion in order to validate our findings. The focus group enabled us to gather insight on strategies that could be applied to resolve the challenges found to a large extent. The focus group included faculty from Chalmers and some of the employees from the automotive company.

This study can be replicated in other research settings to obtain ideas to strike balance between plan driven requirements and agile user stories and dig out other complicated challenges that could arise when trying to attain balance.

Contents

1 Introduction

Earlier, the automotive industry was purely mechanical and hardware based. Lately OEM's have become more software focussed as they have begun to recognise the opportunities provided by software. The software development is typically embedded into a V model life cycle [1]. In the V model life cycle, different processes are executed in a sequential way. Here, every new phase begins only after the completion of the old phase. The traditional V model way of working offers some advantages and disadvantages. The V model defines all steps and deliverables, making it an easy to follow model. Thus, working in a V model seems to feel easy. An advantage of V model is that it gives a good overview over complex development effort. Thus, it makes it relatively easy to synchronize development of mechanic, electrical and software components. The major disadvantages of this model are rigidity and low flexibility. Since the V model works in sequential steps, it is difficult to go back to requirement analysis once past the requirement phase. If changes have to be made mid-way, then both requirements and test documentation needs to be updated. Due to this, defects have to be found at the early stages itself in the V model [60]. With respect to Requirement Engineering (RE) in the V model, Salim et al. [2] highlight some of the challenges with this model as extensive documentation that increases lead time, unnecessary requirements leading to requirement waste, inability to address late requirement changes, minimum customer interaction etc.

Moraes et al. [3] state that, in recent times, many software organizations are employing agile methodologies. Unlike the traditional V model, agile methods offer flexibility by allowing changes in requirements at a later stage of the development cycle. It also offers dynamicity by satisfying customer needs in less time and even enables continuous delivery of useful working software [3]. There is close cooperation between business people and developers when working with this model. Additionally, there is close interaction between customers, developers and testers as well. However in this methodology, it is difficult to predict the efforts required in large projects beforehand as much emphasis is not laid on necessary design and documentation processes [2]. According to Daneva et al. [2], some of the challenges associated with agile RE are insufficient documentation, lack of budget and schedule estimation, neglect of non-functional requirements, no support for requirement traceability issues, requirements volatility etc.

Studies from Inayat et al. [3] emphasize that agile methods have been able to solve several problems of traditional software development methods because of its flexibility and emergent nature. However there is not much knowledge available about the solutions that agile RE have brought to traditional RE problems. Again, little is known about the possibility of agile RE opening up new challenges while bringing solutions on traditional RE challenges [3]. Considering the above facts, it can be said that both plan driven RE and agile RE have their own set of benefits and challenges. This motivates the need for a new way of working that incorporates the best of both traditional RE and agile RE methods when working with requirements. For this reason in the case study, we chose to investigate this topic

The purpose of our study is to explore new methods and processes of working with requirements which will bring out the benefits of both traditional RE and agile RE practices while also compensating their challenges. Further, we aimed to understand out how plan driven RE and agile RE can be made to coexist together in large scale software intensive automotive sector.

For our case study, we chose a reputed automotive company, based in Sweden, as our research setting because they were working with a requirement model that accommodates aspects of both traditional RE and agile RE methods. We wanted to determine how the model was accommodating both requirement engineering forms and whether it was able to bring in the benefits of both traditional RE and agile RE practices and reduce their individual challenges.

We expect, our study to address the research gap in academia by overcoming the challenges of different requirements engineering practices related to plan driven model like V model and agile software development framework. Specifically, we have addressed this gap by analysing the requirement model and RE practices in the case company. Also, our analysis of the case company’s RE practices have identified their current existing challenges related to RE. We have also come up with possible strategies for overcoming the mentioned RE challenges.

This remainder of this paper is structured as follows: Chapter 2 presents background on software development lifecycle models like V model, agile software development framework, requirement model at the case company and requirements engineering methods adopted by V model and agile software development framework. Chapter 3 presents related work on challenges of different requirement engineering practices and challenges in automotive software engineering. Chapter 4 presents research methodology. In Chapter 5, we present the research findings while in Chapter 6, we provide discussions. Conclusion and possibilities for future work are stated in Chapter 7.

2 Background

In this chapter, we present the background on Software development lifecycle models like V model and agile software development model, and some of the Requirements Engineering methods adopted by them. We will also be very briefly describing the requirement model at the case company in this chapter.

2.1 V Model

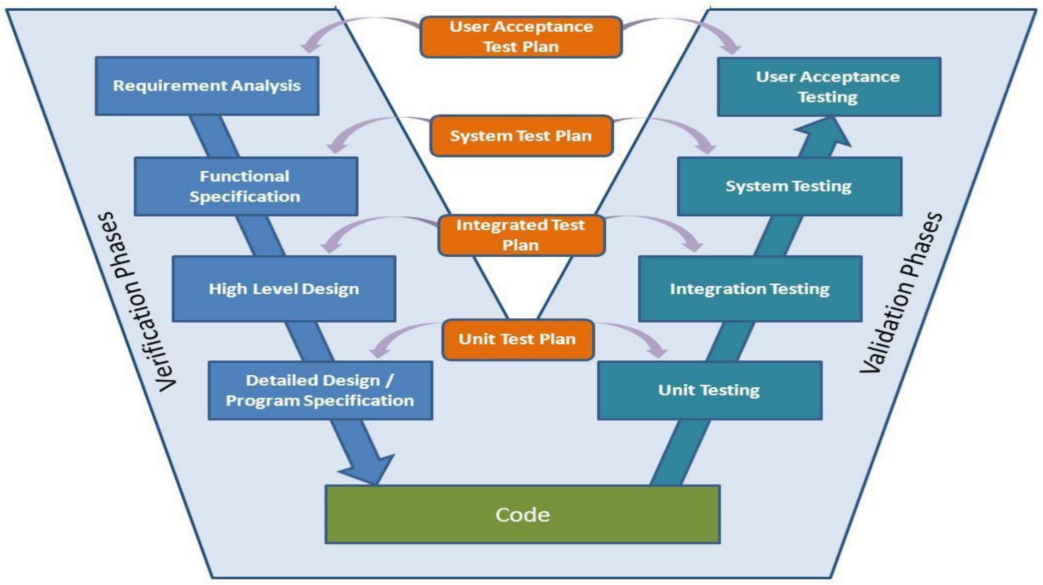

Figure 1: V model (Image Courtesy: http://crackmba.com/v-shaped-model)

The V model shown in Figure 1, represents development process in software development. It is an SDLC model where in processes are sequentially executed. Also known as the verification and validation model, it is an extension of the waterfall model where each development phase is associated with a corresponding testing phase. As mentioned by Andreas et al. [62], other models for planning and realisation of projects for software development include V Modell and V Modell XT, which provide flexibility and support agility in software process. The different phases of software V model along with the relationship between each development phase and its corresponding testing phase is illustrated below.

2.1.1 Left Hand side

In the following, we discuss the Left Hand side of the V model.

- *Requirement analysis:* The needs of users are analysed to collect requirements of the system. After the elicitation process is completed, user requirements document is generated. Business analysts use the requirement specification document as a means to communicate what they understand of the system to the users. The specification document is later carefully reviewed by the users. This phase is usually concerned with determining what the ideal system will do and does not relate to software design or built.

- *System design:* Once the requirement specification document is ready, it's time to design the system. By studying the user requirements documented, system engineers understand the business of the intended system. The possibilities of whether all user requirements can be implemented are discussed here. The user is informed in case any of the requirements are not possible. Certain negotiations are reached and the requirement documents are edited. The software specification document generated in this phase guide the development phase.

  Based on the system design, system test plans are developed.

- *Architecture design*: As the name suggests, architecture specification are understood and designed here. Of the several technical approaches proposed, the most feasible one is selected. System design is broken further into modules of different functionality. The designed architecture should realise the list of modules, functionality of each module, interface relationships, dependencies, databases, architecture diagrams etc. Communication and data transfer between internal modules and with other systems is understood and defined here.

  With the information from architecture design, integration tests are documented.

- *Module design:* The already designed system is broken further into smaller modules. Also referred to as the low level design, the detailed internal design for all the system modules are specified here. It is necessary that the design is compatible with other modules of system architecture and also with external systems. The low level design document contains functional logic of the module in detail (pseudocode). The programmer uses this for coding purposes.

  Based on the internal module designs, unit tests are designed.

- *Implementation Phase:* Coding is done in the implementation phase. Developers convert module design to code here. After the completion of coding, the right hand side of the V model starts. The test plans that were written earlier are now executed.

2.1.2 Right Hand side

In the following, we discuss the Right Hand side of the V model.

- *Unit testing:* In the verification phase of the V model, during module design, several unit test plans are developed. During the corresponding validation phase, these unit tests are executed on code which help to remove bugs at the code level. It verifies whether smallest units like a program module are correctly functioning when in isolation with rest of the code/ units.

- *Integration testing:* In the verification phase of the V model, during architectural design phase, integration test plans are developed. Integration tests verify if all the units that were tested independently can communicate and coexist together. Customer gets to see the test results.

- *System testing:* In the verification phase of the V model, during system design phase, system test plans are developed. System test plan tests the whole system for functionality, and checks the communication and interdependency of the system under development with external systems. It also verifies if both functional and non-functional requirements are met.

- *User acceptance testing*: In the verification phase of the V model, during requirements analysis, test plans are developed by business users. It is performed using real data in a user environment closely resembling the production environment. Compatibility issues with other systems in user environment can be found with these tests. The tests verify if the system meets the user's requirements.

2.2 Requirements engineering in general

Software development process is a collection of several phases. Requirement engineering, one of the phases in this process, plays an important role for every software product, as Shukla [5] mention. As reported by Pohl [6], the area of requirements engineering is gaining lot of attention owing to its area of growing importance**.**

According to Pohl [7], RE is the process of eliciting individual stakeholder requirements to later develop them into detailed, agreed requirements documented and specified in a way that can serve as the basis for all other system development activities. Requirements are attributes or something, which is discovered before building products. It is a condition that must be met by a system to satisfy a contract, standard, or other formal documents [1]

As stated by Sawyer [8], the goal of requirements engineering is to produce a set of system requirements which, as far as possible, is complete, relevant, and consistent and reflects on what actually the customer needs.

In accordance with Inayat et al. [3], in traditional requirements engineering, the elicitation process includes users getting involved in gathering requirements. During the elicitation process, users try to understand business needs, stakeholder needs, domain, system constraints, and the problems the system is supposed to solve. After the completion of elicitation process, requirement analysis is performed to check for necessity, consistency, completeness, and feasibility of requirements. The selected necessary requirements is written as a requirements specification document to communicate with stakeholders and developers. The next step in the process is requirements validation. Requirement validation is important for confirming customers’ real needs to avoid errors in requirements specification document which could result in expensive rework if discovered late during development. The last phase is the requirement management phase where requirement changes are dealt with. These changes originate from changes that come from stakeholder’s understanding of what the system is expected to do during the development process. Requirement changes are often not welcome as it involves lots of rework, costs and sometimes customer dissatisfaction.

