User Stories In An Integrated Development Environment

Final Version

L.H.F. (Laurens) Müter, l.h.f.muter@students.uu.nl

First Supervisor: S. (Sjaak) Brinkkemper, S.Brinkkemper@uu.nl
Second Supervisor: F. (Fabiano) Dalpiaz, f.dalpiaz@uu.nl

Master of Business Informatics
Utrecht University
17-11-2018

Content

1. Introduction

In an ever-emerging field of technology within the age of information, software development is starting to become one of the most prominent disciplines within modern companies. While software related projects take a reasonable share of the budget, software management becomes equally important. Hofmann and Lehner (2001) even stated that the second biggest cause of software project failure is related to insufficient requirements. Since Charette (2005) is convinced that nearly twenty percent of software projects with a budget of more than ten million dollars fail, a firm's requirements management strategy could be essential to stay aligned with the competition. Requirement management in this context stretches beyond requirements themselves since a project can only succeed if every stakeholder can interpret the requirement from his field of expertise. For example, it should be possible to derive the architecture, documentation and test plans from a set of requirements where these artifacts coincide with one another. In practice, many problems emerge from the misinterpretation of requirement, so projects with an unclear start are more likely to fail (Hofmann and Lehner 2001).

In more recent work, data gathered from more than 228 companies spread over 10 countries in order to find problems that practitioners experience when building large-scale software project (Méndez Fernández et al., 2016). Requirement engineering problems that often lead to project failure can be summed in communication flaws between the project team and the customer, incomplete or hidden requirements, underspecified requirements, communication flaws within the project team, and insufficient support from the customer.

User Stories have already found their way into practice and are mostly used in Agile Methods to describe functionalities in terms understood by customers. The format in which a requirement is represented is fixed to the phase: *"As a <role>, I want <goal/desire> so that <benefit>."* (Bourque & Fairley, 2014), where role, goal, and benefit are related to the end-user of the required functionality. The main goal of User Stories is to make a smooth transition into the next step in the development process. This could be achieved by using the domain knowledge of a User Story in other activities along the way in order to avoid waste in requirements gathering that becomes invalid before the work begins. This central role of User Stories should also be represented in tools that support them. Thus, tools to support User Stories must be flexible in making bridges between different steps in the development process. Another aspect of User Stories is that they should contain just enough information so developers can estimate the effort to implement a specified requirement. This aspect should be taken into consideration when a User Story is written since both customers and developers should have a clear understanding of what is to be delivered since an acceptance procedure is written by the customer to determine when the goals of the User Story are met.

Although User Stories are already used in practice, there is not much scientific research done about them. Also, the lack of tooling to support User Stories to their full potential is not widely distributed. To tackle the problem of implementing functionality to support User Stories for requirement management and design of software projects, this thesis will describe the implementation of a tool to relate User Stories

with software development environments. When the tool to support User Stories is in place, a study is conducted to evaluate the usability of User Stories in practice using the implemented tool.

By looking at the process of gathering requirements and management surrounding those requirements, one can state that the success or failure of a project can be influenced. Moreover, when placing the requirements at the foundation of the architecture of a project a strong basis can be created for software projects, reducing the chance of failing and improving the quality of a project. Since this management of requirements and the architecture of the functionalities of a project need to be structured, the software development environments should contain a well-structured process of gathering requirements and implementing them.

In this study, I am going to investigate the features to support the development process and how these features can help in forming a project within an integrated development environment.

MRQ: How can requirement management and architecture functionality for software product development be implemented in an integrated development environment?

To narrow down this main research question, the MRQ is divided into five sub-questions:

RQ-1: What are the characteristics of contemporary integrated development environments?

RQ-2: Which vision and characteristics would best describe the requirements and architectural demands within integrated development environment?

RQ-3: What design would best suit the requirements and architectural functionality of this vision in a development environment?

RQ-4: How to implement these functionalities for development support systems?

RQ-5: How to evaluate these functionalities for development support systems?

To dive into the topic of implementing a tool related to User Stories in large-scale software development kits, this thesis is divided into four different chapters.

The **first section** is about User Stories and how they are related to different techniques and methods that are used in practice.

The **second section** takes us on a short tour through the industry and describes current trends related to requirement engineering and software architecture.

The **third section** is about the characteristics of systems to support the development process. Multiple tools and software development kits will be compared to map the current situation of requirement management and to decide in which environment the User Story tools will be implemented.

The three theoretical sections are linked back to practice in the **fourth section**, where five interviews describe the perspectives of experts from different software development companies on requirement engineering.

The **fifth section** is devoted to which vision would best suit the requirements and architectural needs of development process supporting systems. Where a vision is created about how software development should be integrated with the use of User Stories. This section also describes how a design of requirements functionality for development support systems would look like.

After the requirements are clarified, a set of implementable requirements is derived in the **sixth section**. Design decisions related to the User Story supporting tool are also discussed in this section.

Next, the implementation of the clarified features is discussed in the **seventh section**. The look and feel of the tool appear in this section as well as the technical design of the tool.

In the **eight section**, the tool is evaluated by means of experts interviews and architectural comparisments with finished projects.

1.1. Research Method

The main goal of this study to implement supporting tools in a practical context. Since there is not much literature available about tooling to support requirement management in practice, companies have to conduct studies themselves in order to find supporting tools in helping to manage requirements. To address this problem, the main research question is stated:

MRQ: How can requirement management and architecture functionality for software product development be implemented in an integrated development environment?

In order to answer this questions, five sub-questions are formulated which are answered using different methods of research, where the overall design of this study is based on the design cycle of Wieringa.

The design cycle of Wieringa contains four phases, which are related to research question one to four:

1. **Problem investigation** (RQ-1 and RQ-2)
2. **Solution design** (RQ-3)
3. **Design validation** (RQ-3 and RQ-4)
4. **Solution implementation** (RQ-4)

After the last step of this cycle, another cycle starts by evaluating the implemented solution, which is discussed in RC-5:

5. **Problem investigation** (RQ-5)

While the base of this study is the design cycle of Wieringa, there are also two sub-cycles where interviews are used to gain more knowledge about the practical aspects of the research context, as seen in *figure 1.1*.

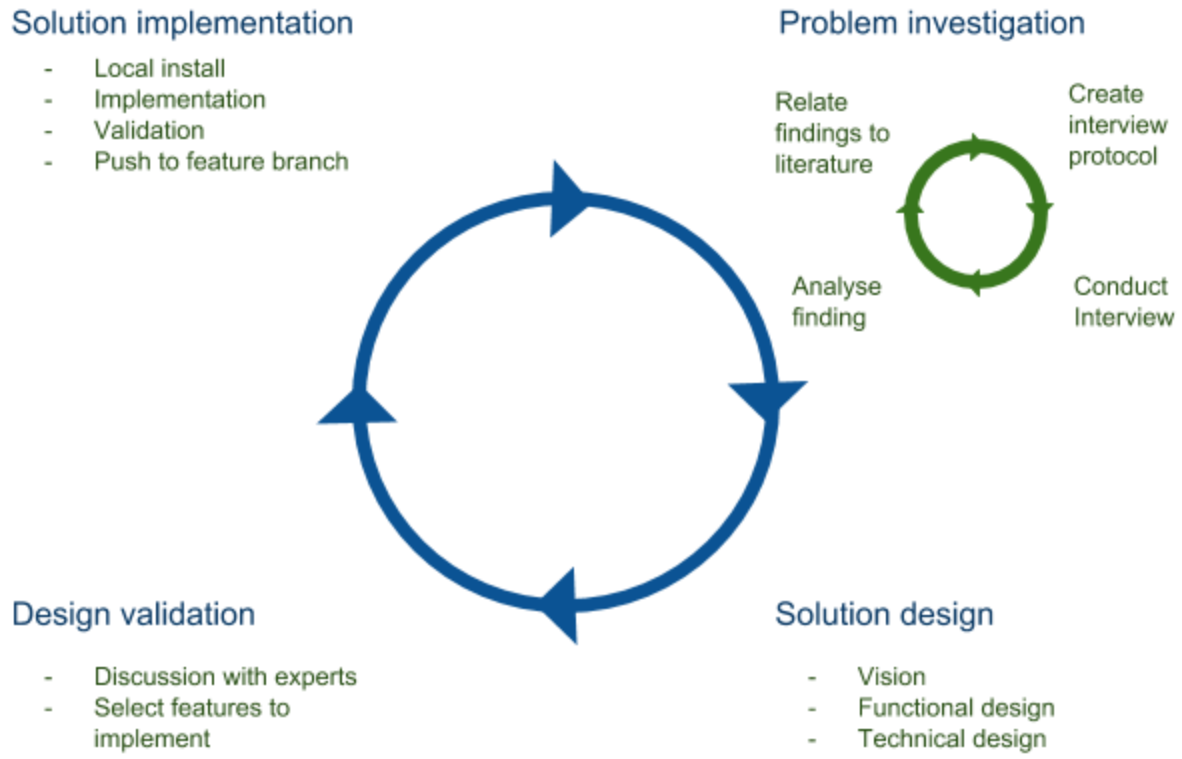

***Figure 1.1**: Research methods summary, based on the Regulative cycle of Wieringa (2009).*

1.1.1. Research Questions

RQ-1: What are the characteristics of contemporary integrated development environment?

In order to get insights into the characteristics of development environments, multiple tools that are used in practice are analyzed using product information and reviews. From these sources of information, a matrix is constructed to describe the key features of each tool and compare those features with each other. This question is limited to three widely used tools, Jira, Github, and Gitlab.

RQ-2: Which vision and characteristics would best describe the requirements and architectural demands within integrated development environment?

This question is answered by means of conducting interviews with experts and a literature study. The expert-interviews are conducted at different companies using a semi-structured interview. A protocol is created where questions regarding requirement management and architecture are stated. The goal of these

interviews is to get insights into how requirement management and architecture coincide with User Stories at different companies.

Interviews will be conducted at five different companies that develop software products. These companies are selected to be a diverse and representative sample of the development ecosystem. Ranging from large companies with many products to small companies who depend mainly on a single product. The semi-structured interviews provide some flexibility in additional questions regarding related subjects. The main subject of the interviews is what tools, processes and artifacts are in place to support User Stories at these companies, regarding requirement engineering and software architecture.

