Martin Steinkellner, BSc.

Quality of Requirements in Agile Software Development and its Effects on Reliability of Planning

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Felfernig, Alexander
Dipl.-Ing. Ingomar Wascher

Institute for Softwaretechnology
Head: Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Slany

Graz, September 2018

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Abstract

Requirement analysis and verification are necessary activities to obtain confidence that written requirements are specified in an appropriate way. The first expression of requirements has often been vague or informal. Therefore, it is an important task to find errors as soon as possible to reduce risk and project costs. Modern requirement management is nowadays supported by tools, that give stakeholders the possibility of the automatic inspection of defined requirements. The maintenance of the investigated requirements is managed at the company AVL with Atlassian Jira, a commercial requirement engineering tool. In this thesis, a practical approach is presented to perform requirement classification as a starting point with Atlassian Jira. Therefore, several indicators were implemented for AVL to analyse requirements for different projects. The results gathered from the data evaluation highlight that the quality of written requirements correlate with agile properties. Finally, additional enhancements are discussed to better support the requirement classification process.

Keywords: requirement engineering, scaled agile framework, agile software development, quality metric, atlassian jira

Contents

List of Figures

1 Introduction

The implementation of software products is concerned with the quality of software requirements. Poorly defined requirements are one reason for software project failures. Therefore, the creation and the definition of software requirements are important parts of a successful project [31].

Requirements are gathered, specified, analysed and validated in the requirements engineering process. The output are documented requirements, which are written in natural language [27].

Quality analysis is a main indicator for the success of software products. Low quality may result in introducing more errors in software artifacts and those errors are costly to correct. Creation of high quality requirements has recently been [27] recognized as an important phase in requirement engineering. The creation step consists of several activities, from the initial elicitation till the final validation. With regard to the product, a large amount of requirements could be necessary and many roles are involved in the project. Therefore, an automatic assignment of a quality value, which is based on specific standards, would support the whole engineering process.

Beside the traditional software process, agile software development practices become very popular for small and large organisations. One goal of agile methods is to reduce the overall process effort. It is more focused on the development aspect. The core elements of agile development are simplicity, regular customer involvement, incremental releases and frequently changing requirements [32].

1.1 Research questions

The main contribution of this thesis is to improve the quality of written requirements, managed by tools. In this context we investigate different aspects to answer the following research questions:

1. How should the quality of agile requirements be verified?
   This question implies that correctly specified agile requirements contribute to a higher final software product quality, an assumption that has been considered to hold for traditional requirements.
2. Is there any correlation between the quality of requirements in regard to committed and planned agile points?
   Estimation of agile points is done by the team. Statistical reports should show that high quality requirements are more in line with planned and committed points than low quality requirements.

The goal of this work is to recommend possible methods to measure the quality and monitor requirements. These methods shall support AVLs organization in the definition and analysis phase of requirement engineering. The final implementation aims to support the organization to start writing requirements of better quality.

1.2 Thesis outline

The content of this document is structured as follows. Chapter 2 starts with the definition of agile software development methods and describes on which values and principles it is based on. Sections 2.2, 2.3, 2.4, and 2.5 in this chapter discuss prominent agile methods with their main artifacts. Section 2.6 highlights the differences between two common methods, scaled agile framework (SAFe) and Scrum. The next Section 2.7 describes a derived version from SAFe, invented at AVL. It summarizes the main techniques used in this process. Section 2.8 highlights the main contributions to requirement analysis. The first two subsections cover main principles to perform the analysis task. The last subsection deals with advanced machine learning approaches to perform automatic requirement analysis. Section

2.9 emphasizes which terms and phrases should be avoided when creating requirements. Additionally, the description of a quality model to categorize requirements is presented. The last Section 3.5 discusses a tool, invented and developed at TU Graz. It is a requirement engineering tool with focus on automatic link detection and group decision techniques. Furthermore, a comparison with other commercial tools is carried out to highlight the biggest differences.

In Chapter 3, the concrete implementation and possible additional enhancements are discussed. Section 3.1 highlights the main lexical and syntactical indicators which have been implemented. Section 3.2 deals with quality metrics of written requirements and how they are calculated. In the next Section 3.3, the reporting gadgets are demonstrated and how they look like. The last Section 3.4 gives some insights in possible enhancements to support planning and analysis of requirements in AVLs environment.

In Chapter 4, the results of the thesis are presented. The data used for this evaluation are based on three different AVL projects. The data of these projects are stored with Atlassian Jira. Furthermore, the defined research questions are discussed, and the answers are highlighted.

The last Chapter (Chapter 5) concludes the work. A recommendation will be given concerning the indicators which should be used in the future. Additionally, further enhancements and possible extensions are discussed.

2 Related Work

The general goal of agile software development is to develop software with a high degree of stakeholder engagement, faster time to market due to short iterations, flexibility in terms of changing requirements, and more transparency for all stakeholders due to the integration of learning sessions in the project. The basis foundation of agile software development was defined in the Agile Manifesto. Agile tends to focus on creating simple code, regular testing, and providing functional demos as soon as they are applicable [5]. The focus lies on the development process itself and reduces the documentation effort. In this section, an overview of some important agile development methods will be presented [14].

2.1 Agile Manifesto

In 2001, 17 practitioners of Scrum, Extreme Programming, and other well known software development methods met to find a common ground. The result of this meeting was the "Manifesto for Agile Software Development". They proclaimed the need for iterative techniques in software development [5].

The manifesto is based on the following four values [5]:

- Individuals and interactions over processes and tools,
- Working software over comprehensive documentation,
- Customer collaboration over contract negotiation,
- Responding to change over following a plan,

The values on the left side are more important than the values on the right side. From those four priorities, twelve important principles were derived and summarized. Those principles form the basis to follow the agile vision [5]:

1. Satisfaction of the customer by continuous delivery of working software.
2. Welcome changing requirements, although implementation is in the middle or in the end of the software project.
3. Deliver working software more frequently with preference to a shorter time scale.
4. The development team and the business have to collaborate on a daily basis.
5. Create products around motivated individuals. Trust them and support their needs and give them an environment they need.
6. Face-to-face communication is the most effective method for information sharing.
7. The fundamental measure of progress is working software.
8. Agile methodologies support sustainable development. The stakeholders should be able to manage a constant pace indefinitely.
9. Technical excellence is achieved by continuous attention. Good design enhances agility.
10. Simplicity is based on the art of maximizing the volume of work not done.
11. Self-organized teams enable the best architectures, requirements and designs.
12. At frequent intervals, the team reflects on how to become more effective. Adjustment and improvement of its behaviour are necessary.

At first glance, the presented values are in direct conflict with traditional software engineering process and seem to support a more undisciplined approach. However, there is indeed a solid methodology behind agile processes. They depend on the harmony of the whole team and a high level engagement of discipline to follow and perform the rules and principles agreed upon. It has been shown that the combination of non-traditional methods and agile software development methods is effective, in particular for products and projects with a high level of uncertainty [9].

2.2 Extreme Programming

The agile method Extreme Programming (XP) was invented by Kent Beck in 1999 [14]. The technique is based on the following properties:

- simplicity,
- communication,
- feedback, and
- courage

To accomplish these properties, XP builds up on several practices. The main practices are short iteration cycles, compact releases, regular feedback, and on-site or close customer communication. Additionally, continuous refactoring, testing, integration, and pair programming have a huge impact on the success of XP.

The team in XP starts with a planning game. In the whole XP process, the term team does not only refer to the development team but also to the customer who is part of the team. In the planning game activity the members sit together and start to write User Story cards. A User Story card is a feature description provided by the customer.Hence, the customer must know what the expected system should do before the meeting starts.

A User Story card should have an overall implementation time between one and three weeks. Whenever the effort is higher, the feature has to be split up into smaller sub-tasks. For several reasons, not every requirement can be transformed immediately into a Story card. Therefore, spikes were invented, which are special types of Story cards used to eliminate risks and uncertainty.

Developers try to estimate the effort for implementation of each Story card. After the estimation process is done, the customer prioritizes the user cards based on their business priority and the developers on the possible risk. After that, the iteration phase starts. By definition, such iteration cycles should not take longer than two weeks. Features with the highest overall prioritization will be developed first in the upcoming iteration. The test driven development approach, which is often combined with XP, comes

into play. Developers are writing test cases before they start with concrete implementation. In general, these test cases may be used as a very explicit specification. When writing a test case the first time, it should fail on the system. Customers are supposed to write acceptance tests. Those test cases are then used to determine if the functionality, described by a Story card is met. After an iteration is finished, developers can install the particular software, and the customer has the possibility of reviewing the release version.

2.3 Kanban

Kanban is a method to distribute the work of a project across groups of people. It originates from automotive production where different groups in the manufacturing process depend on the work of others. It was invented by David J. Anderson in 2004 [2] when supporting a small-scale development team at Microsoft. It is a flow control procedure for pull-driven software development. Based on literature, Kanban follows the ”less is more” approach and does not define any roles at all. This does not imply that no project manager exists, but the role definitions are up to the project [21]. The core principles of this method are:

- Workflow visualization,
- Limit Work in Process,
- Flow management, and
- Improved collaboration

The method pushes development teams to visualize the workflow, limits work in progress (WIP) at each workflow level and measures cycle time. Cycle time in Kanban context is the average time to finish a single element. The Kanban board is an essential part of the method. It is a tool to visualize the work and the workflows. Each column of a Kanban board represents a status of work. The work items will be moved from left to right, starting from a work backlog.

When a requirement is completed by the development team, the work item will be moved from the current column to one on the right side to

show progress. Furthermore, its aim is to minimize the work in process limit. It is used to prevent bottlenecks in software development. These limits for each status were agreed before by the development team. It is not allowed to add items beyond the capacity. For example in a team consisting of five developers a limit of five in the development queue assures that every developer has at most one task to focus on.

The development team cannot push stories directly into the testing stage. It is only allowed to mark the items as done, and as long as the items are not pulled by the testers, they stay in the development stage and use the stages capacity. The benefit of organizing work in this way is to spot bottlenecks faster during the development process and react accordingly. Additionally, changing requirements are welcome to gain early feedback and to minimize the risk that important features will be implemented too late[2].

2.4 Scrum

Scrum is a framework with the goal to manage, develop, and deliver tasks in complex projects or products. The framework includes roles, artifacts, events, and specific rules behind. The core of Scrum is based on a small-scale team of people, which are remarkably flexible and adaptive.

Empirical process control is the main idea behind Scrum. It insists that know-how derived from experience and decision making is established on the basis of what is known. The main concepts in this empirical process control are transparency, inspection, and adaption. The discussion of the Scrum process is based on the Scrum Guide by [30].

Transparency

Transparency means that important features of the process must be visible to stakeholders. Therefore, a common standard has to be defined to share an coordinated understanding of work. An example would be that those people who perform the actual work and those who supervise share a common understanding when a task is done.

Inspection

Inspection is defined as frequent review of artifacts and support to find

undesirable deviations. Inspection should not hinder the work in progress, it should be performed by experienced stakeholders at every point in the development process.
Adaption
Adaption deals with finding aspects, which may result in an unacceptable product. Adjustments must be performed on the process or on the process material as soon as possible to minimize the risk of further inconsistency.

Inspection and adaption are combined in four formal actions within the framework [30]:

- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective

2.4.1 Scrum Team

The Scrum Team consists of three roles of responsibility, the Product Owner, the Scrum Master and the Development Team. The self-organized Development Team acts in a cross-functional way, which means that they decide how to manage their work best themself, rather than being directed by others, who are not part of the Scrum Team structure. In other words, cross-functional means that they have the know-how to perform their work without external dependencies to others, who are not included in the Scrum Team. The goal of the Scrum Team is to enhance productivity, flexibility, and creativity. The delivery of products is done iteratively and incrementally to maximize opportunities for regular feedback [30].

The Product Owner
Based on the results of the Development Team, the Product Owner is responsible for maximizing the value of the specific product. How this is achieved may differ widely across individuals and organizations. The Product Owner is the single point of contact of a project. He is responsible for the Product Backlog management, which includes [30]:

- Precisely expressed backlog items

- Prioritization of those items which are best to achieve the required objectives
- Transparency of the Product Backlog to ensure it is visible and understandable to all
- Ensuring that backlog items are understandable to the Development Team

The Product Owner represents the interests of a committee within the maintained Product Backlog, but is still the only one who is allowed to change items in the backlog. Additionally, the Development Team is engaged to work on this backlog and no one can force the team to work on a different set of requirements [30].

The Development Team
The Development Team consists of experts who perform the work to deliver a releasable Increment at the end of each Sprint. Only members of this team are allowed to create such an Increment. The efficiency and performance are optimized by the team size and the empowerment from the organization to manage their own work. The Development Team in Scrum should follow the following properties [30]:

- It is completely self-organized. No one has the authority to tell the Development Team how to apply Product Backlog items into Increments.
- It is cross-functional. They have the know-how as a team needed to create Increments
- In Scrum, no titles will be assigned to single team members, regardless of their responsibilities within the team
- Additionally, regardless of tasks such as testing, operations and so on, no sub-teams are defined
- Accountability applies to the complete team and not to individuals, although some team members may have special skills and focus areas

The size of the Development Team is also an important factor. The number of team members should not be under three, otherwise this would have the effect of less team interaction, and another consequence would be smaller productivity improvements. Another negative effect which merits highlighting is the possibility that skill constraints could be encountered during Sprint. This may lead to the problem that a potential Increment

could not be delivered. As opposed to a Development Team consisting of too few members, too large Teams would create too much complexity for a process to be effective. When talking about Development Team size, the role of the Product Owner and Scrum Master is not included [30].