The major requirement engineering activities include requirement elicitation, documentation, analysis, prioritisation, tracing, verification, validation and management

[11]. In what follows, we elaborate on some of the above traditional requirement engineering practices namely:

2.2.1 Requirement Documentation

As Paetsch et al. [12] highlight, requirement documentation serves as a way of communication between stakeholders and developers. It acts as baseline for evaluating product and processes like design, verification and validation activities, testing and for change control. Some of the features of a good requirements specification document are namely:

- Completeness
- Correctness
- Unambiguous
- Understand ability
- Consistence
- Concise
- Feasible

2.2.2 Requirement Prioritization

As put by Paetsch et al. [12], “It is absolutely essential that the most valuable features are delivered at the earliest especially when the project is running on a tight schedule with limited resources and high customer expectations”. Though the customer does the requirement prioritization, both customer and developer can provide inputs for the same. While customer points out features which provide most benefit to users, developers point technical risks and costs involved. With further inputs from developers, customer might adjust some of the priority of features. Requirements can be prioritised using several prioritisation techniques.

2.2.3 Requirement Management

According to Lauesen [11], despite all the attempts made to specify requirements right the first time, requirements keeping changing during the software development process. Therefore, it should be possible to do addition of new requirements, updating of requirement specification, and assessing the consequences of change in requirements.

As Pandey et al. [13] reports, requirements management is performed continuously both after development and during maintenance because requirements usually always change. Considered as the most complex requirements engineering process, it involves keeping control and track of the following:

- Changes of agreed requirements.

- Relationships between requirements, dependencies between requirement documents and other documents produced during system and software engineering process.

2.2.4 Requirement Communication

According to Wnuk [14], efficient communication throughout a project’s lifecycle plays a key role in development and release of successful software products to the market.

Berry [15] explains that the process of requirement communication starts with the customer and later involves people from different roles throughout the development project. Initially, requirements elicited are communicated, then changes made to those requirements are negotiated and communicated between all possibly affected roles like requirements engineers, developers, and testers. Since requirement change is constant throughout the project, requirements communication must also continue during the entire project life cycle.

2.2.5 Requirement Tracing

Requirement tracing refers to following the life of a requirement. Tracing is done to establish relations with respect to the development artefacts [16, 17]. According to Lauesen [83], requirement tracing is of four types and happens in two directions:

2.2.5.1 Forward Tracing:

- Tracing from requirements to program to ensure all requirements are dealt with.
- Tracing from demands to requirements to check if all demands are fulfilled by requirements.

2.2.5.2 Backward Tracing:

- Tracing from requirements to demands to know if requirements serve the purpose.
- Tracing from program to requirements to realise whether all parts of program are needed.

2.2.6 Requirement Validation

According to Paetsch et al. [12], the purpose of requirements validation is to certify that requirements are an acceptable description of the system to be implemented. This means that requirement specification is validated by the customer to ensure that his needs are met. The customer reads and approves the specification if it meets his needs. The end result of requirements validation is a list of reported problems with the requirements document and the actions to correct or cope with the reported problems.

2.3 Agile/ Scrum Model

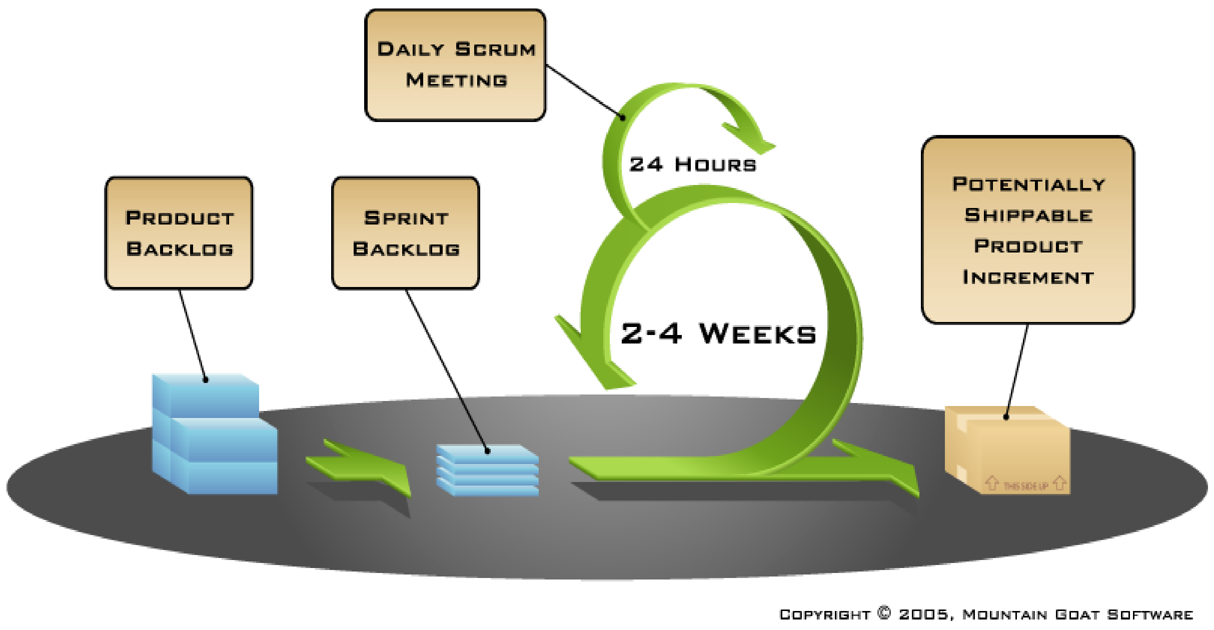
Daily Scrum Meeting
Product Backlog
Sprint Backlog
24 Hours
2-4 Weeks
Potentially Shippable Product Increment
Copyright © 2005, Mountain Goat Software

Figure 2: Agile/ Scrum Model (Image Courtesy: https://www.mountaingoatsoftware.com/agile/scrum/images)

As Beck [18] stated, "Agile Methods are a reaction to traditional ways of developing software and acknowledge the need for an alternative to documentation driven, heavyweight software development processes". As Abrahamsson and colleagues [19] highlight, agile software development are incremental, straightforward, cooperative, and adaptive. According to Cohen et al. [20], being agile means to deliver fast, and to respond quickly and frequently to change. Leffingwell [21] identifies the business benefits of software agility as increase in productivity, team morale, job satisfaction, faster time to market, and better quality. As Cockburn [22] highlights, most of the agile practices are not new concepts. Instead, it is the focus and values behind agile Methods which differentiates them from the more traditional methods.

As explained by Beck [18], the Agile Manifesto reads as follows:

- *"Individuals and interaction over process and tools".*
- *"Working software over comprehensive documentation".*
- *"Customer collaboration over contract negotiation".*
- *"Responding to change over following a plan".*

The Agile Manifesto is an important part of the Agile Movement which characterizes the values of agile methods and how Agile distinguishes itself from traditional methods [18]. Agile Methods favour common values, but differ in the suggested practices. Some of the followed agile Methods are: Extreme Programming, Scrum, Crystal Methods, Feature Driven Development, Lean Development, and Dynamic Systems Development Methodology. Scrum (as shown in Figure 2) and Extreme programming are the most common agile methods [20]. In what follows, we elaborate on Scrum as the studied company uses Scrum for software implementation.

As described by Schwaber [23], in 1996, scrum as a process accepts that the development process is unpredictable. Scrum projects are split over several iterations called sprints. The sprints consist of pre-sprint planning, sprint, post-sprint meetings, team size, iteration length, support for distributed teams and system criticality. Some of the practices of Scrum as Leffingwell [21] reports are:

- Sprints are iteration of a fixed duration.

- All work to be developed is characterized as a product backlog, which includes requirements to be delivered, defect workload, as well as infrastructure and design activities.

- Sprint backlog is written for a sprint. Work within a sprint is fixed. Once the scope of a sprint is committed, no changes can be added to the sprint backlog during a sprint.

- Scrum master mentors and manages the empowered, self-organizing, and self-accountable teams that are responsible for delivery of successful outcome at each sprint.

- A daily stand-up meeting is a primary communication method.

2.4 Agile Requirement Engineering/ User stories

As put by Sillitti [26], agile methodologies do not rely on standards of traditional development requirement processes [24, 25] but have adopted some of the ideas on requirement elicitation and management from it.

Beedle [45] explains that in agile software development, the customer and the entire development team is involved in requirement elicitation and management. Teams engaged with agile development capture high level requirements, just-in-time for feature development. Agile requirements are brief and written just sufficiently enough, to enable development and testing process with efficiency. The rationale behind this is to minimise spending time on activities like documentation that don't actually form part of the final product. Usually, the agile teams capture these high level requirements by working together in a collaborative manner so that the team members understand the requirements as well as each other. Unlike traditional projects, where Business Analyst independently gather and write requirements; it is the joint team activity that allows everyone to understand and contribute to what's needed.

According to the Agile Manifesto by Beck [27], collaboration between development team and customer is maintained through involvement of customer in the development process instead of fixed contracts. Agile practices emphasize on continuous interaction with the customer to address requirement elicitation, specification, documentation, prioritisation, changes and delivering most valuable functionalities first.

Maurer [28] states that, in agile software development, complete requirement specification document is seen as infeasible. Since there is direct interaction between the development team and the customer in agile software development, number of documents and its usage to share knowledge is reduced. Also misunderstanding is lessened among team members because of the missing unnecessary communication layers. An important thing to note by Sillitti [26] is, since, the documentation is reduced to minimum, there is often a need for clarification of requirements. The presence of onsite customer helps replace the documentation of requirements in details. In case, a requirement is considered complex, the customer splits the requirement. Splitting of requirement helps developers to better understand the functionality. Requirements are collected in customer language and do not follow any formal language for requirement specification.

As per Succi [26], waste requirements can include both wrong and useless requirements. AM focus on avoiding wasteful requirements and only delivering value to the customer. For this, the development team produces only those features what the customer needs and perceives as useful, thus, eliminating requirement wastage. In order to avoid requirement wastage, the development team and customer team assign priorities to individual requirements to identify features of most priority. The highest priority features are implemented first so that the most important business value is delivered first. Leite [29] states that prioritisation is repeated over the entire development process to keep priorities updated. According to Succi [26], prioritization is done in four steps:

- Estimation of time for functionality implementation is determined by the development team. Requirements requiring high efforts are split into simpler ones requiring less effort for implementation.

- Business priority is allotted to each functionality by the customer.

- Risk factor is assigned to functionalities by the development team. This is done on the basis of business priority.

- In the next step, functionalities to implement in the iteration are identified. Both the customer and the development team is involved in this process.

.
Succi [26] goes on to add, that an important feature of agile methodology is that while time period is fixed, the features are variable. Also the AM accept and expect changes since they are aware that requirement variability is a common problem in almost all software projects. In this case, future changes are not forecasted, as focus is only on features which are paid by customer, thus, avoiding wastage of time and energy that comes with developing a general architecture.

As Eberlein et al. [29] say, it is tricky working on non-functional requirements with customers. Cysneiros [30] has reported numerous problems when eliciting non-functional requirements from customers. Sillitti [26] highlight that in AM, non-functional requirements are not explicitly specified. At every iteration, the customer gets to test the product. If problems related to non-functional qualities are found, the team tries to meet these NFR in the subsequent iteration.

2.5 Requirement Model at the Case Company

The case study was conducted at a reputed automotive company in Sweden. From the data gathered from the interviews, we were able to obtain information on the software development process model, plan driven RE and agile RE practices followed in the company.

In the department, new hardware platforms are integrated with sensors to get a view of the environment. The features and functions developed here are integrated into these hardware platforms. The department uses system supplier to develop the sensor platform and their own software is integrated into that processing hardware so that the component developed includes one part which takes care of the sensor while the other is a general purpose microprocessor.