The literature study is devoted to the theory behind User Stories and how User Stories are used in practice. Combinations with different techniques are studied and how the progress and completion of a project can be measured. A snowballing technique is then used since there are not many papers available on User Stories in practice. The literature of Gram Lucassen in used as a starting point from where other papers can be found using the references. These papers are ordered by relevance and quality were the most relevant papers with the highest quality are then used to find more papers, and so forth.

Finally, the information from the interview is combined with the information gathered from the literature study to find overlapping features that serve as input for the next research questions.

RQ-3: What design would best suit the requirements and architectural functionality of this vision in a development environment?

Designing the tool, related to RC-3, is done by using the characteristics from RC-1 and RC-2. The tool will contain multiple elements to support different phases of a project, with respect to the overall vision of requirement management, as described in RQ-2. Within the design of the development support system, there should also be room for the larger picture. User Stories, for example, can be part of an epic with can be part of a Job. On the architecture side, there are different levels of detailing, from overall product description to features and class descriptions. Furthermore, every feature should be self-contained or independent for situational choices defined by the end-user. Also, a metadata model is created to provide a functional overview of the design decisions.

RQ-4: How to implement these functionalities for development support systems?

The implementation of the tools will be done using the open source version of an existing platform, which makes it more easy to review the tools in RC-5. When the design of the supporting system is done, the tool will be implemented so it can be tested in a practical setting. Gitlab is the preferred platform for the implementation, because of its infrastructure, open source character, technical possibilities, and community support.

RQ-5: How to evaluate these functionalities for development support systems?

Evaluating the tools, implemented in RC-4, will be done by asking experts to review the tools and to give their perspective on the question if these tools can contribute to the workflow in practice. Three finished projects are also analyzed, where the implemented tool is used to analyze the User Stories related to the projects, this analysis is then compared with the architecture of these project.

1.1.2. Milestones

1.1.2.1. Milestones per Phase

The milestones for this research project are described per phase of the Wieringa design cycle, by making use of a process deliverable diagram (PDD) as described by van de Weerd and Brinkkemper (2009). The activities and deliverables are displayed in *figures 1.2A - 1.2E* and the activities are described in *tables 1.1A - 1.1E*.

Phase One (Problem Investigation)

The main objective of this phase is to find gaps in the functionalities supported by the most used SDKs at this moment. Existing features are displayed and compared with recommendations of SWEBOK recommendations.

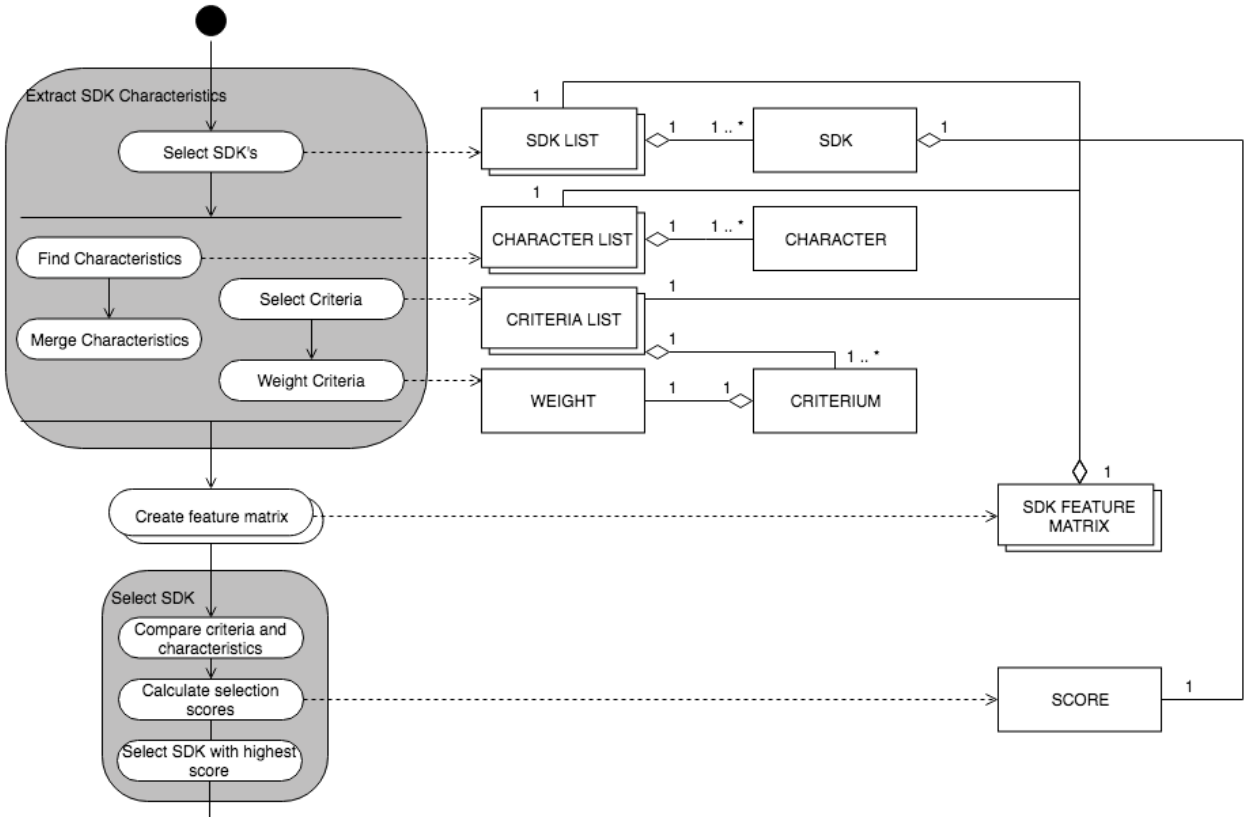

***Figure 1.2A**: PDD of the first phase, where an SDK is selected to implement the new features of this study. The main deliverable of this phase is the SDK FEATURE MATRIX, which contains the most important features of the existing SDKs and a comparison between the most used SDKs.*

Milestone	Activity	Sub-activity	Description
MS1-1-1	Extract SDK characteristics	Select SDKs	Pre-select three SDKs based on their exposure on the web and familiarity with practitioners.
		Find characteristics	By using the SWEBOK recommendations and online documentation, find the most prominent features of current SDKs.
		Select criteria	Make a selection of characteristics and recommendations to make selection criteria for an SDK.
		Weight criteria	Give a weight to all criteria depending on the importance of the criterium.
MS1-1-2	Create feature matrix		Create a matrix to describe the key features of development platforms
MS1-1-3	Select SDK	Compare criteria and characteristics	For every preselected SDK, fill-in the feature matrix to determine the selection score.
		Calculate selection scores	Calculate for every SDK its selection score by multiplying every criterium score with its weight and then sum these values.
		Select SDK with the highest score	Make a selection of an SDK, based on the highest criteria score.

***Table 1.1A**: Activity table related to the PDD of this phase, the activities are grouped into milestones with descriptions. Every milestone has a number corresponding with its **phase**, related **research question** and unique **identifier**.*

Phase Two (Solution Design)

This phase consists of two different parts, a vision (*figure 1.2B*) is proposed which serves as input to design features to realize (*figure 1.2C*) this vision. This ordering has been chosen because of a feedback loop between phase two and phase three. The PDD shows that the design has to correspond to the existing style guides and has to be checked by experts. If the design does not fulfill the expectations of the experts, adjustments can be made which will take the process back to the second phase.

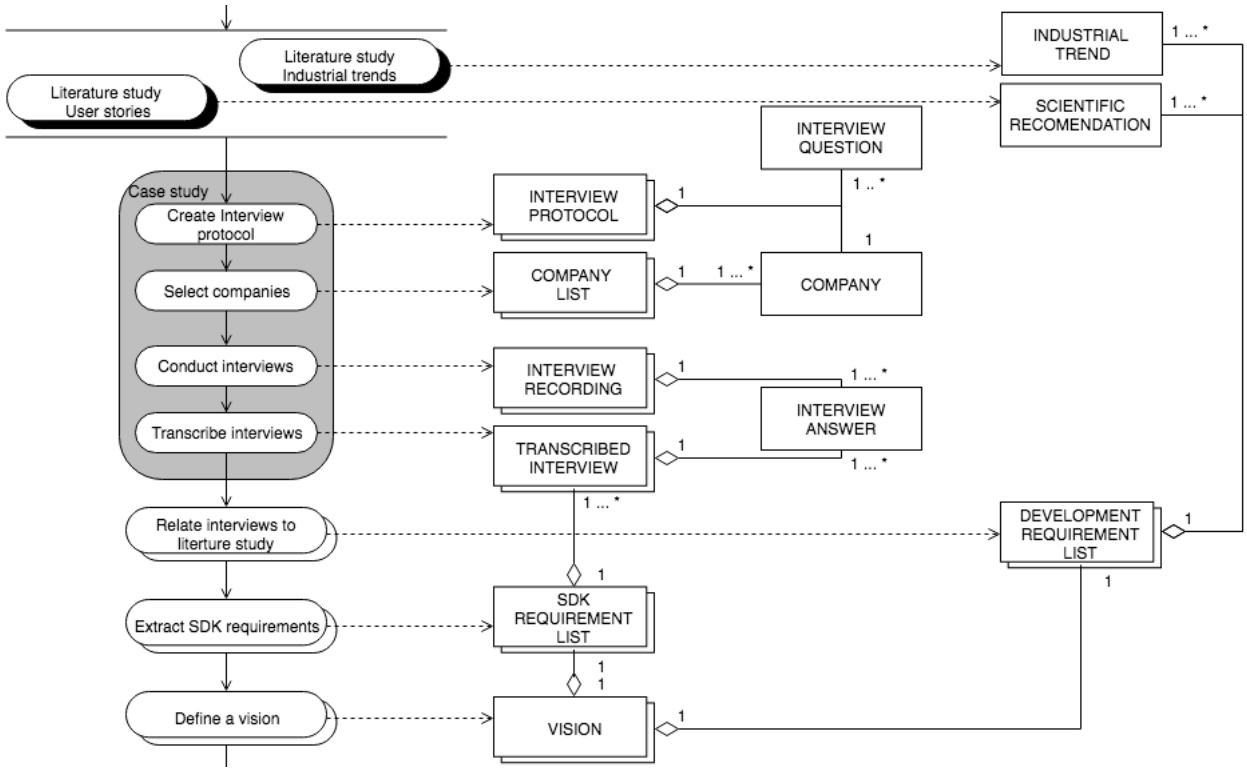

***Figure 1.2B**: PDD of the second phase (part one), where case studies are conducted which, together with a literature review form a vision.*