The Scrum Master
The role of the Scrum Master is in general a supporting and promoting one in the given framework. He is responsible for supporting everyone in understanding the Scrum practices and theory. This person acts as a servant-leader to the Scrum Team and additionally supports people outside the framework to understand which actions are helpful to the Scrum Team and also which actions are not helpful. The overall goal is to support everyone to change their actions to maximize the value generated by the Scrum Team [30]. The Scrum Master provides services to three different entities [30]:

- The Product Owner
- The Development Team
- The Organization

The service to the Product Owner includes:

- Assuring that the Scrum Team comprehends the project domain, aims, and scope as good as possible
- Exploring methods for efficient Product Backlog management
- Supporting the Scrum Team to comprehend the need for concise requirement specification
- Support in maximizing the Product Backlog in order to maximize the overall value of the artefact at hand
- Providing helps in applying agility
- Organizing Scrum events

The service to the Development Team includes:

- Tutoring the team to achieve the self-organization and cross-functional processes required
- Supporting the team in generating high-value projects
- Removing obstacles to the teams progress
- Helping to apply Scrum in organizational environments where it is not correctly applied

- Organizing Scrum events

The service to the Organization includes:

- Supporting the organization for Scrum adaption and planning of its implementation
- Motivating changes to increase the efficiency of the Scrum Team
- Collaborating with other Scrum Masters to increase the effectiveness of Scrum in the organization

As mentioned before, the Scrum Master is in charge of arranging Scrum events. Such events are used in Scrum to minimize the demand for other meetings which are not defined within the framework, and also to create some type of regularity. Those events are time-boxed and each of them has a specified limit of duration. Except the Sprint event, this one may end if the goal is achieved. Additionally, every event, except the Sprint event, provides the opportunity to inspect and adapt something and to create transparency [30].

2.4.2 Scrum Events

The Sprint
The Sprint is a time limited event with a maximum duration of one month. It is often refered to as the heart of Scrum. The goal of a Sprint is to create a potentially releasable product. Sprints are recurring events, meaning that a new Sprint will be started when the previous Sprint ends. Included in a single Sprint is the Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective and the actual development work. When a Sprint has started, the following rules must be followed [30]:

- Requirement changes are not allowed which would expose the Sprint Goal
- The aligned quality aims do not decrease
- Re-negotiation and scope clarification are handled between the Product Owner and the Development Team

Similar to a project, a Sprint has a goal as well. Its goal consists of what has to be built, a design plan which will guide building it, the actual task, and the final Increment.

By definition, a Sprint duration should not take longer than one month. A longer duration may increase complexity and changes what has to be built. A certain predictability may be achieved by regular inspection and respective adaption every month.

Another important aspect is the Sprint cancellation. The Product Owner has the authority to cancel a Sprint before the time-box is over. In the case that the Sprint Goal becomes obsolete, the Sprint has to be cancelled. This might happen, for example, when the company vision changes, or the technology or market requirements change. Generally, cancellation should occur when it makes no longer sense to work on a project. Such a cancellation requires time and resources because additional Sprint Planning must be done to start the next Sprint. Cancel situations are unpleasant to the whole Scrum Team, but unfortunately they happen only rarely [30].

Sprint Planning
The Sprint Planning event is used to plan what should be done in the next Sprint. The entire Scrum Team works together to create this plan. By definition, for a one-month Sprint, the Sprint Planning should not take more than eight hours. The Scrum Master is responsible for the event to take place, and his responsibility also includes that all participants are aware of the purpose of this event, and that the duration of the event remains within the defined boundaries. The meeting should answer the following two questions [30]:

- What can be done in the upcoming Sprint?
- How will the selected work be done?

The Product Backlog, latest Increment, the planned capacity of the Development Team for the upcoming Sprint, and the last performance or velocity of the Development Team form the input off the Sprint Planning. The Product Owner discusses the objective of the next Sprint with the whole Scrum Team, and also which items from the Product Backlog are necessary to complete the Sprint Goal. The Development Team selects the number of items from

the Product Backlog because they can assess what could be achieved in the next Sprint.

The combination of the selected items from the Product Backlog and the arranged plan is called Sprint Backlog. In case that the Development Team concludes it has too little or too much work, the items from the Product Backlog may be renegotiated together with the Product Owner. If technical advice or domain guidance is needed, the team is allowed to invite consultants for support. At the end of the Sprint Planning, the Development Team should have the knowledge how to achieve the potential Increment [30].

Sprint Goal
The Sprint Goal is an objective which will be discussed in the Sprint Planning event. It is used as guidance for the Development Team to answer the question why they are implementing this Increment. To fulfill the Sprint Goal, the team develops the necessary technology and functionality and keeps always the goal in mind. Should the expected work be different from what was originally assumed, a renegotiation of the scope from the Sprint Backlog can be carried out in collaboration with the Product Owner [30].

Daily Scrum
The Daily Scrum or stand-up meeting is an inspect and adapt meeting for the Development Team and is limited to 15 minutes. This event will be performed every work day of the Sprint. In Scrum, the strategy of the Development Team is to plan the work for 24 hours in advance. This strategy is used to optimize the overall team collaboration and the total performance of the team by inspecting the previous work since the last meeting and conclude the next Sprint work. An important aspect of this meeting is that it should be held at the same place and at the same time to decrease complexity. A common example of how the discussion is done in a Daily Scrum are the following topics [30]:

- What have I achieved since the last Daily Scrum?
- What will I do today?
- Which issues do I currently face to meet the Sprint Goal?

The Scrum Master takes care that the Development Team attends the meeting. He is responsible for that the duration of 15 minutes is not exceeded.

In case that issues arise and the time frame is too short, discussions for adaption or replanning will be held directly afterwards. The benefit of the Daily Scrum event is that it eliminates the need for other meetings, it improves the overall communication in the team, highlights bottlenecks, and improves the total level of knowledge within the team [30].

Sprint Review
At the end of each single Sprint, a review session will be performed to inspect the resulting Increment and to adapt the Product Backlog if required. The entire Scrum Team and the involved stakeholders attend this meeting to discuss what has been achieved in the current Sprint.

Based on this event, the next steps will be discussed to optimize the value for the upcoming Increment. In particular, this is an informal meeting with the purpose of getting early feedback and promoting collaboration. The Scrum Master is again responsible for the event to take place, and he checks that it takes less than four hours. The Sprint Review includes the following[30]:

- The invitation to the event is given by the Product Owner and involves the Scrum Team and other key stakeholders
- The presentation of the ”Done” and not ”Done” Product Backlog items
- The development Team discusses which actions worked well in the Sprint and which issues have arisen and how they have been resolved
- The demonstration of the current Increment with a question and answer session about it
- The attendees of the event collaborate on the next tasks of the upcoming Sprint. Therefore, the output of this meeting is used as input for the Sprint Planning
- Review of resources (for example time, budget, capabilities) for the next potential release
- Review of potential changes in the market and prioritizing what is the most valuable feature to do next

The final result of this meeting is an adjusted Product Backlog which illustrates the possible items for the upcoming Sprint [30].

Sprint Retrospective
The Sprint Retrospective will be held after the Sprint Review and before

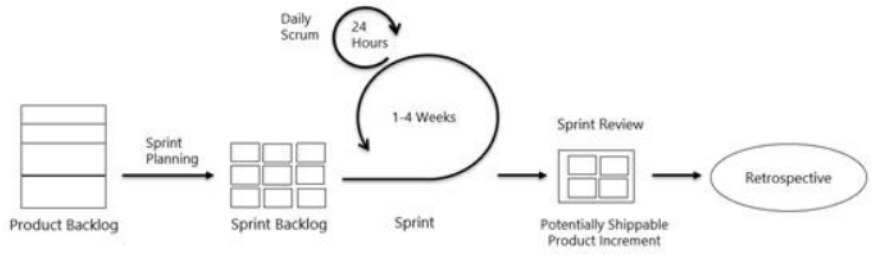

Figure 2.1: Scrum: Overview of the Scrum process [30].

the Sprint Planning event. It is used to investigate possible improvements for the upcoming Sprint. The Scrum Master is responsible for the event to take place, and for the duration, which must not exceed three hours. The objective of the meeting includes [30]:

- Inspection of the last Sprint in regard to people, communications, process and utility tools
- Highlighting the significant items which went well and order to determine potential enhancements
- Organizing a plan for realization of the potential enhancements

The Scrum Master attempts to motivate the Scrum Team to improve its performance to make the development process more effective. The Sprint Retrospective is used as a lesson learned session to improve the work process or to adjust the specification of "Done". At the end of the event, improvements should be collected by the Scrum Team, which should be realized in the upcoming Sprint [30].

2.5 The Scaled Agile Framework

In comparison to the previous agile methods, SAFe (scaled agile framework) is a framework for large organizations. It provides guidance for all stages of the enterprise domain, which are actively involved in software development. Companies such as Intel, Cisco and HP Enterprises have successfully

adapted SAFe to their development. SAFe distinguishes between three levels of responsibility, team level, program level, and portfolio level [16], [24].

2.5.1 Team Level

The lowest level of SAFe is the team level. In a sequence of iterations and releases, agile teams of about seven team members define, build, deploy, and test user stories. In the case of larger organizations or enterprises, agile teams form groups to work together. They have the possibility of enhancing and supporting the functionality of features, products, components and sub-components, although there are also further objectives which are not considered in this overview.

Literature often highlights that Scrum is used as the dominant agile method at the team level, but also other default agile practices like XP, Kanban or a mix of them can be used. This decision is up to the team and the final method must be adapted to their own environment [21], [24].

The backlog of user stories and their prioritization are responsibilities of the teams product owner. Ideally, this person is co-located with the team and contacts the team on a daily basis and contributes to its activities. The Features of this level will be managed by a Kanban program backlog [16], [24].

2.5.2 Program Level

At this level, large-scale system functionality is developed. It is accomplished by multiple teams in a synchronized Agile Release Train (ART). ART consists of frequent time-boxed iterations and milestones, which are quality- and date-fixed but scope is not fixed. The time boundaries are limited between 60 and 120 days and it produces releases or potentially shippable increments (PSI). The product manager is responsible for the definition of Features at this stage[16], [24].

2.5.3 Portfolio Level

This level is used to follow the investment priorities for the enterprise. In general, a mix of investment themes is discussed. This level should guarantee that the tasks being performed are the necessary tasks to realise on the chosen business strategy. The portfolio vision will be expressed in a sequence of scale-initiatives (Epics). Those Epics will be allocated to release trains by time. The approved Epic, which is required to create a portfolio solution, will be managed by the portfolio backlog [16], [24].

2.5.4 Agile Release Train

The Team and Program Level will be generalized by the Agile release train (ART). In large scale environments there could be multiple ARTs. It is used to aggregate the work of several teams. It uses also iterations and the last iteration is called Innovation and Planning. The teams in ART are basically cross-functional and consist of all persons who are needed to establish products from the beginning of an idea or concept, through the whole development process until deployment and release. The main principles of ART are[16], [24]:

- **Fixing the schedule:** Release dates are fixed. If a Feature misses a train, it can catch the next one.
- **System increment every two weeks:** Each increment will be delivered within two weeks. During a system demo, ART gives stakeholders a objective measure of progress. It is a critical event to gather feedback. The planning and presentation of a useful system demo need some work by the teams, but it is the only way to get feedback to guarantee to build the right solution.
- **Develop on cadence, release on demand:** cadence and synchronization techniques are used in the variability of development. Releasing itself is a decoupled element from the development cadence. ART is able to release a solution or part of solutions at any time. Independent to the business driver and release criteria.
- **Face-to-face planning:** Via face-to-face planning events, ART work will be planned periodically. This event serves as the core of the ART to

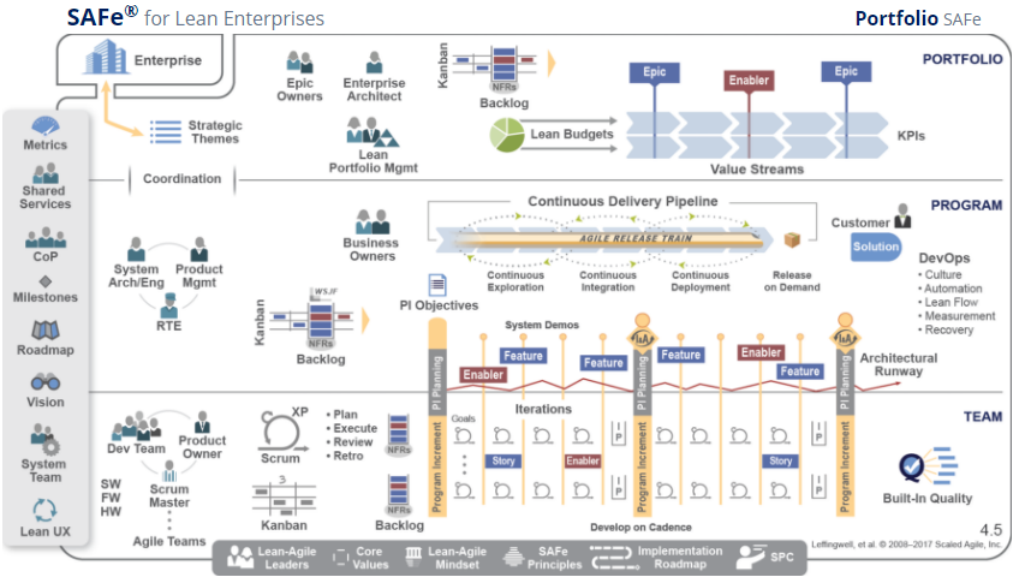

Figure 2.2: SAFe: Overview of the Program and Team Level. Multiple teams contribute to the Agile Release Train to guarantee releases in a frequent manner [16]

align the teams to a shared vision. It is suggested to organize a face-to-face event with a default agenda, which includes, for instance business vision or team planning breakouts. For geographically distributed organizations, that event should be streamed simultaneously with audio and video between all sites. It should take place in the last week of the last iteration and should last for about two days, to avoid affecting the capacity or scheduling of other iterations.

2.6 Differences between SAFe and Scrum

On a bottom line, both frameworks are based on agile principles and releasing in short iteration cycles. In contrast to Scrum, SAFe describes a framework and its structure for an agile methodology to work at a project on enterprise level. It is a successful technique to enhance agile at an enterprise scale and is applied in the whole enterprise and not only on team level. It is specialized for covering what Scrum can not.