The department follows a requirement model which supports requirements and agile user stories to coexist together. Very high level requirements for each project came from outside the department. These very high level requirements are also called attribute requirements. Inside the department, the Function Owner writes high level requirements in the System Weaver tool. These high level requirements are broken into parts and assigned to the person responsible, for the functions that are affected. The assigned employee further breaks down the requirement into very detailed requirements at the component level. In the next step, the component requirement specification is used for in house development or handed over as contractual document to external suppliers.

As soon as the requirements are broken down, the departments test division start to write test cases. The tests are developed independently of the source code for the actual software.

Currently, the software implementation works in an agile way, specifically scrum model in three week sprints. The agile team works with user stories in JIRA.

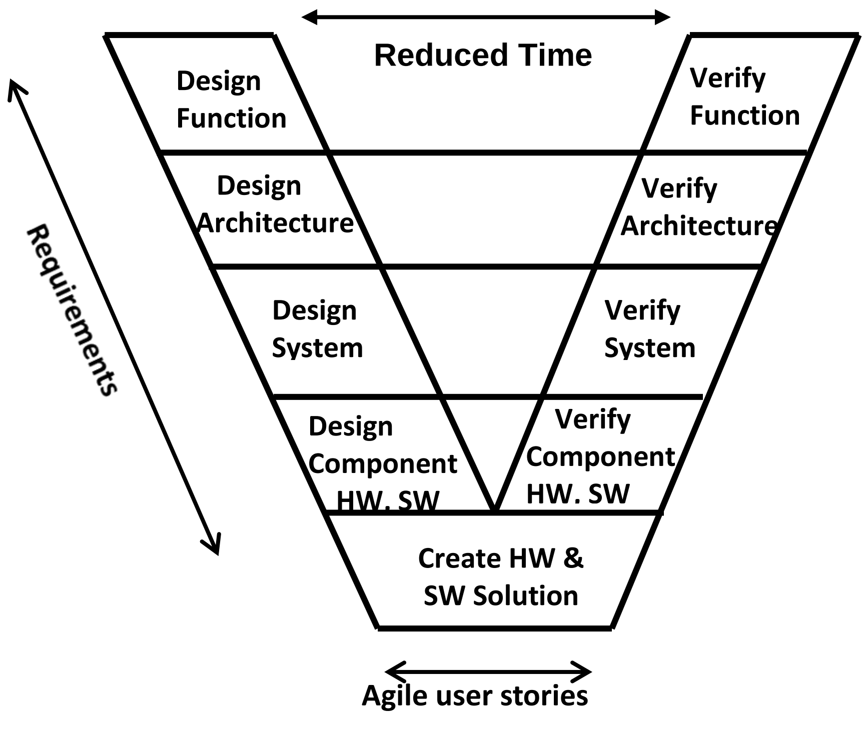

Figure 3: Requirement Model

As Figure 3 illustrates, the requirement model at the case company allows requirements and agile user stories to coexist together. While requirements are practised and followed on the left hand side of the requirement model, agile user stories are practised at the bottom i.e. implementation phase of requirement model. Details on how the model enables coexistence of requirements and user stories are explained in Chapter 5, Section 5.1

3 Related Work

OEM’s have begun to focus more on software in recent times as they have started realising importance of software like better management and shorter time to market, mention Knauss [1]. As highlighted by Broy [58], increasing size and complexity of software project results in several challenges in the automotive industry. Braun et al. [59] states inappropriate requirement engineering as the major challenge in the automotive domain. According to Amrit [4], many software organisations are now preferring agile methodologies and management practices. Many automotive companies have started accepting agile methods due to its flexible nature, mention Moraes et al. [3]. The software development community is unfamiliar with the role of requirements engineering in agile methods, highlights Inayat et al [2]. Salim [2] has done a systematic literature review to identify adopted requirements engineering practices and also understand if agile requirements engineering can solve traditional requirements engineering issues. The adoption of agile methods could impact the manner in which RE activities are conducted and pose newer challenges towards their realisation [2].

Also, software is embedded in a V model process. While taking advantage of these opportunities, organisations struggle with the formal process imposed on software development. This resulted in emergence of new ways of working with requirements; they are partly not supported, partly hindered by old tooling and processes for requirements engineering.

Firesmith [46] states that there are several drawbacks with the traditional RE process. The elicited requirements may be ambiguous, incomplete, incorrect, inconsistent and outdated. Improving the quality of requirements is important because bad quality requirement leads to increased development costs and results in time delay and stretching the schedule. Also, the customer can never elicit 100% of what he expects from the system to be developed beforehand as he is usually not fully aware of the system before development [46]. Large systems end up with several thousands of requirements which are derived into several subsystems. Spotting missing requirements is often difficult and their absence is not found until when the system integration, testing or deployment happens [46].

As stated by Wora [48], during the elicitation process, requirements elicited from stakeholders could include some unnecessary requirements. There could be serious consequences if irrelevant requirements are elicited. Detection and elimination of unnecessary requirements is painfully time taking, which means avoiding requirement wastage with traditional RE process is crucial.

In the traditional RE process, after the elicitation and analysis process, requirements are extensively documented into a requirement specification document. This requirement specification document written is often complex and lengthy. A lot of time and effort is put into writing lengthy requirement specification document. Also the technical language used in the document is difficult for non-technical customers to understand [3]. Customers are involved in limited ways during elicitation and specification phase. The creative functionalities of the system that are acquired from customers are not accommodated during intermediate development phase. Less customer involvement is often seen as reason for problem creation during requirement negotiation and validation in traditional RE, as Patel [49] mention.

In the V model, requirements are tried to get right at the first attempt. After all the stakeholders agree upon what will be the requirements of the system to be developed, the requirements are locked. But the customer could try to bring requirement changes at any time during the development phase [49]. However, introducing requirement changes is difficult with traditional RE process. Still, requirement changes coming from customer end need to be addressed at times though it is difficult. Since requirement change is inevitable in the software development process, requirement management is an important part of traditional RE. As per Suman [13], requirement management involves tracking changes of agreed requirements, and relationship between requirements, and dependencies between requirement documents and other produced documents. Often requirements are stored and managed either in paper documents or spreadsheets with dissimilar formats and are managed by various different individual groups. This individual management of scattered and distributed requirements makes it tough to authorize and assign people to do operations on requirements. Requirement management is often a problem with traditional requirement engineering process.

Just like requirements change and evolve throughout software development process, the stakeholder’s mind fluctuates with regards to priority of the requirements. As Shah [49] mentions, in the traditional RE process, it is difficult to change the priority of requirements once the initial priority has been allocated to all the requirements in the specification document.

As Damian [47] highlights, communication gap between RE team and the customer is a major disadvantage with the traditional RE process. The RE team and customer hardly communicate after the initial elicitation and validation process. This gap in communication results in generation of serious problems as you move towards design, modelling and implementation [47].

According to Marczak [2], agile RE practices are able to resolve some of the established challenges brought upon by traditional RE like extensive requirements documentation, over scoping of requirements, communication gap, requirement validation problems and less customer involvement. Communication problems that arise with traditional RE practices can be tackled by agile RE practices like frequent face to face meetings, collocated teams, onsite customer, alternate customer representations and integrated RE process. Meanwhile, over scoping requirements in traditional RE can be compensated through agile practices like gradual detailing and cross-functional teams. While requirement validation problems can be fixed with requirement prioritisation agile practice, lengthy requirement documentation issue can be overcome with writing user stories, an important feature of agile methodologies. Customer involvement which is very rare in traditional RE practices can be overcome by having onsite customer which is again an agile RE feature [2].

However it is unclear whether agile RE can solve all traditional RE problems [3]. As described in [3], it is also possible that agile methodologies have brought up new problems when bringing solutions to traditional RE problems. Some of the challenges brought upon by agile RE include minimalistic documentation, customer availability, budget and time estimation, incorrect requirement prioritisation, neglecting non-functional requirements, lack of requirement traceability, customer inability and agreement, customer contractual issues, requirements change and its evaluation [3].

Minimal documentation is one of the basic features of agile RE, as stated by Beck [27]. It is written as user stories. User stories are a list of features written in product and team backlogs. Most of the knowledge is acquired through knowledge transfer between customer and agile teams during meetings. This results in accumulation of a lot of tacit knowledge within agile teams [4]. All this tacit knowledge remains undocumented, meaning a lot of this knowledge could get lost for future reference.

Customer availability is often assumed for all agile meetings [27]. However lack of time from customer’s side pushes the need to compensate their availability with proxy customers. Sometimes customers/ proxy customers have insufficient domain knowledge, disagree to agree on certain issues and are unable to make correct decisions, all this resulting in problems within the project, mention Wnuk [50].

As stated by Regnell et al. and Boness et al., in agile RE, user story prioritisation is mostly performed by clients based on the business value [51] [52]. User stories are prioritised before the sprint development work begins [53]. According to Farid [54], prioritization needs to be done correctly because incorrect prioritisation causes loss of time, effort, money along with lot of rework that needs to be done following this.

Another issue with agile methods is requirement traceability. Also, known as following the life of a requirement. Since documentation is kept to minimum in agile RE, it is difficult to maintain traceability [3]. Also NFRs or quality requirements like usability, performance and security of the system are sometimes neglected in agile methods, as Bang [55] mention.

Booch [61] describes that modified ways of working with requirements are coming up and these are obstructed by old development processes and tools of requirements engineering. This made us realise that every new development and need in the organisation triggers changes in the organisation and development process, and tool support. In other words a versatile and adaptable tool is necessary to support new ways of working in an organisation like a new development process [61]. Further development process and tools for requirement/ user stories could help to erase the challenges offered by traditional RE and agile RE by allowing to adopt both traditional and agile RE in the right balance.

4 Research Methodology

In this chapter, we first present the reason for choosing our research question. Next, we discuss in detail about research methodology. Later, we present the research setting and research participants. Finally we present the data collection and the data analysis.

4.1 Research Question

Most of the prior work listed in the related work which is relevant to our study focus on the advantages and short comings of different requirement engineering practices. While some papers mention agile RE could help to overcome challenges of traditional RE, some mention they could also bring in some obstacles along the way. There is very limited study available in areas of how both forms of RE can work together. We think a lot of research needs to go into this area.

Based on the gap in existing research work and according to the thesis objective at the case company we formulated the following research questions:

RQ1: *How are plan driven RE and agile RE co-existing in the current way of working?*

RQ2: *What challenges exist when combining both plan driven and agile paradigms in systems engineering?*

The aim of the study is to explore a new way of working which can compensate the challenges of both traditional RE and agile RE practices. In addition, we aim to identify strategies to support plan driven RE and agile RE to coexist together.

To answer our research questions, we have prepared an interview guide to gather necessary data from our case company. The interview guide is listed in the appendix.

4.2 Research Foundation

Crotty's model [31] was conceptualized by Creswell [32] to formulate three basic questions that are central to Research Design. They are:

1. *What knowledge claims are being made by the researcher?*
2. *What strategies of enquiry will inform the procedures?*
3. *What methods of data collection and analysis will be used?*

The answers to the above questions provide rationale for choosing a particular research design.

As Creswell describes [32], researchers begin project with certain assumptions as to what and how they will learn. The claims can be called as knowledge claims. Four different thoughts on knowledge claims as described by Creswell [32] are:

- *Postpositivism*

- *Constructivism*
- *Advocacy/ Participatory*
- *Pragmatism*

Postpositivism knowledge claims: The other names for this position are science research, postpositivism approach, quantitative research, postpositivist approach. This approach does not recognise the traditional notion of absolute truth of knowledge. Postpositivist researchers assume knowledge to be conjectural [33]. Established evidence in research is considered imperfect here.