Milestone	Activity	Sub-activity	Description
MS2-2-1	Literature study industrial trends		Literature study about trends in the software industry
MS2-2-2	Literature study User Stories		Literature study about User Stories
MS2-2-3	Case study	Create an interview protocol	Create an interview protocol
		Select companies	Select five different companies to conduct an interview
		Conduct interviews	Conduct the interview at each of the five companies
		Transcribe interviews	Transcribe the conducted interviews
MS2-2-4	Relate		Relate the information from the interviews to the

	interviews to literature study		theory
MS2-2-5	Extract SDK requirements		Extract characteristics from theory and interviews
MS2-2-6	Define a vision		Use the characteristics to define a vision

***Table 1.1B**: Activity table related to PDD 1.2B, where a case study and two literature reviews are used to define a vision.*

Phase Three (Design Validation)

The PDD in this phase contains the second part of phase 2 (designing the SDK features) because a feedback loop is created when the design does not meet its standards (phase 3).

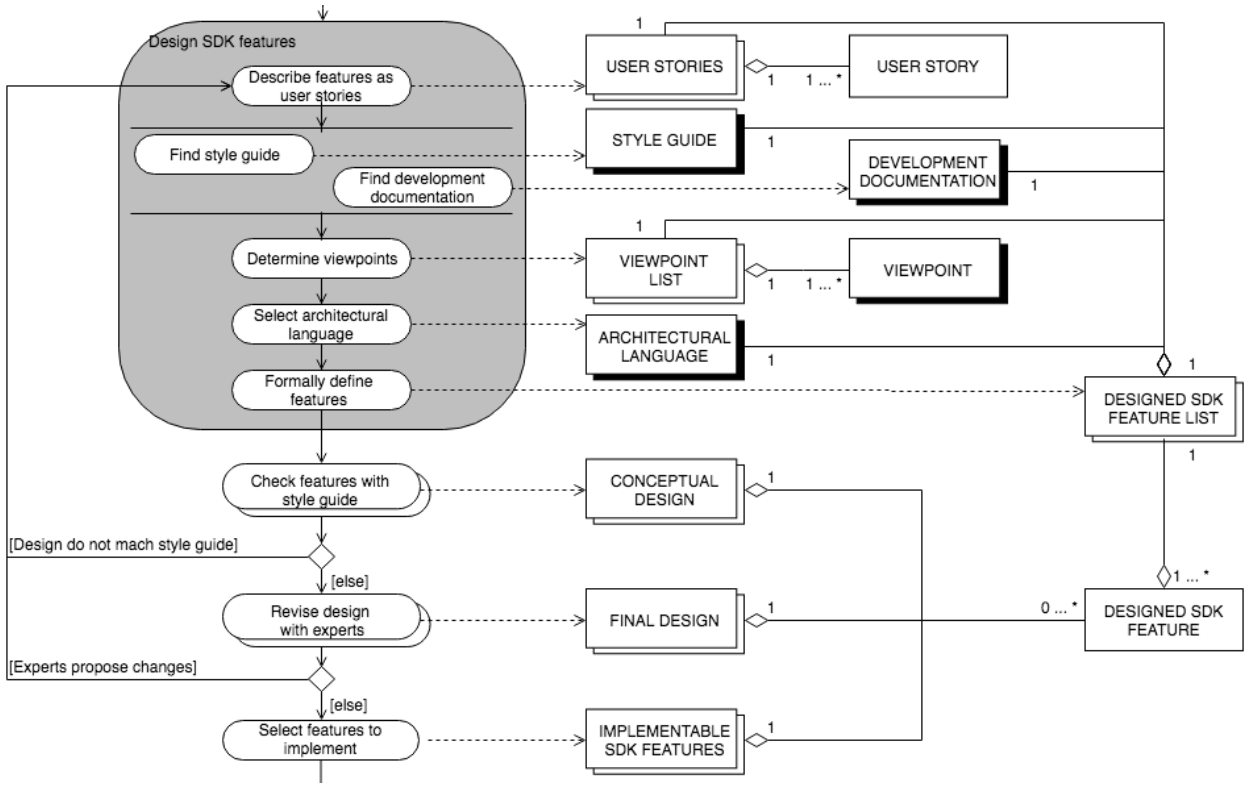

***Figure 1.2C**: PDD of the SDK features design from the second phase (part two) and a feedback loop where changes can be made in the design of the features when they do not correspond to the Gitlab style guide or the expectations of Gitlab experts.*

Milestone	Activity	Sub-activity	Description
MS2-3-6	Design SDK features	Describe vision as	Use the vision of MS2-2-6 to define a set of User Stories, to concretize the vision.

		User Stories	
		Find a style guide	Find design documentation and style guides for Gitlab.
		Find development doc.	Find development documentation and recommendations for developing in Gitlab.
		Determine viewpoints	Determine which viewpoints would best describe the technical and functional essence of the features.
		Select architectural langue	Select an architectural language to describe the chosen viewpoints.
		Formally define features	Describe these features in an architectural language
MS3-3-1	Check features with style guide		Compare the design with the style guide of Gitlab.
MS3-3-2	Revise design with experts		Check if the design fits the expectations of Gitlab.
MS3-3-3	Select features to implement		Select features to implement

***Table 1.1C**: Activity table related to the PDD of this phase, the design of SDK features and a selection which of these features will be implemented in the next phase.*

Phase Four (Solution Implementation)

Implementing the features is done during the first activity, after which a check takes place to test if the feature implementations meet their expectations.

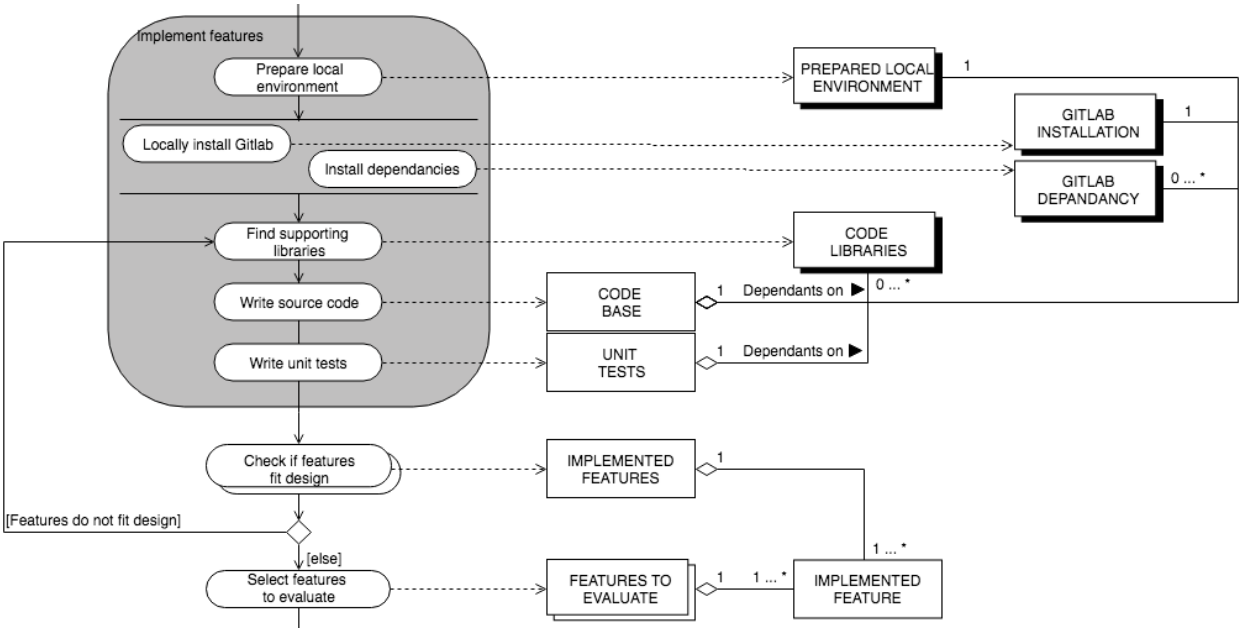

***Figure 1.2D**: PDD of the SDK features implementation. When the features are implemented, a first check will take place in order to spot bugs or unexpected behavior.*

Milestone	Activity	Sub-activity	Description
MS4-4-1	Implement features	Prepare local environment	Check system requirements and make a virtual machine for the development process, possibly with a simple local server to run the local distribution of Gitlab.
		Locally install Gitlab	Install the community distribution of Gitlab to implemented the features.
		Install dependencies	Install dependencies for running the local distribution of Gitlab (for example, programming languages like GO and Ruby on Rail).
		Find supporting libraries	Find libraries in Ruby or Go that could be of use in implementing the features.
		Write source code	Implement the selected features as proofs of concepts
		Write unit tests	Write simple unit tests to spot obvious bugs.
MS4-4-2	Check if features fit design		Check if the implemented features do what the User Stories prescribe.
MS4-4-3	Select features		From the implemented features, select the

	to evaluate		features that are useful for evaluation.

***Table 1.1D**: Activity table related to the PDD of this phase, the implementation and selection of features for evaluation.*

Phase Five (Solution Evaluation)

The final phase is dedicated to evaluating the implemented features by Gitlab experts. When the evaluation of this phase is completed, there might be room for improvements which provide a good input for future research.