Scrum is specialized on a small scale and focuses on smaller projects, while SAFe is used to scale agile and to fit larger organizations. It introduces Program and Portfolio Management together with Release Planning approaches. SAFe takes the collaborative and iterative nature of Scrum teams, adds to the principles of Lean thinking and provides a mechanism to scale those efforts through a number of practices such as aligning the teams around common value delivery.

SAFe does not rely only on SCRUM at the team level, other methods like Kanban, XP or a mesh of them can be used. To apply this framework, more engagement for the organizational management is required together with endurance, reassembling of roles and technical adjustments to solve complicated program issues. One of the biggest issues with traditional agile development is that many agile teams are employed on the same product, but the execution is performed independently and asynchronously. This results in several problems for releasing, integration of the full system or preparation of system demos.

The Agile Release Train procedure from SAFe overcomes this issue here in applying synchronization and cadence methods to encourage that the system is sprinting. The aim is to ensure a continuous evolution of the objective of the full system and not of single individual elements only (see Figure 2.3). At the end of an iteration, the system demo presents the evidence that the system is sprinting [24].

2.7 Alaska process

In Scaled Agile and especially in AVLs organizational environment several of the Scrum terms lost their meaning and were replaced by more fitting terms to avoid confusion to the process performers, even when this leads to some differences between ALASKA and Scrum terms.

The Scaled Agile Framework [24] only provides a frame for processes. Some processes have to be defined in an AVL context. SAFe also has its own

Table 2.1: The table highlights the terminological differences between Scrum and SAFe with the concrete reason for the differentiation

Scrum term	Alaska term	Reason
Scrum Master	Agile Master	ALASKA applies several lean and agile principles and methods. Agile Masters should be proficient in these methods and should also be able to support the teams with knowledge in Kanban, Lean Principles, XP Coding practices and so forth. The focus is not only on Scrum.
Product Owner	Development Owner	ALASKA does not put the ownership of a product at the team level. Often more than one team contributes to a product or one team contributes to several products. The development owner owns the team backlog and coordinates the requirements for the team, but does not own a product. This task belongs to the Product Management.
Product Backlog	Team Backlog	ALASKA combines product backlogs on program level into a Program Backlog. Program Backlog items are then assigned to teams and their Team Backlogs. Team Backlogs can contain backlog items for several products.

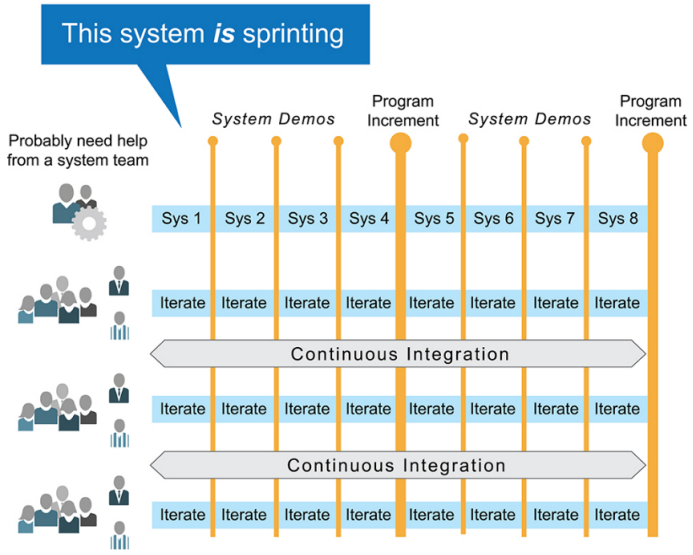

Figure 2.3: Agile Release Train: Aligned development to ensure the whole system is sprinting [24]

release cycle, which AVL will not follow. Therefore, Alaska is based on SAFe 3.0 and will only integrate new elements if they made sense in AVL context. Table 2.1 highlights the different terminology. The process has internally defined boundary conditions for the size of requirements to support managing implementation and fast delivery.

Additionally, an Epic is defined on the highest abstraction level and can further be split up into several Features for the program level. In the next stage, Features will be split up in fine-grained Stories on the team level [24].

- Epic: Epics should fit into one product release and into 2-3 program iterations. In case the Epic is larger, it should be split. Should an Epic take more than one iteration and involve more than one team, an incremental implementation approach should be planned.
- Feature: Feature should fit into one program iteration. It should be bigger than 13 Story points.
- Story: Story should fit into one sprint. It should not be bigger than 13 Story points. Preferably several stories should be completed within

one sprint.

Definition of Ready
Start to work on a requirement that is poorly understood can cause several issues for a team. For instance, a Feature without suitable information can lead to dysfunctionality, which could lead to the project taking a wrong direction or to delay. Therefore, Alaska has defined several formal criterions which have to be met before [**Safe01**].

Epic
Epics are created by Epic Owners with probable contributions by Product Management and System Architects. Before an Epic is handed over to the product management with the aim to be broken down into Features and subsequently integrated into the Program Backlog, it has to conform to the Definition of Ready. The Product Management of the affected products is responsible for checking and accepting the Definition of Ready. The following properties should be included in an Epic: [24]

- Lightweight business case completed.
- Technical solution concept for Epic (system view, product decomposition) created.
- Positioning in product or product line roadmap done.
- Rough estimation, based on historic data done, areas of high risk or uncertainty identified.
- Description is detailed enough for breakdown into Features.
- If the Epic involves several development teams or several products: Definition of participating teams and their responsibilities.
- If the Epic involves several development teams / several products: Epic Increments and integration milestones must have been defined.

Feature
Features are created by the Product Management, typically by a Content Manager, with possible contributions by the development owners and development teams. Before a Feature is accepted into the possible Feature set for the upcoming release planning, it must fulfil the Definition of Ready. The development owners of the affected teams are responsible for checking the Definition of Ready, the product management for fulfilling it. The following properties should be included in a Feature: [24]

- Assignment to a Program and (preliminary) to a team.
- Title: A short phrase, giving a name and some implied context to the Feature.
- Benefit: A short description which describes the benefit to the user and the business. There may be multiple benefits per Feature which are highlighted here.
- Feature Acceptance Criteria (functional).
- Non-functional requirements (NFRs).
- Estimation relative towards other new Features (normalized Story points).
- Small enough to be implemented within one Program Iteration.
- Ranked by Product Management.
- Clarification if Feature is tested by Development Team, System Team or both (for functionality and NFRs not already covered by tests of Stories); team(s) in charge of test have idea how to test and verify the Feature.
- The system test environment (test cases, system team, customer pilot environment) is defined for the Feature.

Story
Stories are created by the Development Owner with possible contributions by the development team. Before a Story is accepted out of the team backlog into the sprint backlog of the upcoming sprint (this happens typically in Sprint Planning Meeting) it has to conform to the Definition of Ready. The development team is responsible for checking the Definition of Ready, the development owner for fulfilling it [24].

- The Story title is written in user Story syntax.
- Precise acceptance criteria are defined and understood by each team member.
- All necessary non-functional requirements (NFRs) are documented and understood by each team member.
- Dependencies to other stories and teams are identified and documented.
- Story is estimated by development team according to ALASKA "normalized" Story points.

Name	Description	Detail	Mandatory	Default
Acceptance Criteria	Includes the required criteria of the Story implement procedure in order to be accepted	multiline text		
Affects Version/s	The version/s of the product which is affected by the Story	dropdownlist	Yes	
Description	Describes the Story realization procedure and limitations	multiline text	Yes	
Iteration (will be defined at RPM)	Definition of Program Iteration	e.g. 18.1 18.2 18.3		
Ready (manual)	Status flag indicating if the story is ready (red = not ready, yellow = undefined, green = ready)	green (equals "ready") yellow (not defined) red (equals "not ready)		red
Story Points	Number that shows the degree of implementation complexity due to required time and cost for Epic realization	Condition: 0 < Story Points <= 13	In Progress	

Figure 2.4: AVL Story Fields: Overview of important Story fields [16], [24].

- The development team has an idea how they will present the Story in the sprint review.
- The development team has an idea how they will test and verify the Story.
- The customer benefit and expectation is understood by the development team.
- It is clarified whether the Story is acceptance tested by the development team or if the system team is needed to support acceptance testing.

The listed fields in Figure 2.4 and 2.5, are currently used at AVL and available in Atlassian Jira. They are primarily used for Feature and Story requirements. There are many more fields available for a requirement in Jira, but those fields need more detailed inspection. Therefore, a readiness check shall be implemented to check if the fields are available, have a value and do not break any conditions (for example Story point size) [16], [24].

Name	Description	Detail	Mandatory	Default
Acceptance Criteria	Determines the required criteria of the Feature in order to be accepted	multiline text	Ready	
Affects Version/s	The version of the product which is affected by the Feature	dropdownlist	Yes	
Benefit	Defines the benefit of the Feature	multiline text	Ready	
Description	Describes the Feature realization procedure and limitation.	multiline text	Yes	
Iteration	Determines the time segment of the feature	e.g. 18.1 18.2 18.3	Ready	
Program	The Program that implements the Feature. In case the Feature is implemented by a team / person that is not belonging to a Program - this field represents the Program the product of the Feature belongs to.	APP, CED, DAT, TSI	Ready	
Story Points	Shows the degree of difficulty due to required time and cost for Feature realization in story points	Text	Ready	
Team	Name of the leading team responsible for feature realization	Text	Ready	

Figure 2.5: AVL Feature Fields: Overview of important Feature fields [16], [24].

2.8 Requirement Analysis

The primary goal of requirement analysis is to assure that the specification of requirements and possible models fit the fundamental quality standard. This is needed that specifications can be used productively to supervise further work. Beside analysis, validation has to be performed to assure that all listed requirements encourage the accomplishment of business value. Validation should also capture the stakeholders need and a fulfillment of its intention and objectives. To assure that requirements are described in a correct way, verification activities have to be performed. Several articles specify correctness. The IEEE-29148 standard has defined several properties to achieve correctness [15]:

- complete
- unambiguous
- specific
- time-bounded

- consistent

The IEEE standard focuses on traditional requirements which are specified before the design and implementation phase.The requirements will be reviewed by the authorized stakeholders and will be treated as a contract of what has to be implemented.

In contrast, agile or just-in-time requirements are small pieces of what has to be implemented. In comparison with the prior definition, agile requirements may be considered as incomplete, not specific and probably ambiguous when first defined. This results in the assumption that the notion of quality is different for traditional and agile requirements [15].

2.8.1 ISO/IEC/IEEE 29148

The international standard IEEE-29148 defines the mandatory content of a required information. It supports and gives guidance for the format of the needed information items. The goal of the construction is that well-formed stakeholder/system requirements shall be implemented. A well-formed requirement is a statement [17] that

- can be verified,
- has to be met or possessed by a system to solve a stakeholder problem or to achieve a stakeholder objective,
- is qualified by measurable conditions and bounded by constraints and defines the performance of the system when used by a specific stakeholder or the corresponding capability of the system, but not a capability of the user, operator or other stakeholder.

This international standard supports also guidelines on how to write well-formed requirement specifications. A simple requirement is a description which transcribes or expresses a demand and its related conditions and constraints. This description is reported in a formal way, often described in the form of a natural language, optionally with support of a description template such as the common user voice template. Such a statement should include a subject, verb and a complement. Requirements should always state the considered subject (for instance, for system or software) and what shall

be done for the specific subject (for example user administration page, create new invoices). A common approach is to stipulate the following [17]:

- Requirements are mandatory binding provisions and use 'shall'.
- Statements of fact, futurity, or a declaration of purpose are non-mandatory, non-binding provisions and use 'will'. 'Will' can also be used to establish context or limitations of use. However, 'will' can be construed as legally binding so it is best to avoid using it for requirements.
- Preferences or goals are desired, non-mandatory, non-binding provisions and use 'should'.
- Suggestions or allowances are non-mandatory, non-binding provisions and use 'may'.
- Non-requirements, such as descriptive texts, use verbs such as 'are', 'is', and 'was'. It is best to avoid using the term 'must', due to potential misinterpretation as a requirement.
- Use positive statements and avoid negative requirements such as 'shall not'.
- Use active voice: avoid using passive voice, such as 'shall be able to select'.

In practice, all requirements should be consistent. Therefore, all terms defined in the domain for requirements engineering should be properly specified upfront and applied along all specifications.

2.8.2 Attributes and methods

This section gives an overview of formal requirement readiness criteria. Readiness describes a state which should be fulfilled in order to start development or design of any requirement.

Traditional Requirement Attributes

The authors of [17] have highlighted a list of traditional requirement attributes which should be considered in the event of requirement specification and analysis to guarantee a good quality of requirements:

- Identification. Each individual requirement should be unique. Uniqueness may be achieved by assigning a name tag, sequential number or mnemonic element. Relationships such as linkages could be reflected on identification as well. Distinct identifies supports in tractability of requirements. It is recommended that once an identifier is assigned, it will never be changed, also if the requirement content changes or it will be reused in a later software project.
- Stakeholder Priority. Requirements should be prioritized through a procedure of potential stakeholders. The scale of priority may vary in different organizations. Prominent schemes are low, medium and high or a numerical scale from 1 to 5. Requirements with a low priority are not intended to be mandatory, but they could be investigated when alternatives will be discussed. Priority among stakeholders could vary in different opinions. Identification is often performed with reference to business value, risk or urgency. Having the right stakeholder for prioritization is an important fact to trade off requirements and to balance the influence of adjustments from different stakeholders.
- Dependency. Identify dependency relationships between requirements when they exist. Requirements could have a low complexity or low priority in the opinion of stakeholders but could be fundamental for the software success. When considering mandatory requirements with dependency on supporting requirements, the removal of such a mandatory functionality may indicate the removal of a supporting requirement.
- Risk. Risk mitigation has to be done for software requirements. Therefore, risk analysis techniques are used to discover potential issues and consequences. Dominant risks are often correlated to financial loss, cost of delay, standards in security terms. Additionally, risks can have an influence on loss of confidence of the stakeholders involved.
- Source. The originator or multiple originators of requirements should be captured with a property. This will help in supporting activities for consultancy. Consultancy includes clarification, modification or removal of requirements. The source property indicates also the ownership and where it comes from. The authority to establish the requirement is moved to the appropriate team.
- Rationale. The rationale or the benefit has to be conducted as a property of each requirement. This property should provide a concrete

reason. It should be linked to an objective evidence such as prototyping, supporting analysis, modelling simulation or other substantive studies.
- Difficulty. Difficulty is used as a measure for cost modelling or complexity and should be captured for each single requirement. There exist several schemes to capture it such as easy, normal or difficult. It gives more context in regard to affordability and requirement space.
- Type. The type indicates the class of attributes they present. Requirements can be moved into different groups of types (for example functional or non-functional) for allocation and further analysis.