Constructivism knowledge claims: In this case, knowledge is claimed through alternative process and assumptions. Here, it is assumed human beings construct meanings as they engage with the world [31].

Advocacy/ Participatory knowledge claims: Kemmis and Wilkinson [34] have summarised this research enquiry as focussing on agendas for change, addressing problems in society and creating political debates to bring in changes.

Pragmatism knowledge claims: As per Cherryholmes [35], researchers are not confined to committing to any one single system of philosophy here. They have the freedom to choose methods and procedures that best suit their purpose for research.

Runeson and Höst [36] agree with the general definitions of case study term provided by Robson [37], Yin [38], and Benbasat et al. [39]. Case study is an empirical methodology aimed at investigating contemporary phenomena in their context [36.

Mentioned below are some of the other different research methodologies:

- *Action Research*
- *Survey*
- *Experiment*
- *Design Science*

Action research: Robson [37] describe the purpose of Action research is to influence or change some aspect of whatever is the focus of the research. As Avison et al. [40] state, action research is a qualitative research method emphasizing on researchers and practitioner's collaboration. Another definition by Rapoport [41] state, action research aims to contribute both to the practical concerns of people in an immediate problematic situation and to the goals of social science by joint collaboration within a mutually acceptable ethical framework. Action research is commonly used as research method for software process improvement, technology transfer studies and information systems.

Survey: According to definition by Robson [37], survey is a collection of standardized information from a specific population, or some sample from one, usually, but not necessarily by means of a questionnaire or interview.

Experiment: Robson [37] and Wohlin [42] define experiment as measuring the effects of manipulating one variable on another variable and that subjects will be assigned to treatments by random.

Design Science: Design science is a problem solving process. Understanding a design problem and its specific solution are acquired through building and application of an artifact.

4.3 Research Methodology

The research methodology used in our case was Qualitative Research approach. Also we preferred to do a case study. In what follows, we provide the rationale behind opting for Qualitative Research and conducting a case study.

4.3.1 Qualitative Research Approach

In quantitative approaches, the inquirer uses postpositivist knowledge claims i.e., cause and effect thinking, measurement and observation, hypotheses, reduction to specific variables and hypotheses and questions, test theories etc. Inquiries include experiments and data collection using instruments that yield statistical data. While in qualitative approach, the inquirer often makes constructivist knowledge claims (i.e., the different meanings of individual experiences, socially and historically constructed meanings and developing a theory or pattern.) or advocacy/participatory knowledge claims (i.e., political, collaborative, or change oriented). The strategies of inquiry used here are narratives, ethnographies, phenomenologies, case studies etc. Pragmatism allows choosing over multiple methods, assumptions and worldviews and various data collection, so it supports mixed method strategies.

Based on our research questions and research purpose (see Section 4.1), we found Constructivism world view the most apt, as for us, taking the view of participants, their experiences, is crucial for understanding a particular phenomenon. Therefore, we decided to follow a qualitative approach.

4.3.2 Case Study

The aim of our study is to observe, study and investigate the software development process at the case company, to understand how they work together with plan driven RE and agile RE /user stories. We think our research can offer some new insights to the Company and others on working with RE and agile user stories. Therefore, case study method was found most well suited in our research as it is about observing and investigating a contemporary phenomenon.

We do not plan to affect the case company by making any changes in the studied company within the scope of this study. We only plan to observe and investigate the current working process involving plan driven RE and agile RE/ user stories. However, the results of our study could be used to bring about few changes. Thus, Action Research can be related to our case study and we would like to see this case study as part of a larger Action Research in future. Similarly, the results obtained from researching and exploring strategies to balance plan driven RE and agile RE in our work could be used to do design science in future.

Defining variables and conducting experiments requires excellent understanding of the domain. Since, in this study, we are still exploring the domain, it was not possible to run experiments in order to provide input to our research questions. However, future research could aim at experimenting with some of our identified strategies.

4.4 Research Setting

The research setting for our study is a premium automotive company based in Sweden. As previously mentioned in Chapter 1, the company works with a requirement model that accommodates aspects of both traditional and agile requirement engineering methods. For this reason, it was the perfect company to conduct case study for investigating our research questions.

4.5 Research Participants

For the case study, we chose 11 participants as the sample size. The participants were selected by Lars Ljungberg, my contact person, who is also an employee at the company. Participants were selected based on the following criteria. The first criteria was that the interviewees should belong to different roles in the software development cycle .i.e. different phases of the V model. The second criterion was that the chosen interviewees should be able to accurately answer all the interview questions listed in the interview guide.

4.6 Data Collection

The data was collected solely through interviews conducted at the case company. For this study, we included, semi-structured interviews with open-ended questions. According to Runeson and Höst [36], in a semi-structured interview, the planned interview questions were not necessarily asked in the order in which they were listed, but should be followed based on the development of the conversation between the interviewer and the interviewees. For a fully structured interview, interview questions are prepared beforehand and are asked in the order of listing [36]. Since we didn’t wish to ask the prepared interview questions in any particular order, we chose to conduct semi structured interviews over unstructured and fully structured interviews.

4.6.1 Interview Question Content

The interview questions try to cover all aspects of requirements engineering like elicitation, documentation, prioritization, updating, communication, validation etc. Questions covering aspects of agile methodologies like agile RE/ user stories were also included. In addition, there were questions related to software development lifecycle process and tools as well. All interviewers were asked almost the same questions. However, for interviewees involved in working with agile methodologies, the questions were slightly altered. For example, RE was replaced with user stories, and other small changes were made to the questions.

Initially, we had a lot of interview questions. To shorten the interview guide, we decided to strike off few questions of slightly lower priority. This was to ensure that we finished the interview in the allotted time slot i.e. one hour. The most significant and relevant questions were retained while the less important ones were taken off the list.

The interview guide begins with open questions on the interviewee background and knowledge about the software development process at their division. To address the research questions, the guide further proceed with questions about the participant’s understanding of requirements engineering and how it was conducted in their division.

Questions that followed next were related to realising what challenges and benefits the participants faced with current RE practices and tools and if they could offer suggestions for overcoming the same. The guide also contains questions related to how the division worked together with requirements and user stories and, benefits and challenges of doing the same. Later, the guide proceeds with questions about realising participant's thoughts on user stories taking over plan driven RE. All our interviewees were asked this question in the end of the interviews*: Is there something you would like to add to what you spoke so far in the interview? Something you forgot to mention?* This last question was included in case our interviewees forgot to mention something or want to add certain information that could be critical to our study. The designed interview questions can be found in the Appendix A and B of the report.

4.6.2 Conducting the Interview

In accordance with Van Teijlingen and Hundley [43], before going ahead with the interviews, we decided to pilot test the interviews. Before starting off with the interviews at company, we did mock interviews with people who were well acquainted in this topic. The purpose of the mock interview was to realise if we had the right questions for the interview and to check if the interviews could be finished on time. My contact person, supervisor and I decided it was best to conduct the first interview in a friendly environment before going ahead with the rest of the interviews. The interviews were arranged by the contact person at the company. In the first week of May, we performed the first interview with Peter Gergely in a friendly environment. The interviewee is a design engineer and has been part of the team doing development for five years. He has recently also taken up the role of a scrum master for the software development team and has worked extensively with System Weaver for ten years now. In overall, he has 21 years of experience in working with the company for system development.

All the rest of the ten interviews were conducted in the month of May and June 2016. The interviews for the study were performed at the research setting. The interviews were planned and booked in advance through mails. As suggested by Jacob and Furgerson [44], scripts were used both during the beginning and closing of the interview. The script provided notes for introducing the interviewer, sharing important details about the study like research purpose, providing informed consent, alleviating concerns of anonymity and confidentiality, obtaining permission for recording the interview, and also for asking the interviewee about the possibility of interview follow ups in near future.

Since interviews are semi-structured, the designed interview questions were not asked in the order in which they are listed, but asked based on the development of the conversation between the interviewer and the interviewees. At the end of every interview, we summarised the important points and took feedback on the same from the interviewee to avoid any misunderstandings later.

As Runeson and Host [36] mentioned, all interviews were recorded and transcribed later for data analysis.

4.7 Data Analysis

We used the Thematic Analysis technique [56] to analyse the interview data iteratively. All the six steps of the Thematic Analysis technique used for data analysis is explained in the following:

4.7.1 Data Familiarization

After conducting each interview, we transcribed the interview and studied the content of the interview transcription. Additionally, after every interview, we also revisited and reviewed previous interview transcriptions to better understand the process in the case company.

4.7.2 Initial Code Generation

After reviewing every interview transcription, we went through the transcribed data once again to look for all those parts of the transcription that were related and relevant to our research questions. The identified related parts were given appropriate labels. In what follows, are our sub-codes: “Software Lifecycle model”, “*Requirement Flow”, “User Story Flow”, “Supporting tools”, “Requirement unawareness”, “too many User Stories” and “Requirement-User Story traceability problem”, ”Requirement Updating issue” and “Changing Requirements”.*

4.7.3 Looking for Themes

We went through all the initial codes and identified common themes among them. In what follows are our common themes: *“Narrow-V Model”, “Software Process for plan driven RE”, “Supporting tool for plan driven RE”,* “*Software Process for agile RE”,* and “*Challenges in Narrow-V model”.*

4.7.4 Review Themes

In this step, we went through all the earlier identified themes to ascertain if the themes needed modification or if there was a need to include new themes. We grouped some themes into more abstract themes. For example, *“Narrow-V model”, “Software Process for plan driven RE”, “Supporting tool for plan driven RE”,* “*Software Process for agile RE”* were grouped into *“Requirement model”.*

4.7.5 Define Themes

At this step, we tried to determine relation between the identified themes and our research questions. Based on our research questions, we placed the themes into two categories: *“A Requirement model for requirements”, and “Challenges existing in the Requirement model”.*

4.7.6 Stating Results

We present results in Chapter 5.

4.7.7 Research Validity

Shenton [57] describes four trustworthiness criteria to qualitative researchers namely credibility, dependability, transferability and confirmability; for performing quality academic research. To ensure validity aspects of our research, we used all of these four trustworthiness factors.

4.7.7.1 Research Credibility

We followed Shenton's guidelines to increase the credibility of our research. In what follows, we describe these strategies:

- We visited the company several times prior to interviews to get acquainted with some of the employees and also receive relevant documents briefing about the department, products and development process.

- With the help of our contact person at the company, we ensured that our interviewee subjects belonged to different roles in the software development cycle. We also enquired if the selected interviewees had good knowledge about the requirement engineering practices of the company.

- At the beginning of every interview, all interview subjects were informed that their voice would be recorded and the data obtained from them would be used anonymously. All the subjects were comfortable with recording the data and using it for the case study as the data would be anonymously used.

- We arranged for several meetings with the experts in this area for making improvements in the interview guide and also took suggestions on our approach of analyzing the collected data.

4.7.7.2 Research Transferability

Since the findings of a qualitative project are specific to a small number of particular environments and individuals, it is impossible to demonstrate that the findings and conclusions are applicable to other situations and populations [57]. Even if each case is unique, it is also an example within a broader group and, as a result, the prospect of transferability should not be immediately rejected [57]. So, if practitioners believe their situations to be similar to that described in the study, they may relate the findings to their own positions [57]. We have tried our best to provide elaborate description of the phenomenon under investigation as we believe this will allow future researchers to relate to our work and be able to use it in their research.

4.7.7.3 Research Dependability

To address research dependability, we have presented the processes within our case study in detail, which can be used by other researchers in future for repeating the work.

Also, we have extensively covered our research design steps, and data gathering which might enable the reader in evaluating to what extent correct research practices were followed.