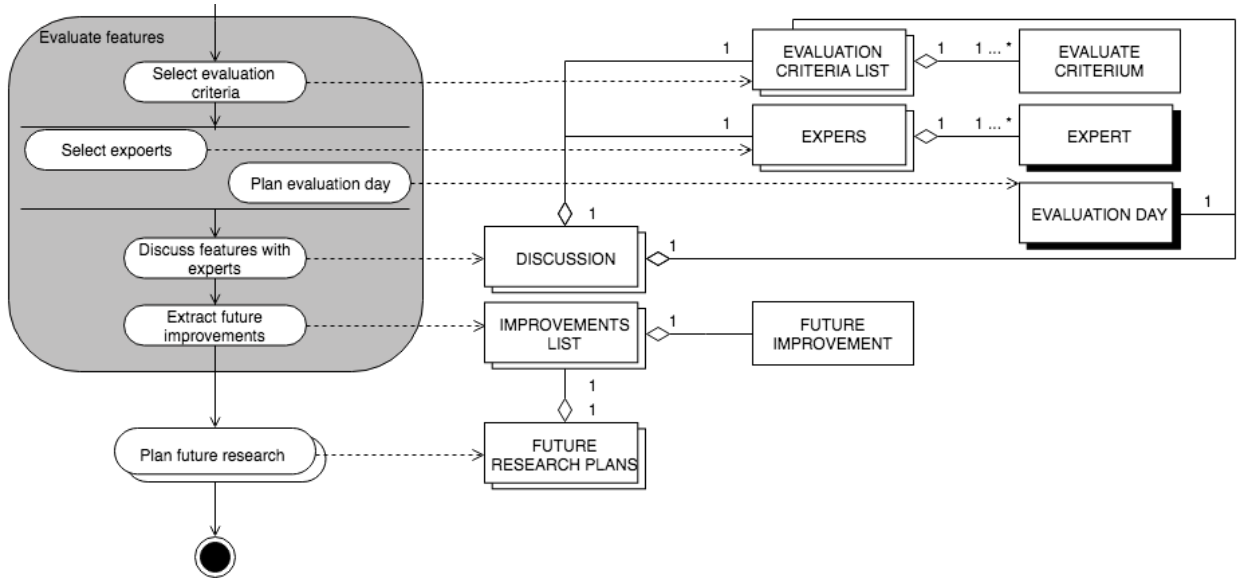

***Figure 1.2E**: PDD of the feature evaluation. The implemented features serve as input for recommendations in future research.*

Milestone	Activity	Sub-activity	Description
MS5-5-1	Evaluate features	Select the evaluation criteria	Select the criteria that will be used to review the implemented features.
		Select experts	Select experts who use Gitlab and invite them to the university for a discussion on the implemented features.
		Plan evaluation day	Plan a meeting for the discussion and reserve the room with the interactive screen.
		Discuss features with experts	Discuss the implemented features with the Gitlab experts, using the prepared criteria.
		Extract future	Extract a list of future improvements and features

		improvements	from the expert-discussion.
MS5-5-2	Plan future research		Make plans for future adjustments and new functionalities. This future research plan holds a list based on recommendations by the expert-discussion.

***Table 1.1E**: Activity table related to the PDD of this phase, where the implemented features are evaluated in an expert discussion at the University.*

1.2.2.2. Gault Chart

A timeline is in form of a Gault Chart displays the milestones and the expected date when they are archived, the color indicates if there is a concrete deliverable associated with the millstone (blue if the milestone results in a deliverable and green otherwise).

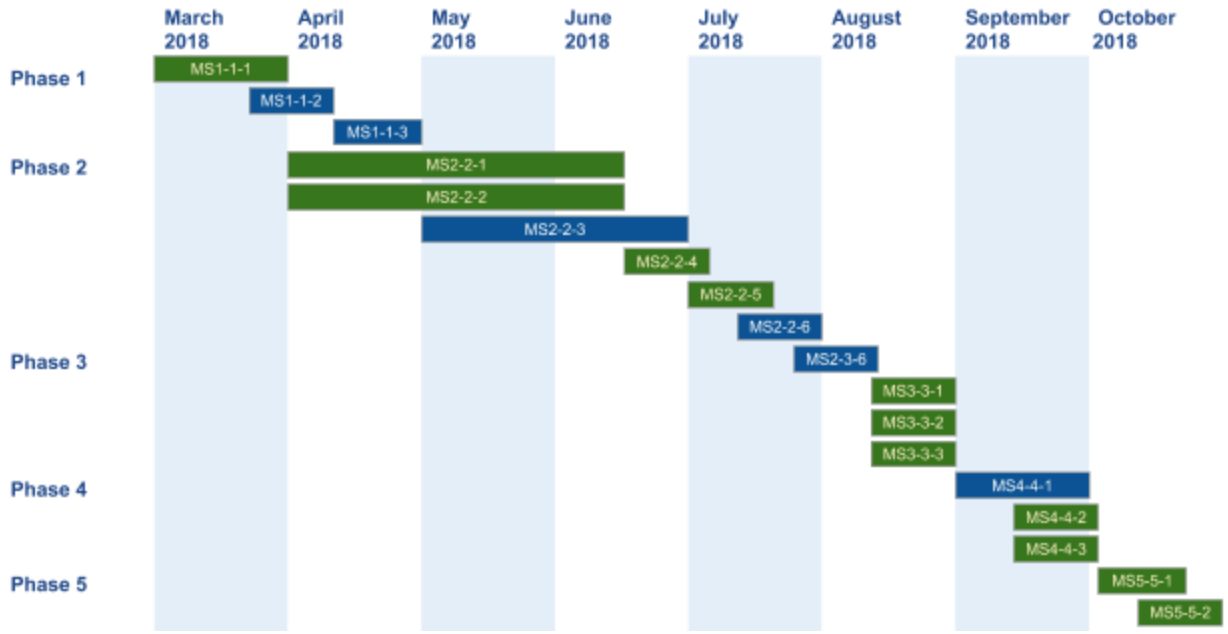

***Figure 1.3**: A Gault chart to describe the timeline of this research project, the milestones are displayed in two different colors (blue when the milestone results in a concrete deliverable and otherwise green)*

1.2. Research Approach

This research contains three perspectives of the User Stories subject, which form the literature review. The core theory about User Stories and the industrial trends together with the five interviews as case-studies form the foundation of the vision.

From the vision, a number of desired features is extracted as input for the design of tools to fulfill requirement engineering and architectural needs in practice.

The case studies serve also as input for available features and tools to support the development process, which forms the basis of the design, together with the desired features.

Some of the features from the design are implemented in the realization phase, which is evaluated in the next phase. Not all features will be implemented and some features will only be implemented as a proof of concepts, to give experts an idea of how these features might work in practice, so new research ideas can be extracted.

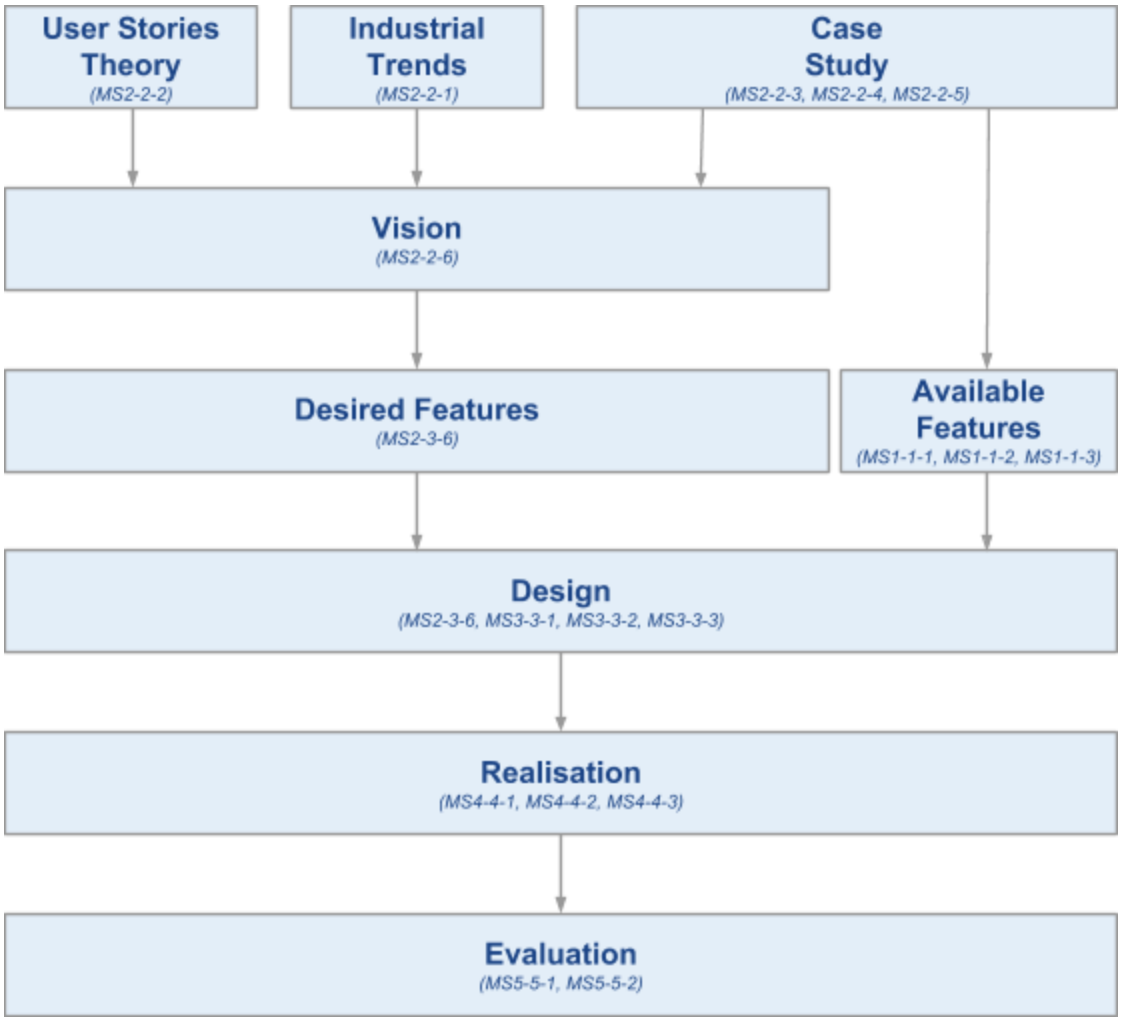

***Figure 1.3**: Summary of the research approach, with milestones included.*

1.3. Relevance

By getting more insights into the practicality of User Stories in practice, and to provide new relevant techniques in analyzing requirements, this study can be useful from practice and scientific perspective.