DRAGON/TREASURE/INVEST

One important property of Agile is transparency. To increase transparency in an agile development domain, visualization techniques might be used. The DRAGON/TREASURE/INVEST method uses 2D-maps with different hierarchy levels instead of flat 1D-records.

The output of this model is a prioritized backlog which can be used as input for the Sprint Planning meeting to maintain the Sprint Backlog. This method suggests a three-level requirement hierarchy which fits in our ALASKA process in regard to Epic, Feature and Story requirements [16], [24]. Each initial letter of the method describes one property of it. Every level of consideration should be checked if its requirements meet the following definitions. Portfolio level readiness-criteria are applied on Epic requirements and should meet the following properties [4]:

- Dependencies
- Risks (focus on business & market-view)
- Assumptions & aspirations
- General requirements & remarks
- Out-of-Scope aspects
- Negotiated agreement between parties
- Constraints & customers
- Artefacts & delivery-formats
- Value-proposition

- Exit-criteria

Program level readiness-criteria are applied to Feature requirements and should meet the following properties [4]:

- Technology stack proposal/ decision
- Risk-of-technology-maturity
- Evaluation-needs
- Architecture runway
- Sourcing-tactics: Home-/Near-/Off-shore; IN-vs.OUT-sourcing
- UX-strategy
- Reliability
- Exit-criteria

Sprint level readiness-criteria are applied to Story requirements and should meet the following properties [4]:

- Independent
- Negotiable
- Valuable
- Estimable
- Sized
- Testable

The INVEST model [24] is a universal model in requirement specification. Many agile teams use this classification to assess their story, in regard to the previously mentioned attributes.

Independent A Story should be independently valued. Hence, it can be implemented, tested, and probably released on its own. In more complex projects, it is obvious that many stories will have sequential dependencies.

For instance, a system might visualize a customer record, list all customer records, then sort the record list. A large number of such stories consist of natural dependencies whereby each single Story provides independent value to the system and might be released independently. Nevertheless, many dependent stories find their way into the backlog, which must be reconsidered [24].

Negotiable In contrast to traditional requirement engineering, a user Story is not a signed contract for specific functions. It might be considered as a placeholder for specifications which have to be implemented, tested and accepted. The negotiation between the stakeholders of business and development team allows room for discussion through collaboration and yields valuable feedback.

Additionally, the degree of negotiability supports teams to accomplish predictability of user stories. Without over-constrained and highly detailed user stories, flexibility may rise for each user Story. With more flexibility, the team has also more opportunities to meet its objectives, which increases trust of the stakeholders [24].

Valuable The aim of an agile team is implementing the most valuable requirements under time and several resource constraints. Every single Story must deliver some value to the stakeholder of the product. Therefore, backlogs are often prioritized by business value. The attribute value is therefore the most important one in this model. The most critical challenge to achieve this attribute is to learn how to write compact and incremental stories.

In contrast to traditional requirements, functional descriptions were often based on the functional breakdown of technical components. This approach might delay the delivery of value because several iterations are needed until all components or layers can be integrated. Bill Wake [1], founder of the INVEST model, described this property in the following way:

> "Think of a whole Story as a multi-layer cake, e.g., a network layer, a persistence layer, a logic layer, and a presentation layer. When we split a Story [horizontally], we're serving up only part of that cake. We want to give the customer the essence of the whole cake, and the best way is to slice vertically through the layers. Developers often have an inclination to work on only one layer at a time (and get it "right"); but a full database layer (for example) has little value to the customer if there's no presentation layer."

[1] Bill Wake. www.XP123.org.

With the vertical proposal, stories provide more value to the user and this results in getting more and earlier feedback [24].

Estimable A user Story should be completed in one single iteration. That means it should be implementable, testable and acceptable in this time frame. Therefore, the Story should be estimable in regard to its complexity. When the team is not able to estimate a Story, there could be some uncertainty or the Story is too large, which must be split up into smaller stories. To remove the uncertainty of stories, technical or functional spikes can be used to reduce this problem. More important than the certain estimate of the Story is the discussion process of all team members. In this conversation process, several hidden assumptions, acceptance criteria and a common understanding of the Story itself will be clarified [24].

Small As stated in the previous attribute, user stories should be small enough to fit in a single iteration. The advantage is that smaller user stories support better productivity and agility. Smaller stories should result in decreased complexity to help to go through faster in the development process. It is important to highlight that complexity has a nonlinear relationship to the size of the Story. Therefore, the estimation of stories using the Fibonacci sequence is very effective because the effort estimation increases in a nonlinear way with the increasing size of the Story [24].

Testable In the agile domain, every line of code should be tested, which implies that the whole Story must be testable. Stories, which are not testable are probably too complex, dependent on other stories from the backlog, or are too vague. A very prominent approach from the XP community is the test-driven development. With this method, unit test cases will be written before the actual development starts. This approach also supports in definition of acceptance criteria and the corresponding functional tests. Additionally, knowledge of how to test the Story helps in the implementation of the Story as well [24].

2.8.3 Machine Learning

The quality of requirements is a key success factor in any software project. Low quality in the requirement engineering process can result in errors in

the development phase of a project, which are expensive to correct.

In regard to a specific project, many people and a huge quantity of requirements might be involved. Machine learning algorithms have been evolved in the last couple of years for text classification. The analysis of quality from requirement specifications can be handled with machine learning techniques as they are robust against noise and able to recognize irrelevant aspects in the different specifications [1], [8] [28], [33].

Case-based reasoning and neural network

Case-based reasoning (CBR) is a method from the artificial intelligence domain to solve issues, based on past resolutions of comparable issues. It is quite similar how humans handle new problems with the help of a lessons learned knowledge base.

Additionally, it can be used to create new solutions to unseen problems ,based on past decisions, and to gain new experience. This process will accelerate the analysis phase because several tasks are quite identical to tasks which have been learned in the past and the possible solution can be derived from an existing knowledge base.

A CBR cycle is based on four fundamental steps: retrieve, reuse, revise and retain [20]. The authors in [1] describe the optimum usage of CBR under the following conditions:

- Reoccurring issues or cases happen.
- The application domain has no specific base model.
- Resolutions of previously learned examples are considered as an advantage.
- Retrieval from applicable solutions are stored in a knowledge base.

In combination with CBR, artificial neural networks (ANNs) can be used to solve issues. ANNs are influenced by the information flow of the nervous human brain system. The neuron receives a number of input values, in this case quality indicator values. These inputs will be computed with the

corresponding weights and given to a summation function. The output of this step is further transferred to an activation function which will generate the final result. The advantages of neural networks are the following [20]:

- Unstructured issues with no specific underlying model can be handled well.
- No reprogramming needed; they learn automatically.
- Can be developed for nearly any application.
- Parallel processing. If a sub-component fails, it still continues with execution.

The following algorithm from [19], [26] was used as a base implementation for the quality analysis which measures the requirements specification quality. Examples of quality categories and indicators are illustrated in Figures 2.8 and 2.9. The algorithm consists of eight steps:

1. Selection of quality indicators categories to measure requirement attributes.
2. Selection of weights w for each quality indicator category. Weight w is a number $0 \leq w \leq 1$.
3. Selection of values between 1 and 5 for indicators categories scores.
4. Selection of minimum and maximum target values for each indicator category score.
5. Selection of minimum and maximum target values for the requirement quality attribute score.
6. Giving each quality indicator a category score (entered by the user).
7. Computing a weighted sum.
8. Comparing the weighted sum with the above defined quality attribute scoring range.

The form of the weighting formula for any requirement quality attribute is

$$w_1 QI_1 + w_2 QI_2 + \ldots + w_n QI_n \tag{2.1}$$

with $QI_1 \ldots QI_n$ are the quality indicators categories measurement and $w_1, \ldots, w_n$ are the weights [19], [26].

To improve this approach, the authors in [20] applied case-based reasoning

to analyse the quality of a software specification to get more accurate results in a more efficient way. It is used to assess the requirements information, in regard to their quality indicators and attributes. It is supplied by the user and allocates the matching quality analysis result combined with the most accurate solution. To further improve the performance in this step, a neural network with a soft computing technique is used and combined with CBR. This hybrid approach can be examined as an expert in the software requirement specification domain. The neural network finds its usage in the retrieve step of the CBR cycle. With reference to recommendation and discovery, it supports the most suitable solution for the given issue by implying to existing cases or past experiences.

The CBR cycle steps in combination with ANN are described as follows [1]:

- Retrieve Step: In the first step, the most comparable case or set of cases is retrieved. At this point, the ANN is used to calculate the similarity level of the unseen case in regard to the knowledge base.
- Reuse Step: At this step, the information will be used to reuse the knowledge and the stored solution if a perfect match exists. A perfect match is reached when the new unseen case has a similarity level of 100% to an existing one from the knowledge base.
- Revise Step: Should a perfect match not exist, this step will adapt the most similar case or set of cases.
- Retain Step: In the last step, the new case will be saved for forthcoming retrievals and issue solving. The knowledge base is updated with the new learning case.

The similarity calculation is based on pattern recognition and the ANN itself follows the supervised learning approach. The training set for the algorithm is based on the knowledge base. A single item from the knowledge base consists of a pair of input and output vectors. When a new case is added in the retain step, the ANN has to be retrained accordingly [20].

The information of the past experiences is stored in a knowledge base. The main advantage of this hybrid approach is the combination of these two methods. It will envelop adaptive learning in solving the issues and this

implies improvement of the efficiency in quality analysis. The following advantages can be summarized [20]:

- Avoidance of step repetition. The quality analysis of software requirements takes many steps to be done before any result could be generated. As lot of steps are similar to past experienced cases, a lot of computation time can be saved by dropping all steps that have been experienced before.
- Reuse step of past experiences. Within the help of the ANN, a new case can be analysed in regard to the similarity level of all stored cases from the knowledge base. In combination with the reuse step from CBR, a solution can be returned without any modification.
- Adaption of existing cases or set of cases to derive a new solution. Often new unseen cases have some similarity with a single or a set of cases from the knowledge base. The most similar cases can be derived and modified to extract a new candidate solution.
- Learning from mistakes. The framework is also able to learn from previous mistakes and prevent the process from making the same mistake again. This results also in an improvement of the computation time.

Rule inference

The authors in [28] discuss a method to use rule inference to analyse domain expert knowledge with a set of categorized requirements. The classification of the quality is carried out by means of two classes, bad and good. To calculate the quality of unseen requirements, the learning algorithm is implemented over a set of pre-defined metrics. The goal of this methodology is that the classifier will predict the class in the same way as a domain expert.

The concept of quality is ambiguous because it depends on several factors. Different stakeholders may estimate the quality of a requirement differently. The key aspect is the training dataset and the quality metrics. Based on these aspects, a model will be built to recognize bad or good requirements. Examples of such metrics are illustrated in Figures 2.8 and 2.9.

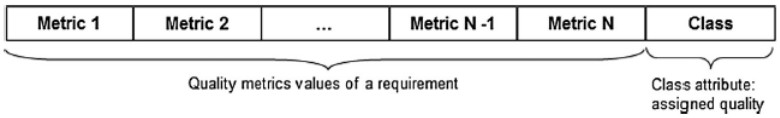

Figure 2.6: Learning Instance: Example format of a learning instance [28].

To generate a classifier, a requirement corpus is needed, which should be classified by domain experts. To extract the quality metrics, a tool called "Requirement Quality Analyser". Each single requirement will be processed by the tool automatically, and the result are the values of the quality metric. With the help of these values, learning instances are ready to be created and will act to generate classifiers. The learning instances will be saved in a Weka-compatible format and consist of a set of attributes and instances. Weka is a prominent open source software and include a collection of algorithms for machine learning tasks [13].

The attribute class includes the numerical values of the metrics and the associated classification class value. On the other hand, the instances include also the numerical values of the metrics and the associated classification classes committed by the domain experts [28]. An example of such learning instance is illustrated in Figure 2.6.

The implementation of a classifier is based on Weka and uses the algorithms PART and C4.5 for rule induction. To improve the accuracy of the classifiers, bagging and boosting techniques are added. Learning instances have the benefit that they add structure to the content which will be handled from the machine learning algorithm. The goal of this step is to create a classifier that has knowledge of the quality value metrics and the dependency itself with the quality. The input for the algorithm are the instances created with the quality metric values. The algorithm then starts to induce rules to decide which class the requirement belongs to. When the learning procedure is finished, new requirements can be estimated. In order to do so, extraction of the quality metrics is needed for each new requirement. The same metrics has to be used as before for the creation of classifiers. Therefore, the algorithm receives the learning instances as input, generated with the new unseen requirements and their corresponding metrics. The output of the

classifier is a class value, which can be interpreted as the most probable prediction which an expert would give [28].

Conditional Random Fields with BIO tagging

Many natural language (NL) requirements are stated in a tentative or speculative manner. In some requirements uncertainty semi-intentionality may have be included deliberately to avoid committing oneself to factual statements about which the author is unsure, for example when transcribing an interview.

It is often the case that such statements contain linguistic cues that appear in the requirements text. Uncertainty cues should be identified and flagged at an early stage, possibly as soon as requirements are written down. The first stage, speculative sentence identification, labels each sentence in a requirements document as either speculative or non-speculative. The authors have used a machine learning approach, first to identify a number of linguistic features typical of speculative sentences and then to applying a Conditional Random Fields (CRFs) algorithm in order to learn models that classify whether or not a given instance of an uncertainty cue is used speculatively.