4.7.7.4 Research Confirmability

Shenton [57] says that objectivity in science is associated with the use of instruments that are not dependent on human skill and perception. However, it is difficult to ensure real objectivity, as even the tests and questionnaires are designed by humans, therefore, the intrusion of the researcher’s biases is inevitable [57]. But we believe, our detailed research methodological description contributes to the research confirmability.

5 Research Findings

In this chapter, we present our findings for the two research questions mentioned in Section 4.1 of the report. Section 5.1 presents a new kind of software development process and also tools that allows plan driven and agile requirement engineering process to co-exist together. Section 5.2 presents challenges and discrepancies that exist when working together with plan driven RE and agile RE.

5.1 Requirement Model

In this section, we present our findings for the first research question:

> **RQ1:** How are plan driven RE and agile RE co-existing in the current way of working?

5.1.1 The Narrow-V model

The company follows the Narrow-V model. The Narrow-V model consists of the following phases on the left hand side: Design Function, Design Architecture, Design System, and Design Component. At the bottom of the model, software solution is developed. For each of these phases on the left hand side, there is a corresponding verification phase on the right hand side of the Narrow-V model as shown in Figure 4.

The general development processes within the company states the kind of deliveries that are expected at the end of each phase.

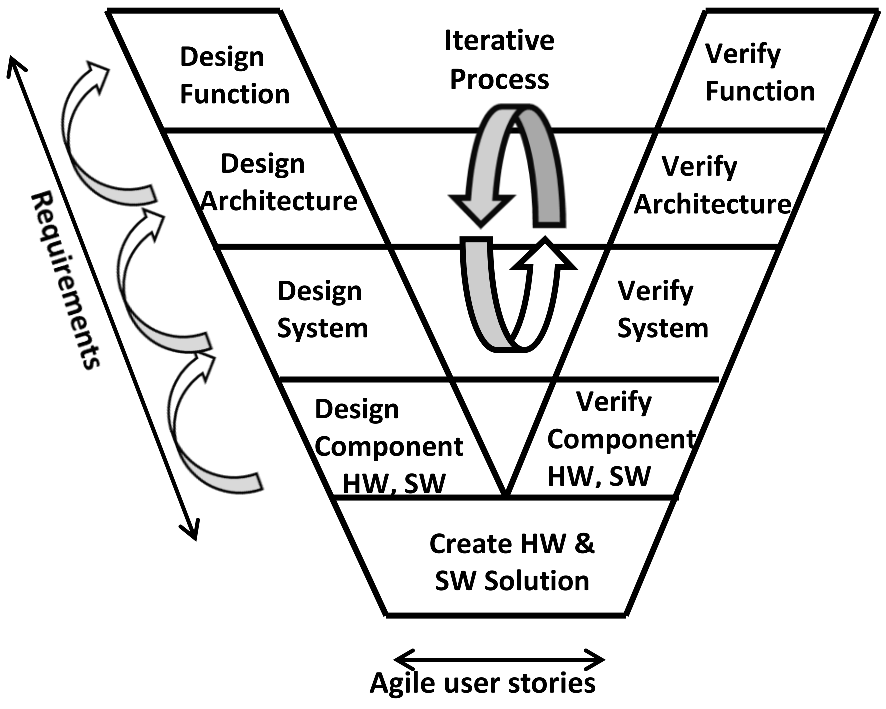

Figure 4: Narrow-V model

The Narrow-V model is quite different from the traditional V model. In the traditional V model, the traversal of information and data is one directional. Information flow starts from the top left hand side of the V model and flows all the way to the top right hand side of the V model. While, as Figure 4 illustrates, the Narrow-V model works with iterations. Traversal through this Narrow-V model happens not once but in several iterations. For example, in the first iteration, guess work on the possible properties or features of the product, the system solution and components are made. By the end of the first iteration, guesses are made. Then a second iteration is done. In the second iteration, the product properties, features, system solution and components solution become clearer. Then in the third stage iteration, there is more clarity in the product features that are going to be put in production along with the system solution for production and also on the requirements and solutions on the detailed components.

There are different waves of maturity in the information flow in this Narrow-V model. As an interviewee stated:

> *"The model is sort of abstraction to say that first you think about properties, then you think about what should the feature look like, how should I make this structure to get system design to realise this, then how should I create components to realise the system that I have specified and that goes down in the V model and the same pops up back again so that you test them on the same level."*

The Figure 5 illustrated below shows how the Narrow-V model accommodates both plan driven RE and agile RE., and allows them to coexist together, which will be discussed in detail in the ensuing sections. As it can be made out from Figure 4, on the left hand side, plan driven RE is followed. At the bottom of this model, i.e. implementation phase, agile software development process specifically scrum is practised meaning agile RE/ user stories are used in this phase here.

5.1.1.1 System Weaver

The Narrow-V model is supported by the System Weaver tool. The company has put in a lot of energy into using System Weaver tool. It is also referred to as information database or information platform. Information is put into it so that the necessary documents required by the process are generated. The documents are created by reusing as much of the detailed information as possible or by just adding new information where it is found that the old information is obsolete. There is a lot of reuse involved. As one of the interviewee subjects stated:

> *"We try to do it a way that we create a new structure for each generation we develop and within that generation we reuse the information leaves sort to say from the old generation. So they are same items or objects which are also linked into this new work. For each generation we have had so far, we always end up with little bit more information we had from the previous generation*."

So far, the company has made five generations of systems. The first one was put in production in 2006, while the latest was put in 2015. This means that within ten years, they have put five generations in the market. As an automotive department developing systems, the company has been doing this with three times the frequency when compared to other automotive companies.

5.1.2 Plan Driven RE

As it can be made out from Figure 5, on the left hand side, in the first phase, plan driven RE is followed with slight differences with respect to traditional RE practices. The Function Owner writes the high level requirements in the System Weaver tool. Information processes and requirements associated with the previous generation of the products are available in System Weaver. For instance, in an earlier version, initial features and requirements related to autonomous parking included automatic parallel parking only. In a later version, features and requirements related to perpendicular parking were added to the already existing features. Thus, in the later versions, it sufficed to add the requirements for the new features, as the base requirements were already in place. As one of the Function Owner's stated:

> *"I would say for the next projects, we use maybe 99.5 percent of the old requirements. The requirements are still applicable. You just need to do some additional work on them. System Weaver has all the old information. That's how the tool helps with requirement writing. Yeah, so you spend very less time on documentation."*

However, the Function Owner continuously reviews the already existing requirements in order to ensure their clarity and relevance. Sometimes, the existing requirements need to be updated as the old requirements may not be reusable. Nevertheless, in most cases, the existing requirements are valid and relevant, and can be reused.

After the Function Owner writes high level requirements for the function, these requirements move to the next level in the Narrow-V model, which corresponds to the Function Realization level as illustrated in Figure 5. The Function Realization determines what kind of sensor system needs to be used, and chooses the most appropriate sensor as per the vehicle's requirements stated in the requirement specification. Sometimes, the Function Realization works with the same requirements that come from the level above, while at other times, Function Realization may break down the requirements into more granular ones. While the Function Owner has an abstracted view of how the function should act, the Function Realization takes into account the vehicle architecture, the platform, how the different functionalities are distributed and then puts requirements on the different subsystems of the vehicle. Then at the subsystem level, requirements are further broken down. The different subsystems have one or several nodes comprising of hardware components with corresponding software components. Requirements are then distributed from the subsystem level to the nodes. Each node can have hardware requirements and software requirements.

As previously mentioned in Section 5.1.1, the company follows the Narrow-V model which is an iterative version of the traditional V model. For every iteration, at each level, feedback from the level below with respect to requirements is taken into account. This also applies to the requirements defined at each level as well.

The Function Realization reviews the function requirement specification document, and provides feedback to the Function Owner in case some requirements are difficult to understand, forgotten, irrelevant or not feasible. The Function Owner receives this sort of input not only from Function Realization, but also from other levels below the Function Realization in the Narrow-V model, and incorporates necessary changes to the high level requirements. It can be said that requirements are created as you move along the Narrow-V model. Similar process is followed with requirement formulation and requirement refining at each phase of Narrow-V model. In this way, the Narrow-V model allows requirements to be more flexible. Requirements are open to change and evolve in this iterative Narrow-V model. As one of the Function Owner's stated:

> *"Most of the prediction of the requirements you do in the beginning is always right. However as you go down the Narrow-V model, you realise it is not feasible to attain as per that requirement."*

Since there are inputs and suggestions on requirements going back and forth, communication remains high between the various phases of the Narrow-V model. Communication happens both formally through tools used for working with requirements, and also through informal means like discussions, meetings and emails. After the initial informal communication, updates are made to the requirements in System Weaver.

5.1.3 Agile RE/ user stories

As can be seen in Figure 5, at the bottom of the Narrow-V model, i.e., in the implementation phase, agile software development process, specifically scrum is practised implying that agile RE/ user stories are used. The Product Owner reads the requirement specification document in the System Weaver tool, which is written by the Function Owner, and writes corresponding user stories in the JIRA Tool. The JIRA tool is used by the agile team for writing user stories. All the details of user stories can be found in this tool. Once the stories are written in Jira, the Product Owner always discusses them with the team. If the Product Owner does not approve of the requirements found in System Weaver, he discusses with the Function Owner and gets the requirements changed.

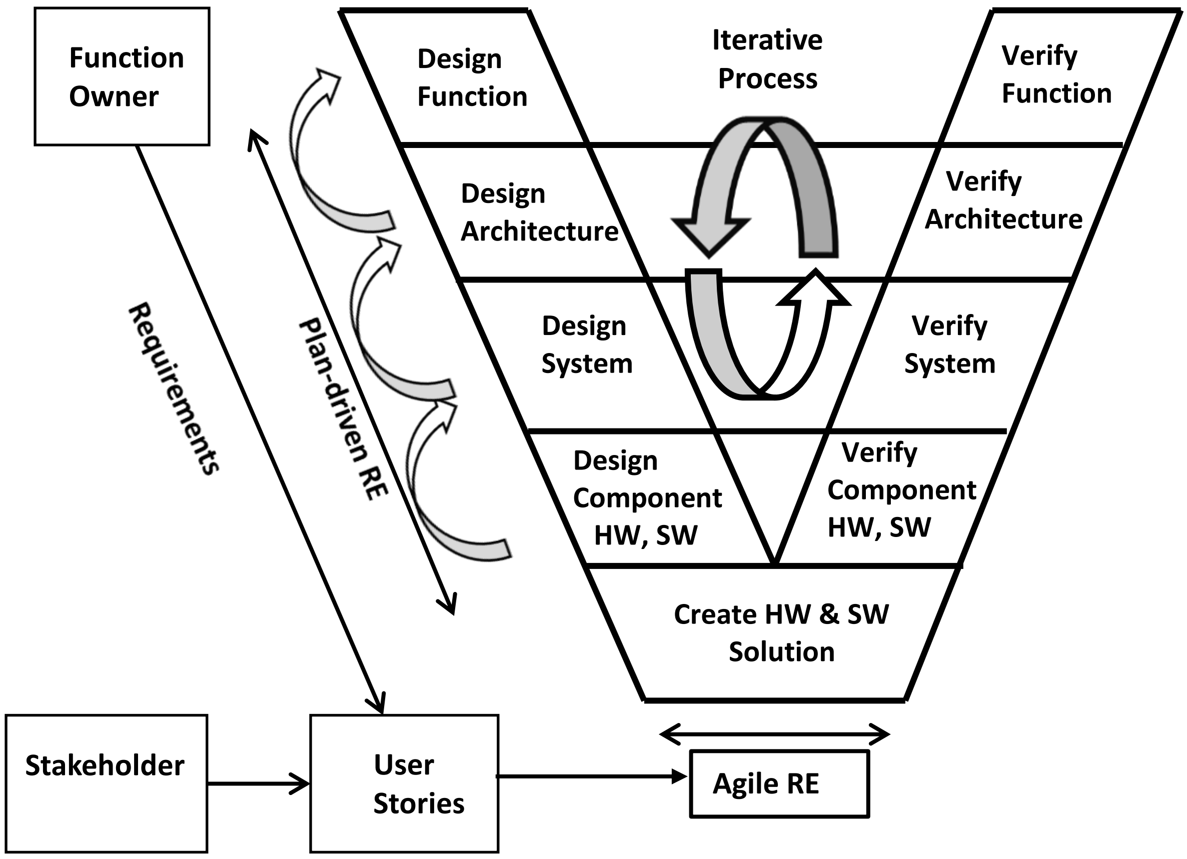

Figure 5: Requirements and user story coexistence

User stories originate not just from requirements but also from other sources. For instance, when agile team members discover aspects that need to be fixed, they write user stories into the product backlog in Jira. Product Owner also writes user stories based on tasks that he feels needs to be done. Also, the stakeholders (the Function Owner, integration team, Project leader, management team, members from other agile teams etc.,) approach the Product Owner or the team in order to formulate user stories. The Product Owner is responsible for managing the product backlog, and also for prioritising user stories. Usually, he takes input from those who have written stories, and also from the team, as the team usually has a good idea about what they should be doing next. Figure 5 illustrates the origin of user stories.