1.3.1. Scientific Relevance

Since scientific research depends on the evolution of the industry, the industry will always be one step ahead in the matter of requirements and architecture. For that reason, it is relevant to study the industry and build studies uppon problems that are relevant at this time. For example, it is common that research literature is based on requirement engineering conferences for practitioners. It is, therefore, possible that problems that are interesting for scientific research could be behind real-life problems, for example, the industry could have adopted a technique or standards whereas the scientific community is posing new solutions. In order to adopt new solutions time, effort and money can play a role, since transitions are costly most of the time, especially in large organizations.

Another point of interest for the scientific relevance of this research is the creation of new methods to automatically analyze requirements, thus opening new grounds in combining natural language processing with requirement engineering. One of the research gaps, for example, is a supported template for writing tasks. There are a lot of template and recommendations related to User Stories in practice, but tasks that are extracted from these User Stories are often composed by scrum teams and are not used in the rest of the project. All information contained in tasks is therefore ignored in the requirement engineering process.

Next to templates and recommendations of tasks, there are also new methods described in this research, since the design of new features is based on and fed back into practice. New workflows to optimize processes are part of the design.

1.3.2. Practical Relevance

In an ever competing market, it is difficult to stay ahead of the competition. For this reason, it is essential for companies to move forward in adopting better techniques to reduce project costs and to make more efficient use of their resources. Requirement management and architecture are starting to play a more important role since most projects fail by unclear or unrealistic expectations of the project scope.

Another interesting problem is the communication between disciplines in software projects. For example, the customer can have a very different understanding of what is build than the software architect, who look at a more technical level to the requirements.

To solve both problems, User Stories in an agile setting are looking promising and are used a lot in practice, however, since there is a lack of scientific support on the effectiveness on these User Stories in a larger context in practice, there could improvements waiting to be found.

2. User Stories

Requirement management is a well-known topic in development processes. Since most of the delays are caused by the flaws in requirements analysis or project planning (Wang et al., 2014), it could be useful to improve processes related to these topics for companies. In practice, most of the attention is given to project planning, followed by requirements analysis, as observed by experts (Wang et al., 2014). Another measure indicated that companies with high customer satisfaction give priority to requirement analysis and testing.

Wang et al. (2014) indicated that User Stories play a prominent role in requirements elicitation, next to interviews and models (like business process models and goal models). Requirement representation is also often related to User Stories in practice, but User Stories do not serve many purposes in requirement analysis in practice.

User Stories are well described by Mike Cohn (2004), who defines User Stories as the description of functionality from a user or purchaser of a software system.

Although popular tools to support requirement management, such as JIRA (2018), Rally (2018) and ScrumWorks (2018), have supportive features for User Stories, there are no modules in place for analyzing User Stories for requirements.

2.1. Background of User Stories

User Stories are most common in agile development processes. Although the method and instrumentation related to User Stories differ from organization to organization, a common denominator can be identified within the requirement engineering. Such a common denominator is described by Cohn (2004), where User Stories belong to the following areas: *gathering, modeling, estimation and planning, acceptance testing* and *communication*.

The role of User Stories in **requirements gathering** is to provide a way of communication between different stakeholders (Cohn, 2004). Projects with a lot of requirements are often defined by a set of User Stories, where a project manager can, for example, play a central role in transforming input from other stakeholders into User Stories.

Related to requirements gathering is the **modeling** area, where different roles, related to a feature or module are modeled. Since User Stories often serve a default format where a role is coupled to a requirement, they form a good input for modeling an application. The process of modeling can be automatized by using natural language processing techniques (Robeer, et al., 2016).

In **estimating project sizes** and making time estimates, User Stories can provide valuable input. Although most of the estimation and planning work is done by hand, there are some tools available to support this work which are listed in *figure 2.1*.

	Planbox	tinyPM	ScrumDesk	Agilo for trac	VersionOne
Low-level iteration planning					
- Decomposition of a user story into tasks	+	+	+	+	+
- Task responsibility management	+	+	+	+	+
- Estimation of task duration and remaining time	P	+	+	+	+
Progress tracking					
- Velocity tracking	+	+	+	+	+
- Chart of planned and actual velocity	–	+	+	+	+
- Release burndown chart	–	+	+	–	+
- Release burndown bar chart	–	–	+	–	+
- Iteration burndown chart	+	+	+	+	+
- Daily burndown chart	–	+	+	–	+
- Additional features	+	+	+	+	+

***Figure 2.1**: A comparison of five popular tools to support low-level iteration planning (Dimitrijević et al., 2015)*

When a system is **tested for acceptability**, the system is evaluated for compliance with the business requirements and whether the system is acceptable for delivery (Ammann & Offutt, 2016). In an agile development environment, User Stories are often used to design these tests, since User Stories provide an easy to read overview of the systems requirements. Moreover, when there is no documentation about requirements available in existing systems, User Stories are sometimes created to structure the acceptance testing process (Ammann & Offutt, 2016).

The final area to which User Stories belong to is the field of **communication**. While communication is an abstract term in the process of developing software, it is mostly applied to exchanging requirements between different stakeholders (Cohn, 2004). A customer, for example, might want to know which features are going to be implemented in an easy to understand language, while a developer wants to know can make use of User Stories in the implementation process.

2.2. Core Theory

The academic interest in User Stories has resulted in common activities and components, that can contribute to the analysis of User Stories for an academic perspective. Common researches methods, like textual analysis and expert interviews, are used to find common grounds in the practical application of User Stories.

2.2.1. Common Ground in User Stories

Common grounds in the practical usage of User Stories can be found in activities related to User Stories and the structural components of which a User Story consists.

2.2.1.1. Activities Related to User Stories

Lucassen et al., (2015) extracted three activities related to User Stories; **creating**, **prioritizing** and **ensuring quality**.

Writing down requirements is in some cases not more than a short description of a required functionality. However, when a semi-structured template is used to compose a User Story, a door is opened towards a feature description in natural language which can also be interpreted by a system. Because of this practical advantage, templates for User Stories are gaining momentum (Lucassen et al., 2016). The default template for constructing User Stories within this study can be described as:

*"As a **<role>**, I want **<goal>**, so that **<benefit>**"*

In this template, a role can refer to a person interacting with the system, the goal refers to the desired outcome of this interaction and benefit serves as an optional parameter to include an advantage of the interaction, (see also *User Stories Components*).

Next to writing User Stories, project managers or customers can also **prioritize** them. Since customers often have the best position to express the desired features of a project, they are commonly involved in prioritizing User Stories. In most cases, this prioritization is based on the question which requirement adds the most value to the organization (Cohn, 2004).

Related to both composing and prioritizing User Stories is the **quality assurance** of a story. Since errors in requirements contribute significantly to the total software errors (Basili & Perricone, 1984), a quality framework is a necessity to compose informative User Stories which give a well-formed overview of a project. Moreover, when high-quality User Stories are constructed, the textual analysis of these stories becomes more accurate, which can improve requirements analysis (Lucassen et al., 2016).

2.2.1.2. User Stories Components

To ensure high-quality User Stories, there are three characteristics of a User Story:

- A User Story should contain a **short description** of the story used for planning.
- Conversations about the User Story should be in place to discover the **requirement details**.
- The story should contain **acceptance criteria** (Cohn 2004).

These characteristics are translated into four different components that make a quality User Story: a **format**, a **role**, **means** and an **end** (Lucassen et al., 2015).

A format can be described as a predefined template or skeleton to depict the conceptual format of a story. A format serves as the foundation to create a clear relation between role, means, and end (Lucassen et al., 2015).

A **role** defines a single actor per User Story. A role can be part of a larger hierarchy, for example, an "editor" is a type of "user" (Lucassen et al., 2015).

Means describe the main goal of the User Story, which is displayed as desire (depicted in a template as "I want" or "am able"), also the action related to the requested feature can be part of the mean (Lucassen et al., 2015).

The **end** of a User Story can serve three different purposes: clarification of the means, form a bridge to another functionality and the addition of a quality measure to the User Story (Lucassen et al., 2015).

2.2.2. Conceptual Model

Next to extracting characteristics and components related to User Stories, Lucassen et al. (2015) also found a relationship between those components which form the basis of a conceptual model for User Stories, see *figure 2.2*.

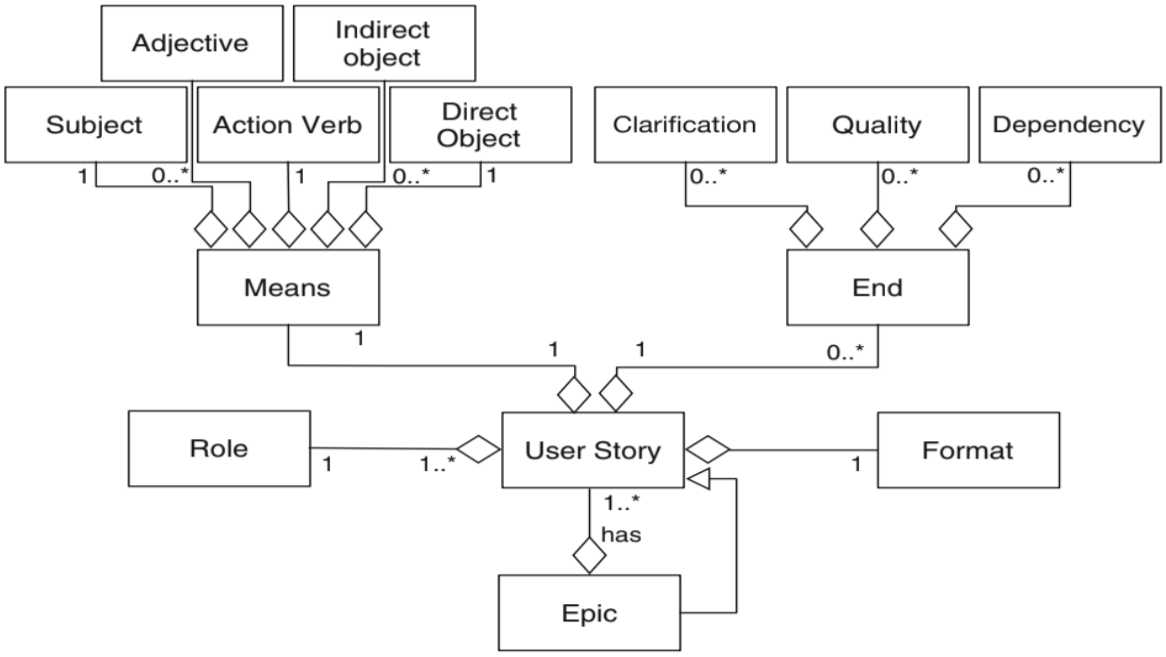

***Figure 2.2**: a conceptual model for User Stories (Lucassen et al., 2015)*

The conceptual model shows that every User Story should have one role, one format, and one means-component. The end-component, however, can occur multiple times in a User Story, for example when the end describes a quality aspect and a dependency. Since user stories can have a common theme or the features related to different stories can be part of the same module, the term Epic is also included in the model. An Epic depicts a large User Story, which can be broken down into smaller implementable User Stories (Lucassen et al., 2015). In practice it is also common to break down a User Story into smaller implantable tasks, tasks are not well-known jet in the scientific community, so there no task-element