In order to find sentences in requirement specifications which contain uncertain descriptions, speculative keywords have to be identified. This can be treated as a sentence classification problem. Classes can be distinguished in candidate-speculative and non-speculative. Any requirement description or any sentence in a requirement description which contain at least one speculative keyword for uncertainty will be classified as candidate-speculative.

To identify uncertainty cues, each word token will be labelled with a scheme tag. The labelling task consists of assignment of BIO scheme tags per token. BIO scheme is defined as follows [33]:

- B: first word of a cue
- I: inside a cue
- O: outside (for example not in a cue)

For classification, a broad variation of domain and syntactic features have to be collected to indicate the semantics. In regards to token classification, features could be grouped in the following categories:

- Word-token features. They consists of word lemma, Part-of-Speech tag and chunk tag of the word.
- Context features. They consists of lemma and Part-of-Speech tag of the three surrounding (neighbouring) words.
- Dependency relation features. This refers to grammatical dependency relations between words.
- Co-occurrence feature. This means uncertainty cue keywords in sentences which co-occur.

It is recommended for the training of the classifier to construct feature vectors which are based on the previously described groups and assigned BIO tag labels [33].

The Conditional Random Fields algorithm is used to generate the classification model. CRF is a method to label sequence data like sentences by building a probabilistic model. Within an observation space, the conditional model defines the probabilities of potential label sequences. This model is convenient for the purpose of uncertainty identification as it is applied to classification problems with less fixed class instances, and can also be trained with an exponential loss objective function. The main disadvantage of this algorithm is the slow convergence rate compared to other methods [22].

After training the above discussed CRF model, extraction of the uncertainty cues with the BIO scheme tags can be performed. In case of multiple word tokens marked with B or I, a tag priority is needed such as B $>$ I $>$ O. There exist two cases:

- First token of a cue starts with the B tag. If no B tag exists, look forward for the I tag, and this tag is then considered as cue.
- A cue ends either with an O tag or with a B tag. B indicates here that a new cue starts.

Finally, an additional post-processing step must be performed. Infrequent token cues can not be recognized in regard to data sparseness. Therefore, a

set of tokens were aggregated from the training data. Recognition of such cues is handled by string matching, and a sentence containing such cues, will be marked as speculative [33].

Support Vector Machines

The choice of an algorithm is a key text categorization problem. Experimental results have shown that support vector machines (SVMs) are a prominent classifier in regard to text classification. The support vector machine is a classifier that finds a maximal margin separating hyper plane between two classes of data. There are non-linear extensions to the SVM, but the linear kernel outperforms non-linear kernels in text classification. This algorithm is assigned to the group of supervised learning algorithms in regard to classification problems as well as regression.

An important advantage is the fact that SVMs are able to use different kernels with the goal to transform data. Linear classification techniques can be applied on non-linear datasets. Kernel equations have the power to transform non-separable linear data from the origin domain into another domain where the model turns into linearly separable.

The text categorization process itself is partitioned into two steps. First, the model is trained by means of a training set with collected requirement specifications and known categories. Second, the trained classifier is applied to unknown data to classify it. To train the learning algorithm, text preprocessing has to be performed, including a vector space model, and feature selection.

Feature selection is often preferred due to high-dimensional text characteristics, to reduce the feature space and to evolve the classification quality. Prominent methods for feature selection are document frequency DF, inverse document frequency IDF or mutual information MI.

When the previously described steps are completed, the requirement specification is transformed into a feature vector. After training and learning, the

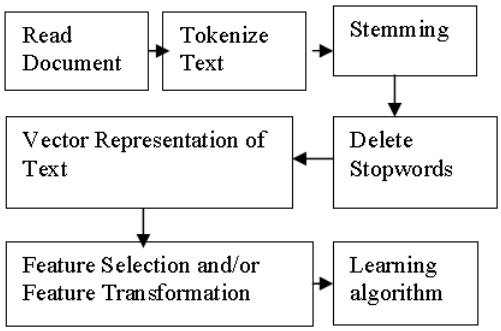

Figure 2.7: Machine Learning: Overview of text classification process [8].

newly learned classifier can be used to classify unknown requirements [8]. The general procedure is illustrated in Figure 2.7.

To use such an algorithm, a training dataset for the classifier has to be created with AVL domain requirements. After the analysis of our current requirements, there exists no correlation to perform online training due to different requirement definitions in different projects. Additionally, there is no information by Atlassian to include a training procedure in Jira. Therefore, a decision was made not to focus on implementing a machine learning algorithm [25].

2.9 Requirement Language Description

Textual requirements should always state and describe what is needed and not how it can be achieved. In the specification, imprecise and general words shall not be used. The consequence of vague requirements are issues in the verification process because they are hard, or in the worst case impossible to verify. Another problematic issue with vague requirements is they could invite multiple interpretations. The following list highlights prominent examples of ambiguous and boundless terms [17]:

- Superlatives (such as 'best', 'most')
- Subjective language (such as 'user-friendly', 'easy-to-use', 'cost effective')

- Vague pronouns (such as 'it', 'this', 'that')
- Ambiguous adverbs and adjectives (such as 'almost always', 'significant', 'minimal')
- Open-ended, non-verifiable terms (such as 'provide support', 'but not limited to', 'as a minimum')
- Comparative phrases (such as 'better than', 'higher quality')
- Loopholes (such as 'if possible', 'as appropriate', 'as applicable')

To guarantee that Features and Stories at AVL meet some level of requirement language criteria, a reporting tool has to be implemented which checks the description of each created issue. To highlight conflicting issues:

- Each Feature and Story will have an additional field which shows the quality value and highlights which indicator is not fulfilled, and
- a gadget will be implemented which visualizes the requirement quality and which requirements have missing fields.

Therefore, several requirement indicators and a comparable model have to be defined. To begin with an evaluation process, the assessment of quality of software requirements, written in natural language, has to be evaluated against a given model. The quality model consists of the following properties [23]:

- Quantitative (allows the collection of metrics)
- Corrective (helpful in the detection and correction of the defects)
- Repeatable (provides the same output against the same input for every domain)

Beside the quality model, indicators have to be defined, which consist of structural and syntactic conditions. They are based on the requirement specification documents or sentences and produce information on a singular property of the requirement itself (example indicators are illustrated in Figure 2.8).

Detection of indicators will be managed in the parsing phase of the requirements. With the help of the indicators, identification of possible issues associated to the assigned property and corrective operations, may be done. The scope of the indicators can be distinguished according to two levels. On

PROP.	INDICATOR	DESCRIPTION	NOTES
NON-AMBIGUITY	Vagueness	A Vagueness Indicator is pointed out if the sentence includes words holding inherent vagueness, i.e. words having a non uniquely quantifiable meaning	Vagueness-revealing words: *clear, easy, strong, good, bad, useful, significant, adequate, recent, ...*
	Subjectivity	A Subjectivity Indicator is pointed out if sentence refers to personal opinions or feeling	Subjectivity-revealing wordings: *similar, similarly, having in mind, take into account, as [adjective] as possible, ...*
	Optionality	An Optionality Indicator reveals a requirement sentence containing an optional part (i.e. a part that can or cannot considered)	Optionality-revealing words: *possibly, eventually, if case, if possible, if appropriate, if needed, ...*
	Weakness	A Weakness Indicator is pointed out in a sentence when it contains a weak main verb	Weak verbs: *could, might.*
SPECIFICATION COMPLETION	Under-specification	An Under-specification Indicator is pointed out in a sentence when the subject of the sentence contains a word identifying a class of objects without a specifyer of this class	Words needing to be more specified: flow (data flow, control flow, ..), access (write access, remote access, authorized access, ..), testing (functional testing, structural testing, unit testing, ..), etc.
CONSISTENCY	Under-reference	An Under-reference Indicator is pointed out in a RSD (Requirement Specifications Document) when a sentence contains explicit references to: not numbered sentences of the RSD itself documents not referenced into the RSD itself entities not defined nor described into the RSD itself	-

Figure 2.8: NL: Indicators for natural requirement evaluation [23]

the first level, single sentences are analysed. On the second level, complete requirement descriptions are analysed [23]. With this information, a possible plugin for Jira could be implemented, which performs the following steps:

- Lexical analysis
- Syntax analysis
- Quality evaluation
- Output results

To measure indicators of requirements, a definition of a sequence of quantifiable indicators is needed. These are linked with the qualitative attributes for the evaluation. A prominent example of such an indicator is the size of a requirement (for example the number of words in its description).

A simple indicator like size affects the quality of the requirement in regard to atomicity. It seems obvious in the case of size that the length of

Indicators	Function	Desirable properties										
		Atomicity	Precision	Completeness	Consistency	Understandability	Unambiguity	Traceability	Abstraction	Validability	Verifiability	Modifiability
Size	Convex	X	●	●	●	●	●	●	●	●	●	●
Readability	Incr./Decr.					X	●			●	●	●
Punctuation	Convex					X	●			●	●	●
Acron. & Abbrev.	Decreasing					X	●			●	●	●
Connective terms	Decreasing	X	X	●	●	X	X	●	X	●	●	●
Imprecise terms	Decreasing		X	●	●	●	X			●	●	●
Design terms	Decreasing								X		●	●
Imperative verbs	Convex	X	●	●	●	●	●	●	●	●	●	●
Conditional verbs	Decreasing		X	●	●	●	●			●	●	●
Passive voice	Decreasing		X	●	●	●	●			●	●	●
Domain terms	Convex	X	X	●	●	●	●	●	●	●	●	●
Versions	Decreasing									X	X	
Nesting	Convex					X				●	●	
Dependencies	Convex	X	●	●	●	X	●	X	●	●	●	●
Overlappings	Decreasing	X	●	●	●	X	X	X	●	●	●	●

Figure 2.9: NL: Indicators for natural requirement evaluation [12]

a requirement must be neither very short nor very long. To classify such measurable content, a set of discrete levels has to be defined. Such discrete levels could be [12]:

- High, Medium, Low or
- Good, Medium, Bad

Primitive measures as size must be transformed into adequate value for the indicator. For this reason, an appropriate function like a convex step function can be applied. This holds also true for other indicators such as amount of ambiguous or domain terms, readability index, amount of imperative verbal forms.

Each distinct indicator has various characteristics. Therefore, different step functions have to be considered as well. In general, the following step functions could be used: increasing, decreasing, convex and concave. The authors in [12] use the indicators shown in Figure 2.9.

3 Implementation

3.1 Approach

To find adequate indicators for usage in the AVL environment, the information from Chapter 2.9 has been collected. In regard to the presented indicators from literature, the following indicators were used and the boundaries and functionality were adapted with reference to the following issues [12], [23]:

- Requirement size
- Imprecise terms and phrases
- Connective terms and phrases
- Incomplete terms and phrases
- Dependency links
- User voice syntax
- Acceptance criteria syntax

3.1.1 Requirement Size

To measure the requirement size, the description field of each Feature and Story requirement is used. The number is based on the amount of words contained in the description. As already discussed, the size of a requirement description should not be too small nor too large. Size is an essential indicator which is related to the properties atomicity, traceability, modifiability, and verifiability [12], [23].

3.1.2 Terms and Phrases

To inspect textual requirement specifications, methods have been implemented to highlight vague and conflicting terms and phrases. Therefore a list of imprecise, connective, and incomplete words and phrases has been defined.

Whenever a Feature or Story will be viewed or edited in Jira, an automatic check will be performed. The conflicting elements will be highlighted in the quality metric field. The syntactic analysis is a necessary step to evaluate the quality of requirements which are stored in Jira. The tool provides currently no automatic inspection of requirement descriptions [25]. Imprecise terms are enumerated due to the fact that those terms introduce ambiguities in requirements. Connective terms are essential in any linguistic formulation, but their abuse might result in a quality reduction. Connective terms have an influence on the properties atomicity, precision, unambiguity, and understandability. The usage of incomplete terms and phrases is in conflict with the atomicity property. The usage also highlights that the requirement has not a clear scope [12], [23].

3.1.3 Dependency Link Complexity

The link complexity will be distinguished between Feature and Story requirements. The Story link complexity is calculated, based on the number of links to other Story requirements. The sum of links represents the complexity value and is also implemented as a custom field on the issue view. The Feature link complexity is calculated with reference to linked Features. When a Feature has a link to another Feature, the sum of the underlying Story complexity value will be added. The existence of dependencies is typically in requirements engineering, but highlighting the complexity of each requirement is necessary. A high complexity value indicates the need for further checks to minimize the amount of dependencies in regard to maintenance and reuse, and is a possible violation against atomicity, traceability, and understandability. On the other hand, a low complexity value may indicate an insufficient analysis of requirements. [12], [23].

3.1.4 User Voice Syntax

As defined in the ALASKA process, each business requirement should be written in user voice syntax. By using this form, the teams are constantly guided to understand who is using the system, what they are specifically doing with it, and why they are doing it. Applying it continuously increases the teams domain competence as they acquire a better understanding of the real business needs of their user. The template for this syntax is as follows [24]:

> As a *"role"*, I can *"activity"* so that *"business value"*

whereby:

- "role" represents a user or perhaps a system who is initiating an activity or receives the output of an activity.
- "activity" represents the action which should be accomplished by the system.
- "business value" represents the value which should be obtained from the activity

This information should be stored in the requirement summary. Therefore a regular expression is used to check if the summary was correct. The only exceptional case are architectural requirements. In this case the indicator will not check the summary syntax [12], [23].