Basically, anybody can put user stories in the backlog. But, the stories need to be marked as a flag, and put at the bottom of the backlog. When the Product Owner sees the flag, he identifies it as a new user story, which is then inspected and prioritized. If any of the stakeholders, outside the team, wants the agile team to do something, they can write a user story, put it in the backlog, but cannot prioritise for the team. The stakeholders are asked to come to the weekly refinement meetings where they discuss about the user story to make sure it is well understood. As mentioned previously, whenever a story is added, a flag is added to it. After the team says they have understood the story, the flag is removed. In Jira, there is a flag for every story that is unclear. The user story is then refined and rewritten together. But the prioritisation of the user story happens afterwards. The team estimates the size of the user story based on the discussions that are had in the refinement meetings. The agile team does handshake with the external stake holders after they mutually agree on the scope in the refinement meetings. Handshakes are always done so that the Product Owner can go ahead and prioritise. For the user story to come into the sprint, the Product Owner also needs to do handshake with the team. The Product Owner can neither estimate the size of the task, nor can he decide what should go into the three week sprint. His main role is sorting and prioritising user stories. The task size is always estimated by the team.

Once the scope of the sprint is decided, it is locked. However, at the refinement meetings, though unusual, requests for changes in user stories in a sprint come up. If such a request pops up, it is always discussed with the team. The team decides if they can accommodate these changes into the current sprint. However, making changes to user stories in the middle of the sprint are not encouraged. Unless urgent, the scope of sprint once decided is locked.

In the sprints, the agile teams develop agile RE/ user stories. Product Backlog as well as user stories for the sprint can be found in the Jira tool. Even with all the user stories present in Jira, the tool is not used when working with user stories in the sprint. Jira is used only to look at user stories and other details in it. During the sprint, a scrum board is used. The scrum board is initially set up in the beginning of the sprint. The team doesn’t spend as much time in Jira during the sprint. Once the user stories are developed, the team writes unit tests to verify the code and to ensure that the user stories are met.

At the end of sprint, there are sprint review meetings, where the team informs the stake holders about who wrote the user stories, how they have managed the sprint, and what they have completed. At the sprint review meetings, the product that is delivered is talked about. The current progress of the product and what has been done so far is discussed. Then there are also retrospective meetings held at the end of the sprint which are attended only by the Product Owner, scrum master and team. Here the team talks about processes, how the work has been done, about tensions in the group, if there has been a shortage of resources etc.

The Agile teams rely a lot on informal communication. Everyone knows where everybody sits. So, they just walk over and talk to the concerned person as everyone knows who wrote what user story and ask specific questions.

5.2 Challenges in the Requirement Model

In this section, we present our results for the second research question:

> **RQ2:** What challenges exist when combining both plan driven and agile requirement engineering in systems engineering?

5.2.1 Agile user story writing and refinement

Since the department has adopted Narrow-V model with agile software development for software implementation, they work with both plan driven requirements, and agile software development. The agile software development teams in the organisation receive user stories from outside the agile teams as well. As mentioned previously, anybody can write user stories into the backlog. New incoming user stories are put at the bottom of the backlog and marked with a flag. New user stories are easily identified by the Product Owner because of the flag. All the stakeholders, outside of the agile team, can write user stories, into the bottom of the backlog. However they cannot prioritise user stories for the team. The team themselves do the prioritisation. The stakeholders Sare invited to the weekly refinement meetings where the user stories are discussed to make sure it is well understood. After the user stories become clearer and well understood during these refinement meetings, they are unflagged. Once a user story is unflagged, it is refined by the stakeholders and agile teams together. The user stories are then prioritised by the Product Owner.

Some of our interviewee subjects complained on how they are usually pressed for time making it difficult for them to attend these meetings. From the interviewee data, we gather that though a standard process is followed in the organisation for user stories after it arrives into the backlog, there are no proper channels for screening user stories before it hits the backlog as anybody is allowed to write user stories. This resulted in several unimportant or wasteful user stories creeping into the backlog. Discussing such irrelevant user stories during refinement meetings takes up a lot of unnecessary time.

Thus, measures should be incorporated to avoid such user stories from coming into the backlog. Proper channels should be maintained for writing user stories. In addition to maintaining proper channels, it is recommended to include representation from the architecture and system design teams in the refinement meetings. This is crucial as some of the newly formulated user stories can impose new structure/requirements on the architecture and the system. Therefore, it is necessary to understand the impact and the implications of the formulated user stories from a system and architecture point of view.

5.2.2 Requirements and User-Story Traceability

As mentioned previously, plan driven requirements are formulated by the Function Owner in System Weaver tool. The Product Owner writes user stories based on these plan driven requirements in the Jira tool. However, no traceability is maintained between requirements and agile user stories. Apart from the Product Owner, nobody is aware of which user stories originate from which requirement.

Due to the lack of traceability, there is lack of transparency in relation to the origin of user stories, which can affect the interworking between the agile team and the plan driven RE team. Traceability helps verify that the Product Owner has covered all the requirements defined by the Function Owner. Also, there is a possibility that user stories can result in new requirements that have not been covered by the Function Owner. Traceability covers this possibility.

We asked the interview subjects, from agile team, why the traceability was not maintained. All of them believed that documentation is not a feature of agile process, and instead of spending time on documentation, preferred to spend time for actual implementation. As an interviewee stated:

> *"I don't think traceability is not required or something like that. It's just that my focus hasn't been on documenting the function. I just focus on doing implementation and developing the function."*

The interviewee subjects also mentioned that it is unclear as to whether it was the Product Owner's or the Function Owner's responsibility to document this traceability. The interview subjects, from the agile team suggested that Function Owner could probably take care of the documentation aspect of requirement and user story traceability, as documentation is a key feature of plan driven requirement engineering.

5.2.3 Requirement Unawareness in Agile development team

From the agile team, only the Product Owner has knowledge about the plan driven requirements. The rest of the agile team is unaware about it. The team has access to these requirements in system weaver. However they don't look into these requirements as they believe there is no need to since they work directly only with agile user stories and not requirements.

The department has recently started integrating testers with the agile development team. Earlier the testing team was considered as a separate entity. But in order to have the product ready and verified by the delivery date, it was decided to try out introducing testers into the agile development team. The testers look at requirements, check if the requirements are testable, and then develop tests in relation to the requirements. The tests might check implementations that have been done in several user stories.

Some of the interviewee subjects were of the opinion that this trial process of fusion of developers and testers in a team will work better if the developers are aware of requirements to a certain extent.

For the testers and agile development team members to work in a team, it is important for the agile developers to remain aware of the requirements. If the agile development team members continue to stay uninterested in requirements, it may become difficult to attain a smooth working collaboration with testers, and this trial process of merging testers and developers into a team might not work as well as expected.

5.2.4 Regular Requirement Updates

As one moves along the left hand side of the iterative Narrow-V model, at each phase, requirements get broken down into more granular ones. As soon as the requirements are broken down, the corresponding test departments in the right hand side of Narrow-V model write test cases for these requirements. Later, after the product is implemented, the testers validate the product against these test cases.

Some of the interviewees from System testing stated that they do receive all the requirements from System Design, but then they don't get any information on whether the requirements were implemented or not. So, it is naturally assumed that for the product that comes in for testing, all requirements were implemented. However, during product testing, they discover that either many of the requirements haven't been implemented or the requirements don't match the implementation. As an interviewee from System Testing stated:

> "*If I have a requirement saying this thing should happen, when I test it, I find out that what is supposed to happen doesn't happen. So the requirement is not validated. And then I find out the requirement wasn't updated. So actually the implementation was correct but the requirement isn't matching the implementation. Then they go back and change the requirement to match the implementation. So, the major issue we have is that the requirement is not updated."*

Confusions arise during testing because system testers do not receive updated requirements. The initial requirements were written way before product implementation. For the actual implementation, agile software development process is followed where the team works with user stories. The agile user stories and plan driven requirements traceability is not maintained. Therefore, it is difficult to verify if user stories have covered all requirements. Also, due to the lack of traceability, it is tough to trace those requirements that were altered to write user stories. Furthermore, for the new user stories which arise during implementation, the corresponding requirements have not been written. All these issues arise due to the absence of traceability. And since traceability is lacking, requirement updating does not happen.

The interviewees felt that they should receive requirements that are actually implemented. They also feel that they should get different versions of the requirements so that they know when they can expect the requirements to be implemented and not waste time performing tests. Because as of now when they test a requirement and find that it has not been implemented, a JIRA issue to report an error is raised that is assigned to the implementation team. Later, the system testers receive information that the requirement isn't implemented and the requirement needs to be updated. So a lot of time is wasted. The interviewees believe that instead of changing the requirements to match the implementation in the end, if the agile implementation team could communicate updated requirements to the System Design team earlier, the System testers will have to neither face confusion nor have to waste time while validating the product against requirements. +

5.2.5 Challenging the Function Owner

As it has been stated earlier, the company follows the Narrow-V model, an iterative version of the traditional V model. This iterative model supports a well organised and structured software process while also providing flexibility and support for requirement changes.

The Function Owners are met with requests for requirement changes from other corresponding levels below in the left hand side of Narrow-V model. Based on the feedback, necessary requirement changes are incorporated by the Function Owner. However, some of the interviewees who work as Function Owner are of the opinion that though it is a good practice that many people could read the plan driven requirements and provide feedback, it could also be a disadvantage for the same reason. As stated by an interviewee who is a Function Owner:

> *”The more people look into requirements, the more they read them, the more iterations it will become. I would say it is a good thing that many people read the requirements. But then it also means there is going to be more opinions, comments and also more work. I think that is a disadvantage.”*

As requirements are written on a very high level, it is easy to read and compare by just looking at the requirement. Since it is easy for people to understand the requirement, it is also easy for people to have an opinion on requirements leading to many requests for changing requirements. This can be challenging to the Function Owner.