included in the conceptual model. The task-gab is also further investigated during this research and can be found in section *2.5 Refinement to Backlog Items*.

2.2.3. Templates

A well-known format that is used to describe User Stories has been described by Lucassen et al (2015):

"*As a **<role>** I want to **<goal>** so that **<reason/benefit>***"

For example:

*As a **project manager**, I want to **visualize requirements**,*
*so that I **can communicate about them with customers**.*

There are many other formats used in practice, summarized by Wautelet et al. (2014). In order to find common ground in multiple formats, Wautelet and colleagues conducted a research where they found 65 different formats to describe User Stories, these formats could then be analyzed were three dimensions where distinguished; **WHO**, **WHAT** and **WHY**.

The **who-dimension** can refer to a role, user or actor. According to Wautelet et al. (2014), a role refers to an abstract characterization of the behavior of a social actor while a user is a person that uses the systems functionality and an actor is an entity that carries out actions to achieve goals. While the term actor is used to describe the who-dimension, it has no semantic relevance since it is closely related to role, therefore Wautelet et al. excluded the term actor. Also, the user is excluded since it is only used as an instance of a role. Therefor Wautelet et al. concluded that the who-dimension only consists of the role-term.

Wautelet et al. (2014) extracted four terms related to the **what-dimension**. The candidates where: Goal, Feature, Functionality, Capability, Task, and Activity. The goal is subdivided into a hard-goal, which is a condition or state that the stakeholder would like to achieve, and a soft-goal with is a condition or state that the actor would like to achieve. Feature and functionality were semantically to closely related so functionality was merged in the term feature, but for the what-dimension, the feature-term was too high-level so Wautelet et al. dropped the term anyway. Furthermore, task and activity where overlapping, so Wautelet et al. excluded the activity-term. So for the what-dimension, Wautelet et al. ended-up with hard-goals, soft-goals, capabilities, and tasks.

The **why-dimension** contains the same elements as the what-dimension (hard- and soft-goals and tasks) and also contains capabilities. Since capabilities, are not necessarily expressed in a direct manner, like in the what-dimension, Wautelet et al. posed to merge the capabilities-term into tasks.

By making use of a template, low-level User Stories can be used to construct a higher level goal net model (Lin et al., 2014). With a goal net model, it becomes more easy to communicate with stakeholders, enabling a Goal-Oriented Requirement Engineering also known as GORE (Anwer, & Ikram, 2006), where User Stories are clustered into high-level goals, see *figure 2.3*.

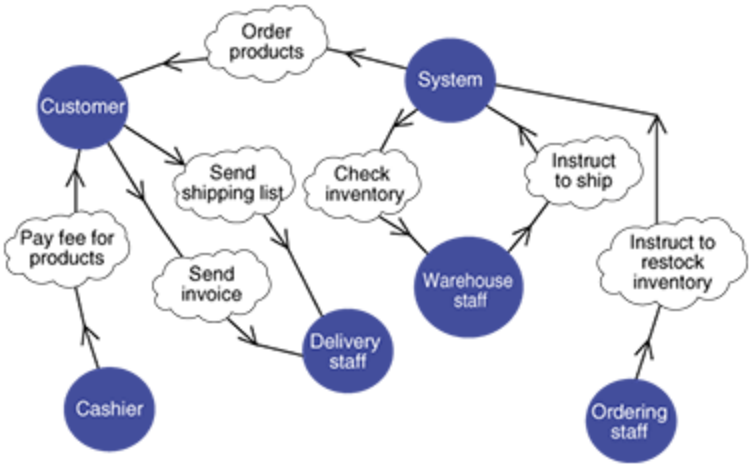

***Figure 2.3**: An example of a Goal-Oriented Requirement Engineering (GORE) diagram (Ntt-review, 2018).*

Another application of User Stories which makes use of the proposed template is generating test-script (Landhäußer & Genaid, 2012), where a test script is contracted by using three elements: preconditions (**Given**), actions (**When**), and expected results (**Then**). In their paper, Landhäußer & Genaid present a tool that builds an ontology to link User Stories with code-artifacts, creating reusable test-steps in the testing phase.

```
Feature: Sign up

  Sign up should be quick and friendly.

  Scenario: Successful sign up

    New users should get a confirmation email and be greeted
    personally by the site once signed in.

    Given I have chosen to sign up
    When I sign up with valid details
    Then I should receive a confirmation email
    And I should see a personalized greeting message
```

***Figure 2.4**: Example of behavior-driven development, using the given-when-then- template (Wynne, Hellesoy & Tooke, 2017)*

2.2.4. Quality Framework

Even in the early day of development, scientists started to realize the significance of requirement management on software projects (Basili & Perricone, 1984). Next to improving techniques and methods related to requirement management, also quality frameworks for these techniques found their way into the development process. For methods related to User Stories, this meant that researchers like Boehm (2000) distinguished four requirement engineering criteria: **completeness**, **consistency**, **traceability**, and **testability**.

Although the criteria from Boehm (2000) serve as a good starting point, his criteria are too general to apply on a specific method like User Stories. A more specific alternative came from Wake (2003), who used the INVEST (Independent, Negotiable, Valuable, Estimable, Small, Testable) verification framework for agile requirements and applied it to stories, where tasks from these stories should meet the SMART (Specific, Measurable, Achievable, Relevant, Time-boxed) criteria, also described by (Kavitha, & Thomas, 2011).

On the User Story perspective, Heck and Zaidman (2014) proposed a different approach by using a verification framework for agile requirements containing the criteria completeness, uniformity and conformance and expand these criteria to ensure the quality of User Stories. According to the expansion, posed by Heck and Zaidman, a User Story should contain the basic elements (role, activity and business value). Instead of summary and description, the required elements were acceptance criteria or acceptance tests to verify the story instead of rationale, optional elements can be included if the team could agree to more detailed attachments to certain User Stories for higher quality, the stories uniformity is met by following the standard of user voice form and lastly, the User Stories should follow the INVEST criteria es described by Wake (2003).

Lucassen et al., (2015) took the quality frame even a step further by looking at quality criteria from different angles, **syntax**, **semantics**, and **pragmatism** which are described in the Quality User Story (QUS) frame, see *figure 2.5*.

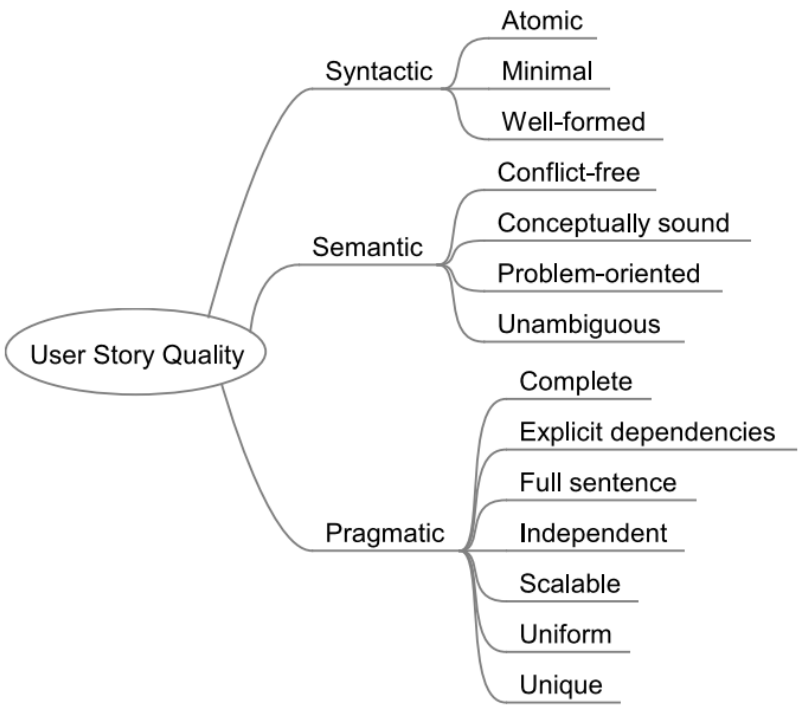

Figure 2.5*: Schematic representation of QUS (Lucassen et al., 2015).*

In order meet the QUS **syntax** criterium, a User Story should be authomic (describes only one feature), minimal (limited to a role, means and ends) and well-formed (describes at least a role and a means).

The **semantic** QUS criteria state that a User Story is conflict-free (not inconsistent with any other User Story), conceptually sound (means only express a feature and ends only express a rationale), problem-oriented (describe a problem and not a solution to it) and unambiguous (just one interpretation).