3.1.5 Acceptance Criteria Syntax

The acceptance criteria define the details of the story which must work at the time of acceptance. They should focus strongly on the business perspective rather than on technical details. The syntax is used to describe the behaviour of a Story. When all described conditions are fulfilled, the Story behaves correctly. This syntax also supports test management in regard to test case generation. The recommended and defined syntax is described as follows [24]:

> Given *"a precondition"* And *"another precondition"* When *"an event happens"* Then *"a desired outcome"*

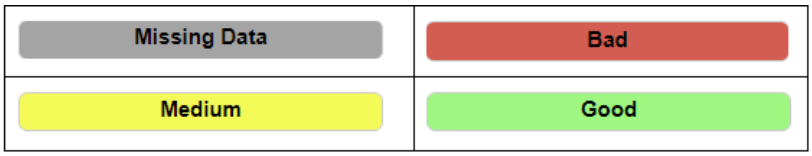

Figure 3.1: Metric: Defined metrics for Story and Features [23]

3.2 Quality Metric of a Requirement

Metrics are reasonable measures to highlight the quality of written requirements. To estimate a value from the above described indicators, four discrete levels have been defined. The levels are Missing Data, Bad, Medium, and Good. These are shown in Figure 3.1.

The metric value will be stored for each Story and Feature requirement which follows the ALASKA process. Additionally, the field visualizes which indicator is not fulfilled and which terms and phrase are violated from the requirement description. A requirement will be classified as Missing Data when one of the DoR fields is absent. In this case, no further indicator will be applied to executed on the requirement.

When every DoR value is specified, the indicators will be processed. The nominal values for these are 0 for Bad, 1 for Medium and 2 for Good. In this context, to derive a quality metric, several indicators may have more influence than others. To determine the value for a requirement, the calculation of a weighted average is performed, which has to be relative to the weight of each single indicator [12], [23]. The weights can be defined in a separate dialogue on the administration page in Jira [25]. The interpretation of the concrete weights are up to the user, according to their preferences. The quality metric for each requirement is finally calculated with the average of the indicators and the levels for those are defined as follows [12]:

- Bad [0, 0.5)
- Medium [0.5, 1.5)
- Good [1.5, 2]

3.3 Jira Gadget Development

The following types of reporting gadgets have been taken into consideration for the implementation. Additionally, with those reporting gadgets a correlation between poor and good requirements can be highlighted. The realization was performed and tested in the AVL environment and self-developed. The implementation was performed as Jira plugin, which are compatible with AVL Jira system [25].

3.3.1 Backlog Readiness Chart

This gadget highlights the velocity for each agile team. Velocity is a calculated value, based on completed Stories per Sprint. Each Story has an assigned Story point value. The aggregated value of Story points is then called velocity [29].

At AVL, the estimation of the value is based on several Sprints. The velocity value will be compared with the accumulated Story Points (Ready SP) from each Story and Spike requirement, linked to a Feature with the specific Team assignment. The Velocity is calculated using the Average of the "Done" Story Points of the last 8 sprints (excluding the last Sprint from an Iteration). The ideal capacity (velocity*4) is shown per team. The number of Ready Story Points is shown per team on Story and Spike level. A traffic light per team indicates if the backlog for this team is ready or not. Figure 3.2 shows an example of the Backlog Readiness Chart. The conditions for the traffic light status is:

- Red: Ready SP $<$ Velocity*4 - 12,5
- Green: Ready SP $>$ Velocity*4 + 12,5
- Yellow: Else

The configuration and the output are defined as follows:

- Selection of a single or multiple teams
- Selection of an Iteration

Team	Velocity	Velocity * 4	Ready	State
AC	0	0	0	
ACDC	13	52	31	
Atlas	0	0	0	
Black	14	56	12	
Blue	0	0	0	
Curry	0	0	0	
Kodiak	0	0	0	
Meerufenfushi	6	24	72	
Peregrine	0	0	0	
Pikachu	4	16	24	
The Avengers	13	52	22	
UX Team	1	4	0	

Figure 3.2: Backlog Readiness: Visualization to highlight the backlog readiness for each team [16]

- Output is a table with five columns (Team, Velocity, Velocity*4, Ready, and State)

3.3.2 Quality Metric Chart

This gadget highlights the quality index from a set of requirements. The chart takes requirements from type Feature and Story into account. Figure 3.3 shows an example of the final Quality Metric Chart. There is one column for:

- Requirements with missing fields
- Requirements with bad quality
- Requirements with medium quality
- Requirements with good quality

The configuration and the output are defined as follows:

- Input is a Jira Query Language (JQL) filter.
- Selection of requirement types Story, Feature, or both.
- Each column shows the number of requirements found.
- With a click on a single column, a drilldown visualization is generated and shows each correspondent requirement key.
- A tooltip highlights the findings.

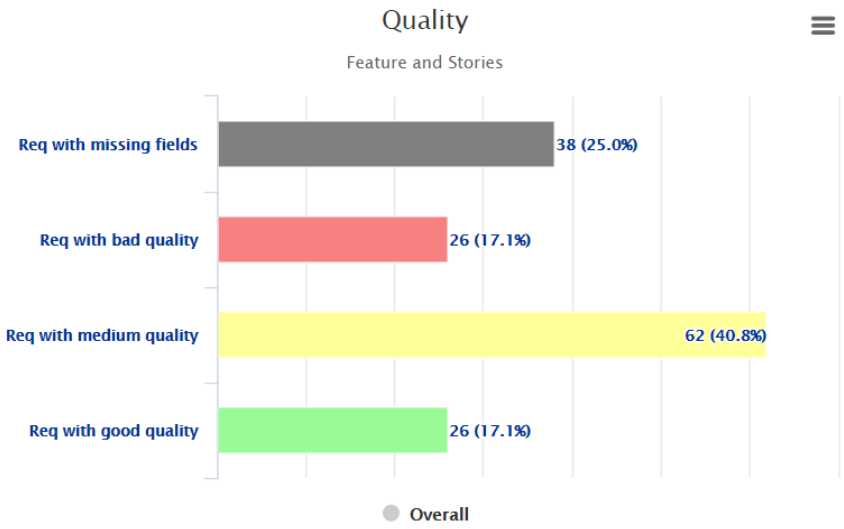

Figure 3.3: Quality Metric: Visualization of each metric with amount of classified requirements [16]

3.3.3 Missing Definition of Ready (DoR) Field Chart

This gadget highlights the amount of DoR fields with missing data. Figure 3.4 shows an example of the missing fields on Feature level.

- Input is a JQL filter or project selection.
- Selection of requirements is configurable between Feature and Story.
- Each column shows the amount of requirements found.

3.3.4 Story Report

This gadget highlights Story requirements in regard to how often Story Points have been changed, how often a Story was moved from one Sprint to another Sprint, and its dependency complexity. Figure 3.5 shows an example of the final Story Report.

- Input is a JQL filter or project selection.

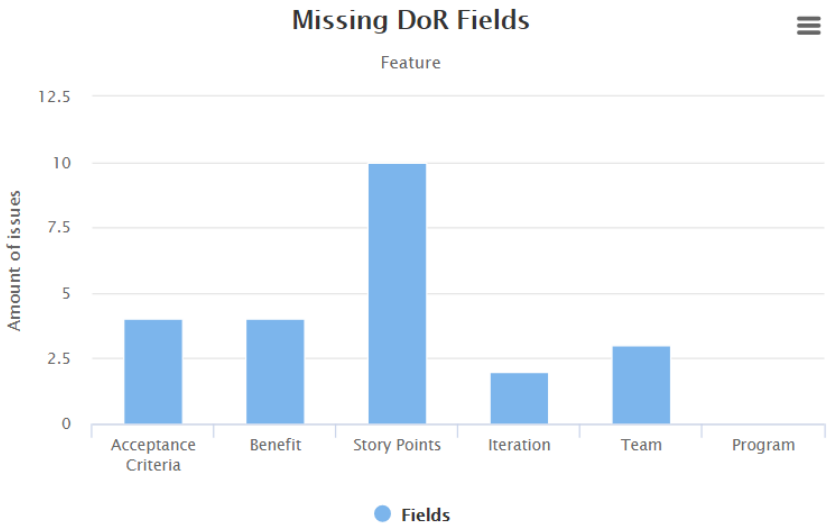

Figure 3.4: Definition of Ready: Visualization of each missing DoR field with the amount of requirements [16]

- Each record has three columns, one for the amount of changed Story Points, one for the amount of participated Sprints, and the last one shows the link complexity value.
- With the click on a single column, the Story will be opened in Jira.

3.3.5 Feature Dependency Report

This gadget highlights how often a Story is linked to a Feature and additionally in which Sprint it was assigned. Figure 3.6 shows an example of the Feature Dependency Report.

- Input is a JQL filter or project selection.
- Each record has two columns.
- The first column shows the number of Stories linked to the given Feature and the second one highlights the link complexity value.
- With a click on a single column, a drilldown visualization is generated and shows the number of Stories per Sprint.
- A tooltip displays the key names of the linked Stories.

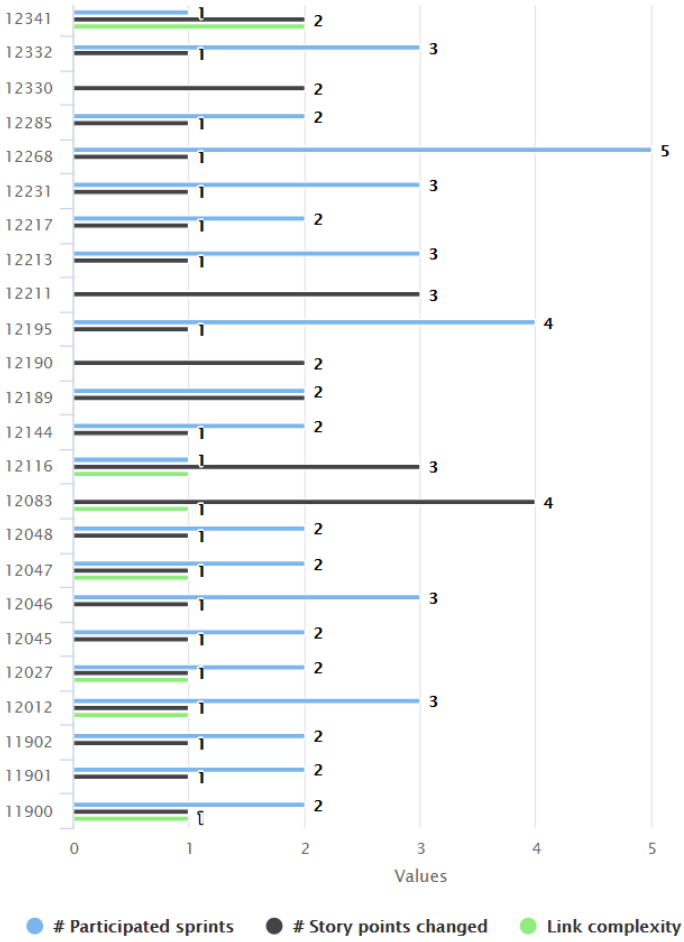

Figure 3.5: Story Report: Statistics of important Story attributes [16]

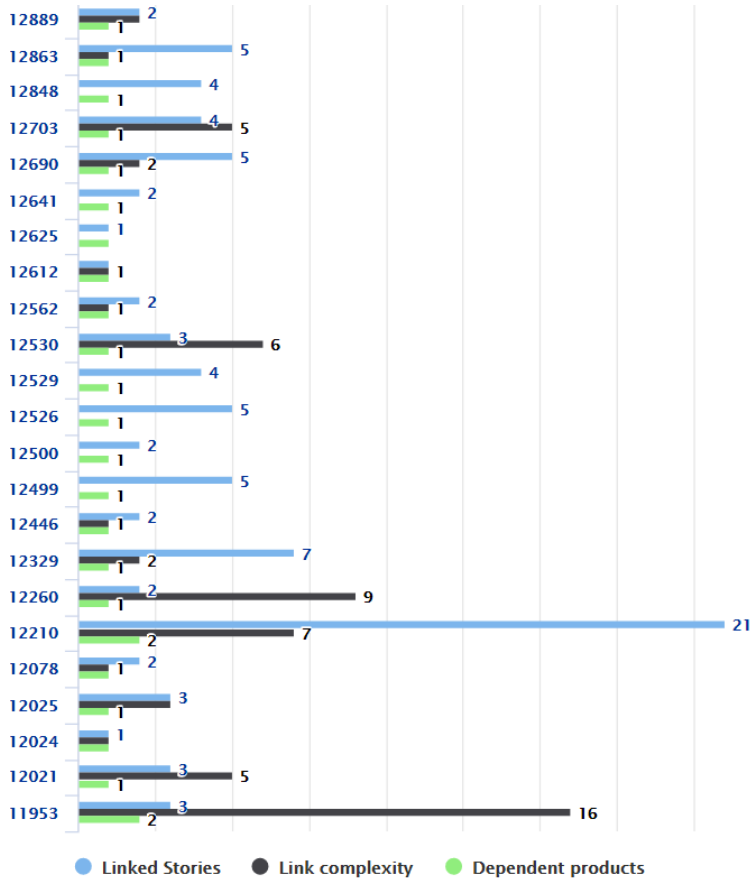

Figure 3.6: Feature Report: Statistics of important Feature attributes [16]

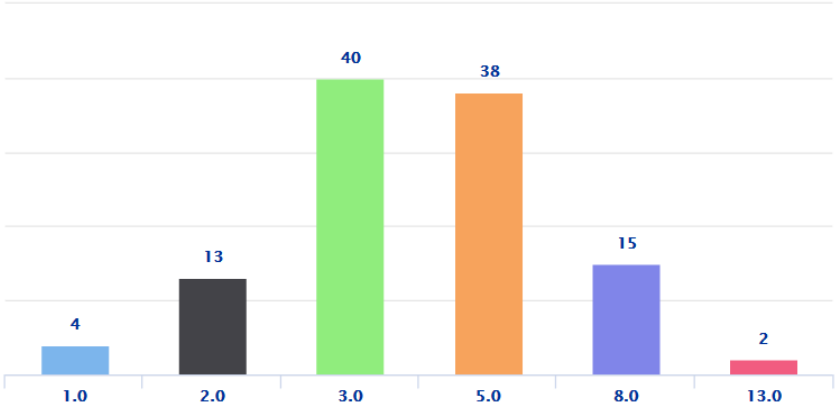

Figure 3.7: Story Point and Sprint assignment: Visualization of complexity amount from different Stories. After drilldown event, the Sprint assignment will be shown [16]

3.3.6 Story Point and Sprint Assignment

This gadget highlights the amount of Story requirements in regard to their Story Point complexity and it shows how often Stories have been postponed to subsequent Sprints. It also highlights if the estimation is defined in the correct pseudo Fibonacci sequence (1, 2, 3, 5, 8, 13). Figure 3.7 shows an example of the assignment of Story Points and Sprints.