6 Discussion

In this chapter, we discuss the findings of our study. First, we will discuss the findings of the case study by relating them to the presented related literature work in Chapter 3. Specifically, on how individual challenges of traditional RE and agile RE are reduced by working with the presented requirement model (Figure 5). Later, we will discuss on the possible strategies to compensate the challenges in this requirement model (See Chapter 5, Section 5.2 for challenges).

6.1 Requirement Model to Minimise traditional RE and agile RE challenges

The related work reports several drawbacks of traditional RE such as extensive documentation, elicited requirements to be found as incomplete, inconsistent, and incorrect; issues with requirement re prioritisation, not involving customer feedback, rigid towards requirement changes etc. It also reports challenges brought upon by agile RE which include minimalistic documentation, customer unavailability, lack of proper budget and time estimation, incorrect requirement prioritisation, neglecting non-functional requirements, lack of requirement traceability, customer inability and agreement etc. However our results report that some of the major challenges of both requirement engineering forms are over come in this requirement model along with the help of supporting tools like System Weaver and Jira.

Coming to the high level requirements, they are written by the Function Owner in the System Weaver tool. System Weaver contains all the base requirements associated with the previous generation or version of the products. Therefore requirements needed to be added only for the new features. Most of the times, existing requirements are valid and relevant, and can be reused. However, sometimes the old existing requirements may not be reusable and needs to be updated. However, in most cases, the existing requirements are valid. Since 99.5% of requirements are mostly reused, documentation work is very less. The Function Owner only documents the requirements and does not do prioritisation of requirements. Requirements are later broken down into user stories by Product Owner and later prioritised.

The company follows the Narrow-V model which as mentioned previously is an iterative version of the traditional V model. In this requirement model, for every iteration, at each level, feedback from the level below with respect to plan driven requirements is taken into account. Requirement feedback usually comes in if the requirement is difficult to understand, irrelevant, missed or if not feasible. This also applies to the requirements defined at each level as well. The Function Owner receives this sort of input from other levels below in the Narrow-V model, and makes necessary changes to the high level plan driven requirements. Due to the iterative nature of the requirement model, it supports customer involvement, getting feedback on requirements and is open to requirement changes. Because of this, flow of feedback, there is active communication between the various phases of the Narrow-V model.

Incorrect user story prioritisation which is usually a major problem with agile RE, is avoided in this case. This is because, the agile team hold refinement meetings along with stakeholders to discuss and prioritise user stories every week. With such meetings happening every week, there is hardly much scope for incorrect prioritisation. The weekly meetings also ensure that all stakeholders who have written user stories are present, which solves the customer unavailability problem in agile RE. Also, with a requirement model supporting plan driven RE and agile RE to coexist together, certain drawbacks of agile RE are negated. For example, in agile teams, work is done in sprints. There is only short term planning involved in agile style of working. No long term estimates are made for budget, time, features or work. But when working in this requirement model, initial long term planning and estimation which is led by the Function Owner is done first, before moving to agile software implementation where short term planning for the sprints are made.

Regarding traceability, all the agile user stories are documented and traceable in the JIRA tool. However, documentation of traceability of user stories from requirements is not covered. We will discuss this in detail in the subsequent following Section 6.2.

6.2 Challenges in the New Requirement Model

A focus group discussion was conducted in the company which was attended by faculty, PhD student, interviewees and employees at the case company. The aim of conducting the focus group was to validate the challenges we found in our research and get collective opinions from the interviewees and employees on the possible strategies to overcome these challenges. The case company has now very recently begun practising these strategies to work against the challenges.

In this section, we throw light on the insights for handling RE challenges in the requirement model. The summary of requirement model challenges and solutions can be found in Table 1.

6.2.1 Agile user story writing and refinement

All can write user stories into the backlog. While this is good in terms of giving freedom in writing user stories, it could result in too many user stories accumulating in the backlog making it a hard task to keep an overview on them. Perhaps, there should be some kind of proper channels for user story writing. This can be done by giving authority to limited concerned people to write user stories.

6.2.2 Requirements and User-Story Traceability

User stories derive from two sources, from high level requirements written by Function Owner and from user stories from external sources. Tracing user story to requirement shouldn’t be very difficult. Once the responsibility of who needs to document traceability is determined, this problem can be resolved.

However, for user stories originating from other stakeholders (See Figure 5), it is hard to identify which user stories relate to requirements. Also, it can be challenging to trace them back to requirements in case they are related to requirements. However, it will prove useful to do this traceability, as new user stories can result in new requirements. One way to tackle the above problem could be, while writing the user story, the writer can make explicit whether the user story is relevant for tracing. Again this suggestion can be challenging if the user story writers are not aware of requirements.

6.2.3 Requirement Unawareness in Agile development team

Fusing testers with agile team will work out successfully only if agile team members are as aware of requirements as the testers are. One potential way to address this issue is to force agile teams to create formal test report for every important release so that they know how requirements are tested. This will force the team to read up on requirements which will also create clear delivery with proper compliance to requirements.

6.2.4 Irregular Requirement Updates

System testers do not receive updated requirements. They work with the initial requirements which were written way before product implementation. Therefore, confusions happen during testing. For the software implementation, agile software development process is followed where the team works with user stories. User stories are implemented here. Owing to traceability issues between requirements and user stories, mentioned in Section 6.2.2, requirements don't get updated.

Additionally, too many user stories could deteriorate requirements quality (since much effort is not put into maintaining the requirements). One way to handle this problem could be that down to a certain level of requirements, all requirements have to be treated as ASIL-D (max. safety criticality requirements) with respect to tracing from requirement to test. A compromise could be made on agreeing on certain levels of requirements to which complete tracing and test reporting should be provided while allowing the other lower level requirements to fall away and get replaced by user stories within the agile way of working.

Fixing traceability between requirements and user stories should be the first step in solving requirement updating issues. Also, if requirements are updated based on main learnings from the sprint, it can help to document some of the tacit knowledge accumulated in the agile teams.

6.2.5 Challenging the Function Owner

The iterative nature of Narrow-V model provides flexibility and support for requirement changes. Based on the feedback received from corresponding levels below in the left hand side of Narrow-V model, necessary requirement changes are incorporated by the Function Owner. Because high level requirements are easy to understand, readers generally have many opinions about it which makes the Function Owner get overexposed to change requests on requirements. Sometimes, it also happens that people offer suggestions to Function Owner without even reading the requirement specification first.

One way to deal with requirement changes could be to make sure that only those who have read the requirements carefully are allowed to offer suggestions on requirements. Requirements should be allowed to change or get updated based on main learnings from the sprint. This can help to improve performance requirements with arbitrary performance goals. Also, introducing some kind of semi-formal process like filling a form with information like reasons for requirement change, can reduce unnecessary requirement change requests.

Table 1: Summary of solutions for challenges in the requirement model.

	Requirement Model Challenges	**Possible Solutions**
1	Agile user story writing	-Give authority to limited people.
2	Requirements-User Story Traceability	
	- User stories by Product Owner written based on requirements.	-Assign responsibility/ decide who will document traceability.
	- External user stories	- User story writer make explicit whether the user story is relevant for tracing.
3	Requirement awareness in agile team	- Agile team to create formal test reports for all important releases.
4	Irregular requirement updates	-Improve traceability. -Down to a certain level of requirements, consider them as safety critical requirements with respect to tracing from requirement to test. The Lower level requirements allowed to get replaced by user stories. -Update requirements based on main learnings from sprint.
5	Challenging Function Owner	- Only readers of requirement specification document to offer suggestions on requirement change to Function Owner. -Introduce semi-formal processes ex: like filling a form to enter reason for requirement change.

7 Conclusion and Future Work

From literature work, we gathered both traditional RE and agile RE have their own set of disadvantages. This motivated us to research on new ways of working that incorporates the best of both traditional RE and agile RE methods while lessening the challenges when working with requirements. For our research purpose, we conducted a case study at a premium automotive company based in Sweden. In this study, we contribute to a holistic view of the company's requirement model that allows to work with traditional requirements and agile user stories together while alleviating their standalone challenges. We also present knowledge about challenges in following this requirement model at the case company. Semi-structured interviews were done on eleven participants for collecting data. Data analysis on this collected data using Braun and Clarke [56] provided us with results for our research. Later, we validated the results of our research by having a focus group discussion at the case company. The focus group discussion also helped to get insights into how to handle and resolve challenges to a large degree.

We have determined that there is a need for future research in this area as it's a very promising area in the area of requirement engineering and software engineering. First, the software industry practiced just the traditional requirement engineering practices. Then came the agile requirement engineering methods which is contrasting to the former in its approach. Next in line, should be a requirement model which is a mix of both practices. This will help the industry to overcome challenges it usually faces with respect to requirement engineering in general. Performing research on this field and in this direction will pave way for companies who are software focussed to experience better requirement handling, thus, improving their software process.

This study was conducted as a qualitative research with interviews as the method of data collection. It would be useful to follow-up this study by using any method of data collection in the studied company or in other software intensive automotive companies that work with a requirement model supporting traditional requirements and agile user stories to coexist together. Researchers can perform similar studies at various appropriate research settings to identify the different ways both requirement engineering forms can coexist together and explore to what extent the challenges of traditional requirements and agile user stories can be overcome with this kind of coexistence. Furthermore, they can investigate other accompanying challenges associated, when traditional RE and agile RE strive together. This study can also be replicated in other relevant research settings and need not be restricted only to the automotive domain. Researchers may also use the data and results of our study to perform action research, design science and experiments on the same in future.

Bibliography

[1] U. Eliasson, R. Heldal, E. Knauss, P. Pelliccione, “The need of complementing plan-Driven requirements engineering with emerging communication: experiences from volvo car group,” in 23rd IEEE International Requirements Engineering Conference. IEEE, 2015.

[2] I. Inayat, S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, “A systematic literature review on agile requirements engineering practices and challenges”. Computers in Human Behaviour, 2014.

[3] I. Inayat, L. Moraes, M. Daneva, and S. Salim, “A reflection on agile requirements Engineering: solutions brought and challenges posed,” in Proceedings of Scientific Workshop of the XP2015. ACM, 2015

[4] M. Daneva, E. Van Der Veen, C. Amrit, S Ghaisas, K. Sikkel, R. Kumar, N. Ajmeri, U. Ramteerthkar and R. Wieringa, “Agile requirements prioritization in large-scale outsourced system projects: An empirical study,” Journal of systems and software, vol. 86, no. 5, 2013.

[5] V. Shukla, D. Pandey, and R. Shree, “Requirement Engineering, A survey,” Requirements Engineering, vol.3, no.5, 2015.

[6] K. Pohl, “The three dimensions of requirements engineering," in Seminal Contributions to Information Systems Engineering. Springer, 2013, pp. 63-80.

[7] K. Pohl, Requirements engineering: fundamentals, principles, and techniques. Springer Publishing Company, Incorporated, 2010.

[8] I. Sommerville and P. Sawyer, Requirements Engineering: A Good Practice Guide. Wiley, 1997.

[11] S. Lauesen, Software requirements: styles and techniques. Pearson Education, 2002.

[12] F. Paetsch, A. Eberlein, and F. Maurer, “Requirements engineering and agile software development," in 2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. IEEE Computer Society, 2003, pp. 308-308.

[13] D. Pandey, U. Suman, and A. Ramani, “An effective requirement engineering process model for software development and requirements management," in Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 International Conference. IEEE, 2010, pp. 287-291.

[14] E. Bjarnason, K. Wnuk, B. Regnell , “Requirements are slipping through the gap- A case study on causes and effects of communication gaps in large-scale software development,” in 19th IEEE International Requirements Engineering Conference. IEEE Computer Society, 2011, pp. 37–46.