Lastly, **pragmatism** QUS criteria are expressed as complete (set of User Stories should result in a feature-complete application), explicit dependencies (linkage to unavoidable, non-obvious dependencies), full sentence (well-formed full sentence), independent (self-contained and is avoiding inherent dependencies on other User Stories), scalable (coarse-grained requirements that are difficult to plan and prioritize are avoided), uniform (consistency in the User Stories template) and unique (duplicates are avoided).

In addition, Lucassen et al., (2015) also provided some guidelines in writing User Stories by providing a tool for an automated quality check, AQUSA (Automated Quality User Story Artisan).

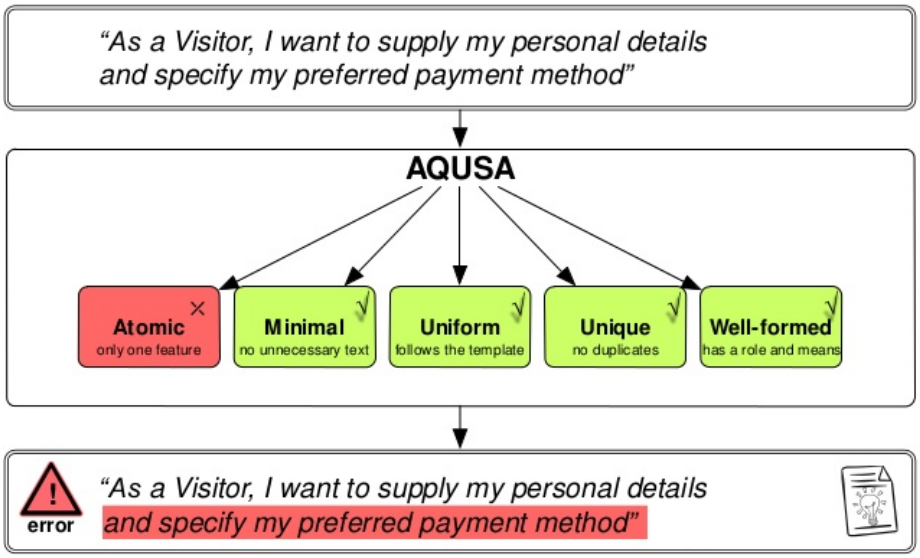

***Figure 2.6**: Impression of the AQUSA tool to check the quality of a User Story (Lucassen et al., 2015)*

As shown in figure 2.6, the AQUSA tool displays the User Story and provides feedback if the story does not meet the QUS criteria.

2.3. Applications For User Stories

The practical applications of User Stories reach beyond the description of requirements. In the world of natural language processing and automated clustering, there are more applications for User Stories in a development setting. Applications like time, effort and project size estimation receive an increasing amount of academic attention.

2.3.1. Time Estimation

Story Points are often used to make time estimates for a project. Story Points were introduced by Wilensky (1982) to characterize text that describes situations that constitute stories, based on content. Cohn (2005) adopted Story Points in his methods for time estimates by taking the sum of the Story Points assigned to each User Story that the team had completed during a sprint. Next to Story Points, velocity is also often used to estimate the team's effort based on Story Points and to indicate the team's rate of progress (Coelho, & Basu, 2012).

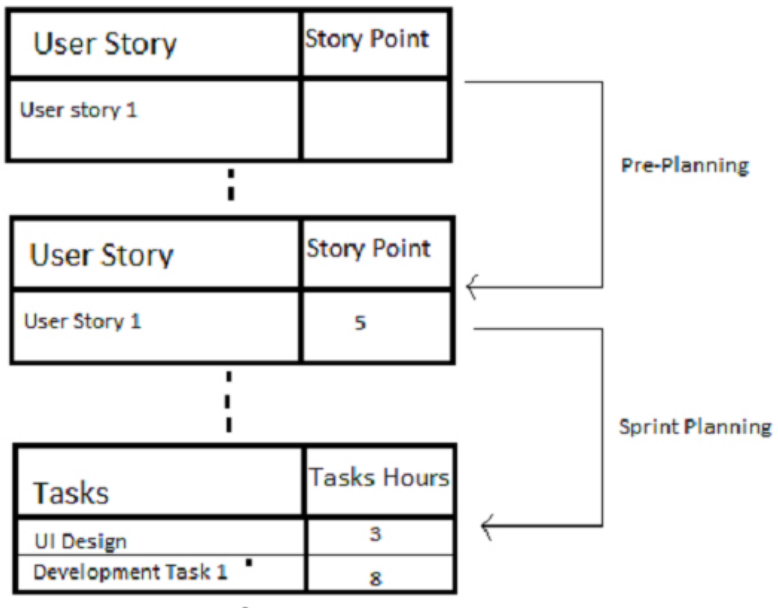

***Figure 2.7**: Example to estimate User Stories with Story Points (Tothenew, 2018).*

2.3.2. Project Size Estimation

There are different methods in existence to make a project size estimation, Coelho & Basu (2012) described four of them:

1. **Source Lines Of Code** (SLOC) where the number of lines of code serves as the main input for estimation.
2. **Function Points Analytics**, which is an estimate based on multiple parameters like external in- and outputs or inquiries, logical files and interface files which are counted manually.
3. **Use Case Points** methods are based on object-oriented methods, so there has to be a UML design in order to make this estimation.
4. **Story Points** estimates where a development team assigns points to every User Story based on the complexity and the expected duration of the implementation process. To work with Story Points, the team should have experience in time estimation and have access to historical data.

A proposed technique to make team estimations is Planning Poker (Mahnič & Hovelja, 2012), where every team member indicates the number of hours a specific task would cost, where only numbers from the Fibonacci sequence can be used. These estimations are then discussed by the team.

Ali, Shaikh and Ali (2016) looked at Story Points from a wider perspective and compared them with the Wideband Delphi technique, Component Estimation and Function Point Estimation where they showed that using Story Points gave the most accurate estimates, since there are certain metrics captured in a story (like weight, age, and complexity), its size could be estimated reliably.

2.3.3. Effort Estimation

Related to time and size estimates, project effort estimations (Ahn et al., 2003) indicates the project size and time in more general terms. Effort estimation in an agile context is described by Schmietendorf, Kunz, and Dumke (2008) who also used Story Point as the basic measure, which would lead to a study of Panda, Satapathy, and Rath (2015) to automate the estimation process by using different types of neural networks.

2.3.4. Requirement Analysis

During the refinement phase of an agile process, product backlog items are prepared for the next sprint, also known as backlog grooming (Kniberg, 2011). For this phase, there are multiple techniques used in practice like a storyboard, focus groups and card sorting (Lopes et al., 2017), which all depends on sorting a set of User Stories by combining knowledge of different experts.

Finding duplicate User Stories can also be automated by textual analysis, where Barbosa, Silva, and Moraes (2016) compared different techniques. The study of Barbosa, Silva, and Moraes (2016) demonstrated that the semantic similarity measures were preferable to measures based on frequency and on the occurrence of terms. The proposed techniques of analysing User Stories where: semantic similarity measures using WordNet, WuP similarity measure (also based on wordnet) (Pedersen, 2010), Lin similarity measure (based on semantic contents of words) (Lin, 1998), Lesk-A relationship measure (by finding overlap in the definitions of words) (Lesk, 1986).

A lower level of requirement analysis can be realized by looking at tasks, which contain information about the implementation of a User Story. Together with ontological information and natural language processing, tasks can form bridges between backlog items, making automated analysis possible, see section *2.5 Refinement to Backlog Items*.

2.4. Industrial Perspective

Since the field of requirement management is evaluated from practical problems, it could also be useful to view User Stories in a more practical sense.

2.3.1. Practical Challenges

Related to requirement engineering, Wang et al. (2014) summed up challenges which the industry has to overcome in order to successfully complete large software projects. These challenges are *requirements elicitation, requirements representation and documentation, requirements analysis* and *requirements management*.

In agile practice, **requirements elicitation** is often in the form of a discussion between group members with a customer. Although User Stories are being used often in this context (about 90%), interviews and models are still very popular (Wang et al., 2014). While User Stories do have the advantage of being

small, to the point and easy to understand, they could contain flaws since it is a representation of the requirements of a customer. By using structured interviews or prototyping, the requirements of the customer could be discussed in more depth while modeling techniques also present a more structured way of displaying an application in a more technical manner, which could be an advantage for the development team.

The **representation and documentation** of requirements is also a challenge where User Stories could be of help, but also, in this case, there are alternatives, for example, Use-Cases. Wang et al., (2015) used the technique of Use-Cases to build an automated way of creating Test-Cases, by using natural language processing. Although the accomplishment of Wang et al is notable, the techniques of natural language processing are not often used by the industry, where the more common means of documentation are still found in manual writing wiki-pages and building models. It is therefore not surprising that most common tools are still Microsoft Excel, Word and Visio, sometimes extended with ScrumWorks and Testlink (Wang et al., 2014).

Use Case: Get paid for car accident
Design Scope: The insurance company ("MyInsCo")
Primary Actor: The claimant
Main success scenario
1. Claimant submits claim with substantiating data.
2. Insurance company verifies claimant owns a valid policy
3. Insurance company assigns agent to examine case
4. Agent verifies all details are within policy guidelines
5. Insurance company pays claimant

Extensions:
1a. Submitted data is incomplete:
1a1. Insurance company requests missing information
1a2. Claimant supplies missing information

Figure 2.8*: Example of a Use Case (Adolph, Cockburn & Bramble, 2002)*

Requirement analysis is an interesting challenge for the industry since there is still a lot of manual work involved with the analysis of requirements. Two prominent tasks are the comparison of process flows and discussing existing requirement with the customer (Wang et al., 2014). Process flows of a set of scenarios have to be checked for inconsistencies and incompleteness between requirements. In practice, there is often a person or team responsible for comparing process flows and discussing requirements with customers. However, when the number of requirement increases, it becomes nearly impossible to read through all requirements.

Lastly, **managing requirements** is related to documenting changes in requirements. A technique related to requirement management is a Change Table, which consisted of details and effects of requirement changes, project risks, solution plans (Wang et al., 2014). There are also tools available to highlight updates, deletions, and additions in existing requirements (such as JIRA (2018), Rally (2018) and ScrumWorks (2018)) but these tools do not have a lot of analytical capacity, so a team is still responsible for doing most of the thinking.