- Input is a JQL filter or project selection.
- Story Points are shown on the x-axis and the y-axis shows the number of requirements.
- With a click on a single column, a drilldown visualization is generated and shows how many Stories were assigned to different amount of Sprints.
- With another click on a column, the Stories will be opened in the issue view navigator.

Figure 3.8: Agile Ranking Matrix: Hierarchical view from Epic to Feature to Story [4]

3.4 Additional Enhancements

3.4.1 Agile Ranking Matrix

In this section, the agile ranking matrix will be evaluated for usage at AVL. This possible plugin might be used for Epic, Feature, and Story requirements. An example of the agile ranking matrix is shown in Figure 3.8. Agile is related to transparency, which can be achieved with visualization concepts. The idea behind this technique (also known as Eisenhower-principle) is to give product owners, product managers, or other managerial roles the possibility of ranking their issues on a 2D map. Unranked issues will be arranged on their importance (for example business value) and urgency (for example cost of delay). The matrix is based on 36 numbered cells whereby the first one expresses the most important and urgent one, and the last one (number 36) the least important and urgent one. Issues can be moved from lower positions all the way to top positions using drag and drop.

This ranking process can be enhanced in a hierarchical way. Starting with top level issues like Epic, followed by ranking the underlying Features. After this step, ranking the linked Stories from the most important Feature. The benefit of this method is to visualize the ranking process of different requirement types in a better way [4].

3.4.2 Effort Estimation

In Section 2.8.3, case-based reasoning (CBR) in combination with neural networks was discussed. The hybrid approach was used to categorize newly created requirements in regard to quality aspects based on previously analysed requirements. In this scenario, neural networks are used to find similar requirements in the knowledge base. Case-based reasoning was used to evaluate unseen requirements with previous requirements in regard to their quality analysis. This approach may be further enhanced to recommend other requirement attributes, apart from quality. Instead of saving only past quality analysis results in the knowledge base, requirement properties should be stored. Effort estimation for each single task is done manually by each person who is involved in the development process. Story Point estimation is performed in meeting with a Product Owner and Development Team members, in AVLs process. For each single Story, there is a discussion between all members to assign the Story Point value.

Therefore, the approach to recommend Story Points may support the process to assign correct values. The idea is to adapt the CBR-cycle steps, hence step 2 (reuse) and 4 (retain) was adapted. In step 2, a neural network is used to retrieve similar cases and report their solutions. In this step, other properties like Story Points may be highlighted to support the decision making for effort estimation as well. In step 4, the requirement will be stored in the knowledge base for future retrievals when no match exists for the new requirement. Together with the already provided information, additional attributes like Story Points will be stored. Effort estimation would be one attribute for usage within this approach [20]. There are other interesting properties for further investigation, for example [24]:

- Benefit
- Acceptance Criteria
- Team
- Priority

All of the mentioned attributes are present in each single defined Story. CBR in combination with neural networks is therefore a practicable method. It has advantages when relevant past cases are available, such as the history of

requirements. Many steps have to be performed until these attributes might be aligned. Many new requirements are similar to past cases. Therefore, a lot of discussion time may be saved when using such an algorithm [20].

3.5 Survey OpenReq

The tool under investigation is called InnoSensr[1]. It is part of the OpenReq project, implemented by the Applied Software Engineering team at the Graz University of Technology. The system supports handling requirement engineering tasks. It is an innovative web application to handle quickly changing requirements. The user interface (UI) is made up of modern elements to give stakeholders the possibility of concentrating on their work. The application consists of three main categories to manage requirements.

3.5.1 Requirements

The application interface is partitioned into two main parts, the requirement section and the release section. In the requirement section, stakeholders can easily start creating requirements. Each single requirement has a title, a description and a status. The current available status are New, Planned, Completed and Rejected. Automatically, a unique ID will be generated when a single item is created. A release is defined by an overall title, description, and a release date. Via drag and drop functionality, above defined requirements can be moved to the release section. For each requirement, comments, ratings and users can be assigned. Several comments can be issued to a single requirement and each single comment might be categorized as pro, con or neutral comment. The rating functionality is based on three dimensions: profit, risk and effort. All of those might be selected with a numerical value between 0 and 10. From these ratings, a Multi Attribute User Theory value (MAUT) is calculated. Several stakeholders can be assigned to a single requirement, where they can be rated by their degree of appropriateness and availability. From all stakeholder ratings, an average result will be calculated

[1] http://www.innosensr.com/

for both dimensions. The weights for the requirement and user rating can be changed for each single project under the settings options.

3.5.2 Dependencies

In the dependency tab, users have the possibility of creating dependencies between individual requirements. The selectable types of dependencies are ”Requires” and ”Excludes”. Already defined requirements will be listed above and may be deleted if they are no longer needed. This tool supports automatic detection of hidden dependencies, which exists when a requirement is related to another requirement (for example, requirement B can only be completed when requirement A is done).

3.5.3 Statistics

The last category visualizes the defined dependencies as a graph. Each link between requirements will be shown with a connection between them. When hovering over a specific requirement, only the dependent requirements will be highlighted and the other ones are greyed out.

3.5.4 Suggestions

Requirements

Title and Description Requirements are described by a title and a description. When the text exceeds the boundary of the text element, it is not possible to see the whole text anymore. It would be helpful to see the complete title or description text when clicking on the text elements [25].
Status field Requirements can be described by a status. It seems that there is currently no workflow engine behind them. A workflow engine would support the creation of a requirement until it is completed or rejected. Currently, a created requirement can be marked as rejected. A defined workflow guarantees that requirements go through their various states in a correct

order. When a requirement is moved to the state rejected from any other state, a mandatory comment would be helpful, so that stakeholders know the reason why it was rejected [25].
Effort Effort is described by incurred costs for developing the requirement. It is not obvious if this is a relative or absolute value. A recommendation would be to include more information about the effort estimation and probably also the type of effort [24].
Acceptance Criteria An additional field to describe the acceptance criteria for a requirement may support the team to clearly demonstrate if it has achieved the goal. This field would provide stakeholders more information to ensure that the requirement is developed correctly. Requirements would get more specific with this information and it supports system quality [24].
Requirement Type Requirements might be distinguished between different types. It would be helpful to define a type for each requirement. They could possibly be categorized as business, customer-specific, or non-functional requirements [24].
Move The move functionality should be directly available on the main screen. In the current version, this functionality must be activated from the settings menu button to move requirements between releases and the unassigned requirement section [25].
Proposed by A stakeholder can be proposed for different requirements in regard to the appropriateness and availability. It is currently not obvious who has proposed the person and how the person can be contacted. The indicator consists of three different colours, which imply different meanings. There is no information what value is added to propose a concrete person [25].

Dependencies

Add dependency An information message would be helpful when trying to insert a new dependency, which would result in a conflict. For instance, when a dependency exists with a ”requires” link, it is not possible to add the same requirements with an ”exclude” link.
Check Consistency After a consistency check, there is no feedback if an error exists in the dependency set.

Statistics

Description of the dependency graph Due to a missing legend and no information about the colour scheme, the chart is difficult to interpret. Additionally, there is no information in the graph how the ”exclude” and ”requires” dependencies are highlighted. It is not clear which set of requirements is taken as a basis for the chart. A configuration to select a set of requirements from already defined releases would be helpful [25].

General

View With increasing number of requirements, releases and dependencies, the view will enlarge vertically. Collapsing of releases or an own tab with already released requirements would support to handle the complete view in an adequate size.

Import and Export There is no information about an import, or export functionality. The Object Management Group (OMG) has published the Requirements Interchange Format (ReqIF). Such files could be used to exchange requirements between software tool vendors [10].

3.5.5 Group decision

At AVLs organization, Atlassian Jira is the main RE tool to store and maintain different types of requirements. Jira is a commercial software, hosted as on promise or cloud solution, which is used for requirement collection and agile product management. It supports planning and organizing of tasks, definition of workflow steps, and reporting mechanisms for the whole organization.

In regard to decision types, Jira gives the possibility of defining entry fields which allow stakeholders to prioritize requirements, assignment to specific releases, move requirements to further states, and other property

decisions. Definition of stakeholders is possible in Jira, an assessment of them is not provided and there exist no plugin functionality to do so. In comparison with innosensr, Jira allows only a few group decision methods, which can be used from the core installation [25]:

- Moving requirements to further states, a group of stakeholders might be defined in the workflow process for approval. Only when every member of this group accepts it will the issue be moved further in the process. Most group decisions are currently not supported and are managed in meetings (for example release planning and effort estimation).
- Weighted Shortest Job First (WSJF) is a prioritization model used to sequence jobs (used for Features and Epics) to produce maximum economic benefit. WSJF is estimated as the Cost of Delay (CoD) divided by job size in regard to the current backlog. The value is always stated as relative to other issues in the current backlog. This procedure is used within Atlassian Confluence, a collaborative wiki tool which supports knowledge sharing among teams and it enables the WSJF procedure. It is a group decision, based on several requirements. The result is a prioritized backlog, starting with the most valuable Epic or Feature which are of the shortest length for implementation [16].
- Requirement voting. Atlassian provides a voting opportunity for requirements. The idea behind is that several stakeholders may vote for different requirements. Those with more votes should be planned for the next iteration cycle. In general, votes can be used as a prioritization concept to select which issues should be worked on next.

The general functionality in regard to group decision possibilities is very limited within Jira. The Alaska process includes the concept of group decisions, which are based on formal and structured meetings, which will be hold frequently. The group decision process itself is not tool supported. However, the reconciliation of the decisions will be maintained in tools. Jira provides the storage and the visibility for the results, but this tool itself supports no generation of argumentation-based decisions [25].

3.5.6 Link Dependency

Jira delivers a core functionality to link different types of requirement with different link types. There are plugins available from the Atlassian marketplace to visualize the complete dependency graph between requirements. It enables linking of requirements to the same types, and linking of different types and test cases.

Hierarchy visualization with a tree view or a network graph is also possible. In comparison to innosensr, there is no mechanism to detect hidden dependencies or cycles between them. Jira provides more flexibility in the definition of link types, but there exists no consistency check [25]. Automatic detection of hidden dependencies has several advantages. For example, it supports the traceability of connected elements, reduces risk by overlooking links, and enables further improvements of the project [16].

3.5.7 Tool Comparison

In the past, the collection of software requirements were done manually in informal or formal meetings. This process has some disadvantages, which includes time constraints, huge work load, and cost assessments. To fix these problems, practitioners have implemented software tools to manage requirements, lower complexity, and decrease the work load.

Tool support in requirements engineering is an import aspect and has become very important in organizations today [6], [7], [11], [18], [31]. They should assist in managing requirements efficiently and cover the complete requirement life cycle. The most prominent commercial tools in the market are highlighted in Table 3.9. Each of them has its advantages and disadvantage in regard to their functionality.

The studies have shown that the main focus of these tools has been on collaborative editing, automatic test case generation, managing agile or traditional software approaches, traceability visualizations, customization

possibilities, and integration with other tools (for example version control services). To achieve a certain level of quality for requirements, few techniques have been provided. The listed tools support template creation, which might be used when creating new requirements. They provide the possibility that every requirement property must be described with a concrete value, but no further consistency checks are done if the provided values are appropriate.

Linking of requirements is another essential functionality. Every tool from the above list supports the functionality to link different types of requirements with different type of links, but an automatic consistency check or feedback concerning hidden dependencies is missing. Most of the tools mentioned are focused on the visualization of linked requirements and especially to provide a traceability matrix view.

The overall group-decision possibility is very limited in the listed tools. The paper [31] highlighted additional tools beside the commercial ones which were explored at universities. These tools (MaramaAI, ARM, TIGER Pro) have additional features in regard to automatic inconsistency checking and requirement quality improvements.

In regard to quality improvements, they have implemented syntactic and semantic checks to highlight unverifiable or inappropriate requirements. These checks are based on phrase extraction and comparison with interaction patterns. Highlighting Stakeholders Communities [3] supports recommendation of stakeholders. The approach is based on community detection, according to past stakeholder participation. To use a tool well, the historical data of stakeholder activities are needed. The main steps are data extraction, analysis of past stakeholder activity and the final group classification.

Name	Vendor
Aha!	Aha! Labs
Blueprint	Blueprint Software Systems
Caliber	Micro Focus
Cockpit	Cognition
Jama	Jama Software
Jira	Atlassian
Modern Requirements Suite of Tools	eDev Technologies
Polarion	Polarion Software
TopTeam Analyst	Techno Solutions

Figure 3.9: Tools: Prominent tools for requirement engineering in the market [18]

4 Results

This chapter discusses the results of the analysis and the relation between requirements quality and agile properties. Therefore, the research questions and the related answers are covered.

To measure the quality of agile requirements, managed and stored with Atlassian Jira, several indicators were implemented to address these quality issues. These indicators can be grouped in syntax checking and validation of forbidden words in the requirement description. Before an overall quality value is calculated, every readiness field has to be specified. As each quality value is evaluated for each single Feature and Story requirement, the quality evaluation can also be performed on a global project level. The evaluation was obtained on three different projects (A, B, C). Project A consists of 388 issues (52 Features/ 336 Stories), project B of 142 issues (41 Features/ 101 Stories) and project C of 177 issues (38 Features/ 139 Stories). To compare the different projects with each other, an analysis was carried out on the basis of missing DoR fields, the overall quality assignment and the detailed Feature/ Story report.