[15] D. M. Berry, K. Czarnecki, M. Antkiewicz, M. AbdElRazik, “Requirements Determination is Unstoppable: An Experience Report,” in 18th IEEE International Requirements Engineering Conference. IEEE, 2010, pp. 311-311.

[16] S. Lauesen, Software Requirements: Styles and Techniques. Addison-Wesley, 2002.

[17] H. Berlack, Software configuration management. John Wiley & Sons, 1992.

[18] K. Beck, A. Cockburn, R. Jeffries, and J. Highsmith. “The Agile Manifesto,” Software Development, vol. 9, no. 8, pp. 28–32, 2001.

[19] P. Abrahamsson, Outi Salo, J. Ronkainen and J. Warsta, Agile Software Development Methods: Review and Analysis. VTT publications, pp.17-36, 2002.

[20] D. Cohen, M. Lindvall, and P. Costa, "Agile software development," *DACS SOAR Report,* pp. 1-15, 2003.

[21] D. Leffingwell, Scaling Software Agility: Best Practices for Large Enterprises, Addison-Wesley Professional, 2011.

[22] A. Cockburn, and J. Highsmith, "Agile Software Development: The Business of Innovation," in the Proceedings of the International Conference on Software Engineering. IEEE, 2001, pp. 120-122.

[23] K. Schwaber, M. Beedle, Agile Software Development with SCRUM. Prentice-Hall, 2002.

[24] IEEE Standard 830 (1998) IEEE recommended practice for software requirements.

[25]. IEEE Standard 1233 (1998) IEEE guide for developing system requirements specifications.

[26] A. Sillitti and G. Succi, Engineering and Managing Software Requirements. Springer Berlin Heidelberg, 2005.

[27] K. Beck, A. Cockburn, R. Jeffries, and J. Highsmith. “The Agile Manifesto,” Software Development, vol. 9, no. 8, pp. 28–32, 2001.

[28] F. Paetsch, A. Eberlein, F. Maurer, “Requirements engineering and agile software development”, in Proceedings of the 12th IEEE International Workshops on Enabling Technologies. IEEE, 2003.

[29] A. Eberlein and J.C.S.D.P. Leite, “Agile Requirements Definition. A view from Requirements Engineering,” in Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE'02), 2002.

[30] Cysneiros, L. Marcio, ”Requirements Engineering in Health Care Domain,” in Proceedings of IEEE Joint International Conference on Requirements Engineering. IEEE, 2002.

[31] M. Crotty, The foundations of social research: Meaning and perspective in the research process. Sage, 1998.

[32] J. W. Creswell, Research design: Qualitative, quantitative, and mixed methods approaches. Sage publications, 2014.

[33] D. C. Phillips and N. C. Burbules, Postpositivism and educational research. Rowman & Littlefield, 2000.

[34] S. Kemmis, M. Wilkinson, “Participatory action research and the study of practice,” Action research in practice: Partnerships for social justice in education, Routledge, 1998, pp. 21-36.

[35] C. H. Cherryholmes, “Notes on pragmatism and scientific realism,” Educational Researcher, vol. 21, no. 6, pp. 13-17, 1992.

[36] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," Empirical Software Engineering, vol. 14, no.2, pp. 131-164, 2009.

[37] C. Robson, “Real world research. 2nd," Edition. Blackwell Publishing. Malden, 2002.

[38] R. K. Yin, Case study research: Design and methods. Sage publications, 2013.

[39] I. Benbasat, D. K. Goldstein, and M. Mead, “The case research strategy in studies of information systems," MIS quarterly, pp. 369-386, 1987.

[40] D. Avison, R. Baskerville, M. Myers, “Controlling action research projects,” Information Technology & People, vol. 14, Issue 1, pp. 28-45, 2001.

[41] R. Rapoport, “Three dilemmas of action research,” Human Relations, vol. 23, no. 6, pp. 499-513, 1970.

[42] C. Wohlin, M. Höst, M. C. Ohlsson, B. Regnell, Runeson P, A. Wesslén. Experimentation in software engineering, Springer-Verlag Berlin Heidelberg, 2000.

[43] E. V. Teijlingen and V. Hundley, "The importance of pilot studies," *Social research update,* pp. 1-4, 2001.

[44] S. A. Jacob and S. P. Furgerson, “Writing interview protocols and conducting interviews: Tips for students new to the field of qualitative research," The Qualitative Report, vol. 17, no. 42, pp. 1-10, 2012.

[45] Schwaber, K., and Beedle, M., Agile Software Development with Scrum, 2001, Prentice Hall.

[46] D. Firesmith, “Common requirements problems, their negative consequences, and the industry best practices to help solve them,” Journal of Object Technology, vol. 6, no.1, 2007.

[47] D. E. Damian, D. Zowghi, “An insight into the interplay between culture, conflict and distance in globally distributed requirements negotiations,” In Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS). IEEE Computer Society, 2003.

[48] N. Nahar, P. Wora, S. Kumaresh, “Managing requirement elicitation issues using step-wise refinement model,” International Journal of Advanced Studies in Computers, Science and Engineering, vol. 2, issue 5, 2013.

[49] T. Shah, S. V. Patel, “A review of requirement engineering issues and challenges in various software development methods,” International Journal of Computer Applications, vol. 99, no.15, 2014.

[50] M. Kumar, M. Shukla, and S. Agarwal, ”A hybrid approach of requirement engineering in agile software development,” In Proceedings of the International Conference on Machine Intelligence and Research Advancement. IEEE, 2013.

[51] E. Bjarnason, K. Wnuk, and B. Regnell, ”Requirements are slipping through the gaps-a case study on causes & effects of communication gaps in large-scale software development,” In Proceedings of the 19th International Requirements Engineering Conference. ACM, IEEE, 2011.

[52] K. Boness, and R. Harrison, “Goal sketching: towards agile requirements engineering,” In Proceedings of the International Conference on Software Engineering Advances. IEEE, 2007.

[53] L. Cao, and B. Ramesh, “Agile Requirements Engineering Practices: An Empirical Study.” IEEE Software, 2008, pp 60–67.

[54] W.M Farid, and F. J Mitropoulos, “Novel lightweight engineering artifacts for modeling non-functional requirements in agile processes,” In the Proceedings of IEEE Southeastcon, 2012.

[55] T. J. Bang, “An agile approach to requirement specification,” in Agile Processes in Software Engineering and Extreme Programming, Springer Berlin Heidelberg, 2007, pp. 193–197.

[56] V. Braun and V. Clarke, “Using thematic analysis in psychology," Qualitative research in psychology, vol. 3, no. 2, pp. 77-101, 2006.

[57] A. K. Shenton, “Strategies for ensuring trustworthiness in qualitative research projects," *Education for information*, vol. 22, no. 2, pp. 63-75, 2004.

[58] M. Broy, “Challenges in automotive software engineering," in Proceedings of the 28th international conference on Software engineering. ACM, 2006, pp. 33-42.

[59] P. Braun, M. Broy, F. Houdek, M. Kirchmayr, M. Muller, B. Penzenstadler, K. Pohl, and T. Weyer, “Guiding requirements engineering for software-intensive embedded systems in the automotive industry," Computer Science-Research and Development, vol. 29, no.1, pp. 21-43, 2014.

[60] S. Balaji, and M. Sundararajan Murugaiyan, "Waterfall vs. V-Model vs. Agile: A comparative study on SDLC." International Journal of Information Technology and Business Management, vol. 2, no. 1, pp. 26-30, 2012.

[61] I. Jacobson, G. Booch, and J. Rumbaugh, The unified software development process. Addison-Wesley, 1999.

[62] R. Andreas, C. Bartelt, T. Ternité, and M. Kuhrmann, "The V-Modell XT Applied–Model-Driven and Document-Centric Development," in 3rd World Congress for Software Quality, 2005.

Appendix – Interview related to plan driven requirements

The interview instrument provided in the coming page was used for the interview round with interviewees involved with requirements usage at the case company. The questions provided in the guide cover aspects about background, organizational software development process, requirement process, requirement tools, and, balancing between plan driven and agile RE.

Interview Questions

Demographic Questions

1. What is your role in the company? Could you briefly describe your role?
2. How long have you been working in this role at the company?

Department Questions

3. Could you briefly describe what the department does?(very briefly)

Development Process Questions

4. What development process do you follow?

Requirement Engineering

5. Based on the concepts described in the previous question, how do you think the process of RE happens?
   a. Where do requirements come from? What form?
   b. What do you do with the requirements received? Refine / break them down?
   c. Do you document the new requirements? What about prioritisation?
   d. How is abstraction of requirements done? How are abstraction levels determined?
   e. What is the next step in the requirement flow? Where/ to whom requirements go?
   f. Do you reuse requirements? How?
   g. How do you manage changes in requirements?
   h. How do you manage unclear requirements? How is it clarified? Make assumptions?
   i. How is requirement specification used throughout the process? Is it written from the scratch?
   j. How are requirements validated? Are requirements traceable? How is traceability ensured?
   k. How much of informal communication do you rely on?
   l. What challenges/ benefits do you encounter with the current requirement process? How do you think the challenges could be overcome?

Connections between V model and agile team

6. How are requirements and user stories linked in your case?
7. How are requirements and user stories coexisting together?
8. What benefits and challenges exist while working together with requirements and user stories?

Tools

9. What tools do you use to facilitate requirements?
   a. Does the adapted tool address everything you think is important to facilitate requirement process? What will they be?
   b. Do you think the tool fails to address few issues? What are they?

Forgot to add something?

10. Is there something you would like to add to what you spoke so far in the interview? Something you forgot to mention?

Appendix – Interview related to agile requirements

The interview instrument provided in the coming page was used for the interview round with interviewees involved with agile RE/ user stories usage at the case company. The questions provided in the guide cover aspects about background, organizational software development process, agile RE/ user stories process, tools supporting agile user stories, and balancing between plan driven and agile RE.

Interview Questions

Demographic Questions

1. What is your role in the company? Could you briefly describe your role?
2. How long have you been working in this role at the company?

Department Questions

3. Could you briefly describe what the department does?(very briefly)

Development Process Questions

4. What development process do you follow?

Agile Questions

5. Based on the concepts described in the previous question, how do you think the process of agile user stories happens?
   a. What agile process do you follow?
   b. Where do user stories come from? From whom?
   c. What do you do with the user stories received? Refine / break them down?
   d. Do you document the new user stories?
   e. On what basis do you do prioritisation? When/who gets involved during prioritisation?
   f. What is the next step in the user stories flow? Where/ to whom user stories go to?
   g. What is the duration of the sprint? What happens at the end of the sprint?
   h. How are user stories validated? Are user stories traceable? How is traceability ensured?
   i. How do you manage changes in user stories?
   j. How do you manage unclear user stories? How is it clarified? Make assumptions?
   k. How do you handle non-functional requirements?
   l. Where/ How much of informal communication do you rely on?
   m. What challenges/ benefits do you encounter with the current agile process? How do you think the challenges could be overcome?

Connections between V model and agile team

6. How are requirements and user stories linked in your case?
7. How are requirements and user stories coexisting together?
8. What benefits and challenges exist while working together with requirements and user stories?

Tools

9. What tools do you use to facilitate user stories?
   a. Does the adapted tool address everything you think is important to facilitate agile user stories process? What will they be?
   b. Do you think the tool fails to address few issues? What are they?

Forgot to add something?

10. Is there something you would like to add to what you spoke so far in the interview? Something you forgot to mention?