Project A This project emphasizes the best results. On the Feature level every DoR field is defined and only 13 Stories have missing DoR fields. The overall quality evaluation illustrates the following results in Figure 4.1.

The statistics show that the project has a quite high link complexity and also relations to other products. Interestingly, it is the project with the most issues and the highest complexity values. As stated before, this project emphasizes the best requirement classification results. The comparision between committed and planned agile points shows also the best results. About 65,8 % of the requirements were completed in one Sprint and only 34,2 % need several Sprints to be finished.

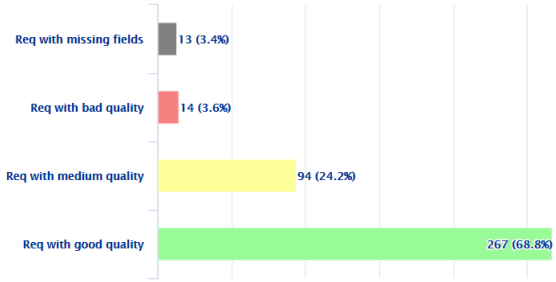

Figure 4.1: Project A: Quality metric

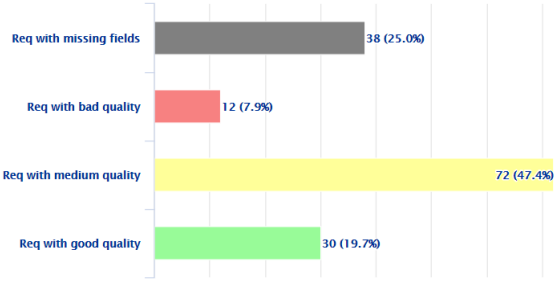

Figure 4.2: Project B: Quality metric

Project B The overall quality evaluation illustrates the following results in Figure 4.2.

About 34,6 % of the requirements were completed in one Sprint and 43,5 % need more than one Sprint to be finished. About 21,9 % were specified but never planned for implementation.

Project C The overall quality evaluation highlights the following results in Figure 4.3.

About 43,8 % of the requirements were completed in one Sprint and 40,3 % need more than one Sprint to be finished. About 15,9 % were specified but never planned for implementation.

The highest correlation of the quality metric and the implemented statistics can be observed with the Sprint assignment. The projects under investigation

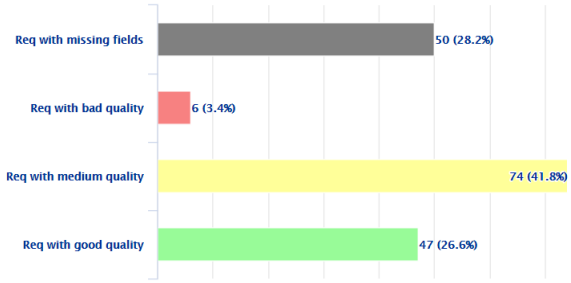

Figure 4.3: Project C: Quality metric

have shown, that postponed Stories are mostly classified as Medium or with Missing Data.

5 Conclusion

This chapter concludes this thesis by answering the research questions, giving indicator recommendatiosn for AVL and indicating limitations and issues for future work.

In this thesis, a method has been presented to evaluate the quality of software requirements. The first research question of this thesis is ”How should the quality of agile requirements be verified?”. First of all, verification must be based on the prerequisite that every requirement field is specified and no information is missing. The results have shown that many postponed Stories were classified with Missing Data. After every field has been defined, concrete quality classification can be performed. The indicators presented in this thesis are reasonable when it comes to usage within AVL organization.

The second research question of this thesis is ”Is there any correlation between the quality of requirements in regard to committed and planned agile points?”. The evaluation of the results have shown, that postponed Stories which need more than one Sprint to be finished were not well defined. These were mostly classified as Medium or Missing Data. This statement shows that the quality of requirements have high impact on committed and planned agile points. Settings of weights for the lexical and syntactic analysis are important aspects. Increasing the lexical indicators results in worse results. To achieve equality between committed and planned agile points, lexical verification has to be considered as main indicator for further improvements.

Overall, this result can be taken into consideration to further enhance the verification and validation process. The lexical and syntactic analysis are first steps to verify the quality level of specified requirements. It supports engineers in acquiring the knowledge necessary to define requirements in the recommended syntax the respective company wants. Due to the

evaluation at the project level, where several thousands of requirements might exist, it is useful to find a group of requirements which need higher attention to solve logical issues. The implemented indicators and metrics, combined as reporting gadgets and custom fields are recommended for usage at AVL environment. To enhance the overall outcome of this procedure, semantic analysis would be a further improvement to guarantee more accurate results. Another advantage of semantic analysis is to provide more explanation to the user. Such explanations help to understand why some words and phrases should be avoided. Another improvement would be to investigate in mechanisms to ensure that no requirement field is absent when a Story will be planned for implementation. To receive more accurate results in regard to the size indicator, the nominal value limits should be configured for each project independently as the requirement description size differs a lot in individual projects. In this thesis, the requirement focus was on Story and Feature requirements.

In AVL, there exist many more requirement types which should be taken into consideration in future analysis. In order to improve the quality of written requirements, there is still a need to enhance the automatic detection of properties. Such properties could be discovered by hidden dependencies or automatic classification techniques with machine learning approaches. It has been shown that commercial tools like Jira support stakeholders in managing requirements and focusing on agile project management. There is, however, as yet room for improvement when it comes to enhancing the requirement quality with appropriate methods.

Appendix

Bibliography

[1] Agnar Aamodt and Enric Plaza. "Case-based Reasoning: Foundational Issues, Methodological Variations, and System Approaches." In: *AI Commun.* 7.1 (Mar. 1994), pp. 39–59. ISSN: 0921-7126. URL: http://dl.acm.org/citation.cfm?id=196108.196115 (cit. on pp. 35, 37).

[2] Muhammad Ovais Ahmad, Jouni Markkula, and Markku Oivo. "Kanban in Software Development: A Systematic Literature Review." In: (Sept. 2013), pp. 9–16 (cit. on pp. 8, 9).

[3] Zeina Azmeh, Isabelle Mirbel, and Pierre Crescenzo. "Highlighting Stakeholder Communities to Support Requirements Decision-Making." In: *Requirements Engineering: Foundation for Software Quality*. Ed. by Joerg Doerr and Andreas L. Opdahl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 190–205. ISBN: 978-3-642-37422-7 (cit. on p. 66).

[4] Kneisel B. *DRAGONs & TREASUREs" – gelebte Innovations-Strategie*. http://2017.agileworld.de/sites/agileworld/files/aglwrld_bokherontix_dragon-treasure_2017-06-26rlsd.pdf. Accessed: 2018-03-26 (cit. on pp. 31, 32, 58).

[5] Kent Beck et al. *Manifesto for Agile Software Development*. 2001. URL: http://www.agilemanifesto.org/ (cit. on pp. 5, 6).

[6] Mohammad Bokhari and Shams Siddiqui. *A Comparative Study of Software Requirements Tools For Secure Software development*. Feb. 2009 (cit. on p. 65).

[7] Juan Manuel Carrillo de Gea et al. "Commonalities and Differences between Requirements Engineering Tools: A Quantitative Approach." In: 12 (Dec. 2014), pp. 257–288 (cit. on p. 65).

[8] L. Dan, L. Lihua, and Z. Zhaoxin. "Research of Text Categorization on WEKA." In: *2013 Fourth International Conference on Intelligent Systems Design and Engineering Applications (ISDEA)*. Vol. 00. Jan. 2013, pp. 1129–1131. DOI: 10.1109/ISDEA.2012.266. URL: doi.ieeecomputersociety.org/10.1109/ISDEA.2012.266 (cit. on pp. 35, 43).

[9] M. Ann Garrison Darrin and W. S. Devereux. "The Agile Manifesto, design thinking and systems engineering." In: IEEE 2017 Annual IEEE International Systems Conference (SysCon), 2017. DOI: 10.1109/SYSCON.2017.7934765 (cit. on p. 6).

[10] Christof Ebert and Michael Jastram. "ReqIF: Seamless Requirements Interchange Format between Business Partners." In: 29 (Sept. 2012), pp. 82–87 (cit. on p. 63).

[11] J. M. Carrillo de Gea et al. "Requirements Engineering Tools." In: *IEEE Software* 28.4 (July 2011), pp. 86–91. ISSN: 0740-7459. DOI: 10.1109/MS.2011.81 (cit. on p. 65).

[12] Gonzalo Génova et al. "A framework to measure and improve the quality of textual requirements." In: *Requirements Engineering* 18.1 (Mar. 2013), pp. 25–41. ISSN: 1432-010X. DOI: 10.1007/s00766-011-0134-z. URL: https://doi.org/10.1007/s00766-011-0134-z (cit. on pp. 46–50).

[13] Mark Hall et al. "The WEKA data mining software: an update." In: *SIGKDD Explorations* 11.1 (2009), pp. 10–18 (cit. on p. 39).

[14] E. Hanser. "Agile Prozesse: Von XP ueber SCRUM bis MAP." In: *Agile Prozesse: Von XP über SCRUM bis MAPs*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. ISBN: 978-3-642-12312-2. DOI: 10.1007/978-3-642-12313-9 (cit. on pp. 5, 7).

[15] Petra Heck and Andy Zaidman. "A Quality Framework for Agile Requirements: A Practitioner's Perspective." In: *CoRR* abs/1406.4692 (2014). arXiv: 1406.4692. URL: http://arxiv.org/abs/1406.4692 (cit. on pp. 27, 28).

[16] Scaled Agile Inc. *Scaled Agile*. Online; accessed 14-November-2017. 2017. URL: http://www.scaledagileframework.com (cit. on pp. 18–20, 26, 27, 31, 52–57, 64, 65).

[17] *ISO/IEC/IEEE 29148: 2011(E): ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes –Requirements Engineering*. IEEE, 2011. URL: https://books.google.at/books?id=bw9%5C_nQAACAAJ (cit. on pp. 28, 29, 43).

[18] Beatty J. et al. *Requirements Management Tool Evaluation Report*. 2016. URL: http://assets.cdnma.com/13314/assets/Website%20Downloads/2016-Seilevel-RequirementsTool-Evauation-Report-FINAL.pdf (cit. on pp. 65, 67).

[19] H. M. Jani. "Applying Case-Based Reasoning to software requirements specifications quality analysis system." In: *The 2nd International Conference on Software Engineering and Data Mining*. June 2010, pp. 140–144 (cit. on p. 36).

[20] H. Mat Jani and A. Tariqul Islam. "A framework of software requirements quality analysis system using case-based reasoning and Neural Network." In: *2012 6th International Conference on New Trends in Information Science, Service Science and Data Mining (ISSDM2012)*. Oct. 2012, pp. 152–157 (cit. on pp. 35–38, 59, 60).

[21] Henrik Kniberg. *Kanban and Scrum - Making the Most of Both*. Lulu.com, 2010. ISBN: 0557138329, 9780557138326 (cit. on pp. 8, 18).

[22] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data." In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282–289. ISBN: 1-55860-778-1. URL: http://dl.acm.org/citation.cfm?id=645530.655813 (cit. on p. 41).

[23] F. Fabbrini ; M. Fusani ; S. Gnesi ; G. Lami. "The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool." In: *Software Engineering Workshop, 2001. Proceedings. 26th Annual NASA Goddard*. 2001. DOI: 10.1109/SEW.2001.992662 (cit. on pp. 44, 45, 47–50).

[24] D. Leffingwell. "Agile Software Requirements, Lean Requirements Practices for Teams, Programs, and the Enterprise." In: *Agile Software Requirements, Lean Requirements Practices for Teams, Programs, and the*

Enterprise. Boston: Pearson Education, 2011. ISBN: 978-0-321-63584-6 (cit. on pp. 18, 19, 21, 23–27, 31–34, 49, 59, 62).

[25] Patrick Li. *JIRA 7 Essentials - Fourth Edition*. 4th. Packt Publishing, 2016. ISBN: 1786462516, 9781786462510 (cit. on pp. 43, 48, 50, 51, 61–65).

[26] Hajar Mat Jani and Salama Mostafa. "Implementing Case-Based Reasoning Technique to Software Requirements Specifications Quality Analysis." In: 3 (Feb. 2011), pp. 23–31 (cit. on p. 36).

[27] Eugenio Parra et al. "A Methodology for the Classification of Quality of Requirements Using Machine Learning Techniques." In: *Inf. Softw. Technol.* 67.C (Nov. 2015), pp. 180–195. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2015.07.006. URL: https://doi.org/10.1016/j.infsof.2015.07.006 (cit. on p. 1).

[28] Eugenio Parra et al. "A Methodology for the Classification of Quality of Requirements Using Machine Learning Techniques." In: *Inf. Softw. Technol.* 67.C (Nov. 2015), pp. 180–195. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2015.07.006. URL: https://doi.org/10.1016/j.infsof.2015.07.006 (cit. on pp. 35, 38–40).

[29] L. Rising and N.S. Janoff. "The Scrum software development process for small teams." In: *IEEE Software Volume 17* (2000), pp. 26–32 (cit. on p. 51).

[30] Ken Schwaber and Jeff Sutherland. *The Scrum Guide*. 2017 (cit. on pp. 9–17).

[31] A. Shah et al. "An evaluation of software requirements tools." In: *2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS)*. Dec. 2017, pp. 278–283. DOI: 10.1109/INTELCIS.2017.8260075 (cit. on pp. 1, 65, 66).

[32] D. Sunner. "Agile: Adapting to need of the hour: Understanding Agile methodology and Agile techniques." In: *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*. July 2016, pp. 130–135. DOI: 10.1109/ICATCCT.2016.7911978 (cit. on p. 1).

[33] Hui Yang et al. "Speculative requirements: Automatic detection of uncertainty in natural language requirements." In: *2012 20th IEEE International Requirements Engineering Conference (RE), Chicago, IL, USA, September 24-28, 2012*. 2012, pp. 11–20. DOI: 10.1109/RE.2012.6345795. URL: https://doi.org/10.1109/RE.2012.6345795 (cit. on pp. 35, 40–42).