2.3.2. Future Expectations

Conducting research on industrial standards are like shooting on a moving target. Since the industry is continuously updating its existing management strategies, it could be difficult to make predictions about the future need for requirement management. However, Kassab (2015) studied the industry and discovered some interesting patterns.

In requirement elicitation, Kassab (2015) discovered that User Stories are gaining popularity while prototyping is declining but still very popular. Representation of requirements is mostly done by using a semi-formal (UML) or informal (whiteboard) representation. While requirement inspection is still conducted, the process of inspection is moved from a formal inspection protocol to an ad-hoc walkthrough because of the exceeding numbers of requirements and the movement of responsibility to development teams. Supportive tools are also finding their way to the development process, where Jira and Microsoft Visual studio take the lead. Lastly, in effort estimation, expert judgment is still a popular method which is adapted in an agile manner. Group estimations and story points are also popular in agile environments, where story points in combination with planning poker showed better prospects in comparing studies (Usman et al., 2014).

2.5. Refinement to Backlog Items

Additional to the main topic of this study, a study is conducted to find structures in tasks defined by user stories. In practice, a scrum team will define specific tasks from a given User Story to make the development tasks more specific.

There is not much research dedicated to tasks, because they are not used in the rest of the development process, and often ignored when a project is reviewed. However, these tasks do contain additional information about the code, architecture, and documentation of a system.

In this study, a template is proposed to define tasks so future reach could be conducted in analyzing tasks to make new ontologies based on tasks.

For example, tasks can start with the noun *"update"*, which indicates that there is some source code in existence with can be associated to the task (and thus also with the corresponding User Story). A similar reasoning can be applied when a task starts with "*delete*" or "*remove*", where there must be some code in existence which is deprecated or non-functional and needs to be removed. Moreover, when a task starts with *"add"* or "*create*", it could be deduced that new classes or functions have to be written, so in these cases, it might be helpful to pay extra attention to multiple duplicate implementations between different teams to prevent redundant development work.

2.6. User Story Summary from Literature

In this section, I have tried to sketch an outline regarding the characteristics that would describe the requirements and architectural demands related to User Stories in an integrated development environment. The primary challenges of requirement management can be found in incomplete and/or hidden requirements, communication flaws between development teams customers and moving targets. Practical solutions can be found in different characteristics of supportive features, these features are summed by subject as User Stories:

2.5.1. Template

ID	Description
UT-1	As a product leader, I want to apply different User Story templates so that existing development methods are supported.

***Table 2.1A**: Requirements related to templates.*

2.5.2. Quality Framework

ID	Description
UQ-1	As a product manager, I want to write User Stories according to a quality framework so that the User Stories fulfill the defined quality criteria.
UQ-2	As a product manager, I want to select a User Story quality frame so User Stories fulfill the preset quality frame.

***Table 2.1B**: Requirements related to a quality framework.*

2.5.3. Estimations

ID	Description
UE-1	As a scrum master, I want to automated **time estimates** so manual work is reduced and estimates are more precise.
UE-2	As a scrum master, I want to automated **project size** estimates so manual work is reduced and estimates are more precise.
UE-3	As a scrum master, I want to automated **effort estimates** so manual work is reduced and estimates are more precise.
UE-4	As a scrum master, I want to automated **requirement analysis** so manual work is reduced and estimates are more precise.

***Table 2.1C**: Requirements related to estimations.*

2.5.4. Practical Challenges

ID	Description
UP-1	As a product manager, I want support for interview techniques so I can acquire in-depth information regarding requirements.
UP-2	As a product manager, I want to import requirements in Excel, Word, Visio, ScrumWorks, and Testlink so that I can work with familiar tools.
UP-3	As a product manager, I want to compare of process flows so I can distinguish duplicate requirements.
UP-4	As a product manager, I want to discuss existing requirement with customers so I can fulfill the customers' expectations.
UP-5	As a product manager, I want to track changes in requirements so backtracking from future adjustments is possible.

***Table 2.1D**: Requirements related to practical challenges.*

2.5.5. Future Expectations

ID	Description
UF-1	As a product manager, I want an automate UML representation of User Stories so the representations are more concise with less manual handlings.
UF-2	As a product manager, I want an automated informal representation of User Stories so the representations are more concise with less manual handlings.
UF-3	As a product leader, I want to ad-hoc walkthrough requirements so I can inspect the requirements.
UF-4	As a product manager, I want to import User Stories in Jira and Microsoft Visual studio so I can interact with familiar tools.
UF-5	As an expert, I want to make a judgment on requirements so that I can make adjustments.
UF-6	As a product manager, I want to automate story points assignments so project estimates are more concise.

***Table 2.1E**: Requirements related to future expectations.*

3. Industrial Trends In Agile RE

User Stories have made their way into the development process of companies (Lucassen, 2016) and their adaptation is evolving (Kassab, 2015). As recommended by Cohen (2004), User Stories are often integrated into an agile software development setting. However, there are multiple trends in the software industry that could be taken into consideration when developing tools to support User Stories. This section is dedicated to trends in the software industry that could influence the position of User Stories within the requirement and development cycles. For this chapter, three subjects have been selected that could be of importance in the future of Agile requirements engineering. The first subject is related to established methods, where the ecosystem of different methods might evolve to a different method within Agile. The second subject is dedicated to industrial transitions since shifting to different methods could influence the role of Agile within organizations. The last subject is related to architecture which is also a critical step finishing a development project.

3.1. Established Methods

Different trends are extracted from multiple studies. In order to give some insights into the diversity of development approaches, there are ten existing techniques analyzed in this section. After a short description and their potential trends, the key characteristics are described in the light of User Stories.

3.1.1. Agile Development Methods

Agile methods are very popular in the industry (Hron, & Obwegeser, 2018). Some sources even claim that 95% of software-related organizations have adapted scrum in their development process (SCRUM ALLIANCE, 2015).

The two most known agile methods are scrum and extreme programming (XP), the central theme in scrum is successful software development while in XP the focus is more on the project level (Salo & Abrahamsson, 2008). According to Dingsøyr and Lassenius (2016) XP is dropping in popularity while scrum shows a slide increase.

The focus on agile methods is becoming more value-centered where continuous development is also gaining popularity (Dingsøyr & Lassenius, 2016). The increased interest for continuous development could be related to an increasing demand of real-time data analysis (Halpern, 2015), with Big Data and the increasing interest in predictive and prescriptive analysis, Agile practices are also adapted by BI practitioners (Larson & Chang, 2016).

Shifts in Agile methods also highlight its limitations, like support for distributed development environments, support for subcontracting, reusable artifacts, development involvement in large teams, support for safety-critical or large and complex software (Turk et al., 2014).

3.1.2. Continuous Software Engineering

Continuous software engineering has gained popularity in multiple fields related to the development process. According to Fitzgerald and Stol (2014), continuous trends are evolving in Business Strategy and *Planning, Software Development, Operations and Improvement,* and *Innovation.*

Planning involves multiple stakeholders where dynamic open-end-artifacts evolve in a changing business environment, with a tighter integration between planning and execution (Fitzgerald & Stol, 2014). While Myers (1999) created a framework to support continuous planning and Knight et al. (2001) described a system to for continuous scheduling and planning activities (Casper), Lehtola and colleagues (2009) identified and used these practices to link business decisions and RE in a continuous planning.

Multiple steps in the **development process** have the possibility of becoming a more continuous process, verification and testing are already well adapted in most companies so developers can automatically test their software and deploy it on a pre/release brache. Security, compliance and even delivery are also candidates for a more continuous process. For critical systems, it could be of great importance to monitor security and make quick updates when threatening situations emerge. Where for compliance and delivery release-bottlenecks can slow down the process of selling new functionality to customers (Fitzgerald & Stol, 2014).

In **operations**, the demand for run-time monitoring becomes clear in the light of running cloud services, where service level agreements (SLAs) form a direct incentive to increase the availability of a service (van Hoorn et al., 2009).

Improvements in a lean perspective, where decision-making is data-driven, could have great advantages when quality improvements are done incrementally and with continuous innovation, the process of planning and development can be more adapted to an evolving market (Fitzgerald & Stol, 2014).

3.1.3. User-Centered Development

It is also common to involve the user more in the development process. Salah, Paige and Cairns (2014) review 71 paper related to Agile and User Centred Design Integration (AUCDI). From these papers they distingue a number of challenges related to involving users in the agile development process.

- Lack of Time for Upfront Activities
    - which can result in a UI that is disjoint, piecemeal and lacking a holistic, coherent, and overall structure and vision
    - Up-front design is a separate pre-development period that is used in Agile projects for eliciting requirements, understanding users,
    - Upfront design can also be used by the development and quality assurance
- The difficulty of Modularization/ Chunking
    - difficulty in determining the right chunk size and the right amount of interaction design work per iteration

    - the difficulty of maintaining the ordering dependency between design chunks
    - difficulty in differentiating between user experience design activities that contribute to breadth or depth
    - interaction designers adopt a holistic view to interaction design
- The difficulty of Prioritizing UCD Activities
- Optimizing the Work Dynamics Between Developers and UCD Practitioners
    - Sharing an Understanding of Users
    - Sharing an Understanding of Design Vision
    - Synchronizing Efforts of UCD Practitioners and Developers
- Performing Usability Testing
    - Method of Usability Testing
    - Scheduling Usability Testing
    - Accessing Users for Usability Testing
    - Shorter Time to Iterate Design
- UCD Practitioner Workload
- Lack of Documentation

With these challenges in mind, the dependence of good tooling could grow, since well structures workflows and transparent information exchange can lighten the challenges like modularization, testing and practitioner workload.

3.1.4. Mob Programming

In contrast to decentralized software development, the trend of mob programming is also gaining popularity. Kattan, Oliveira, Goldman, and Yoder (2017) describe mob programming as comparable with pair programming, where two persons work on the same computer at the same time in the same place, see *figure 3.1*. The main difference between mob programming and pair programming is that with mob programming, an entire team is working on the same computer in the same place and time. Advantages of mob programming are easy knowledge sharing, faster learning, and satisfaction of the programmers.

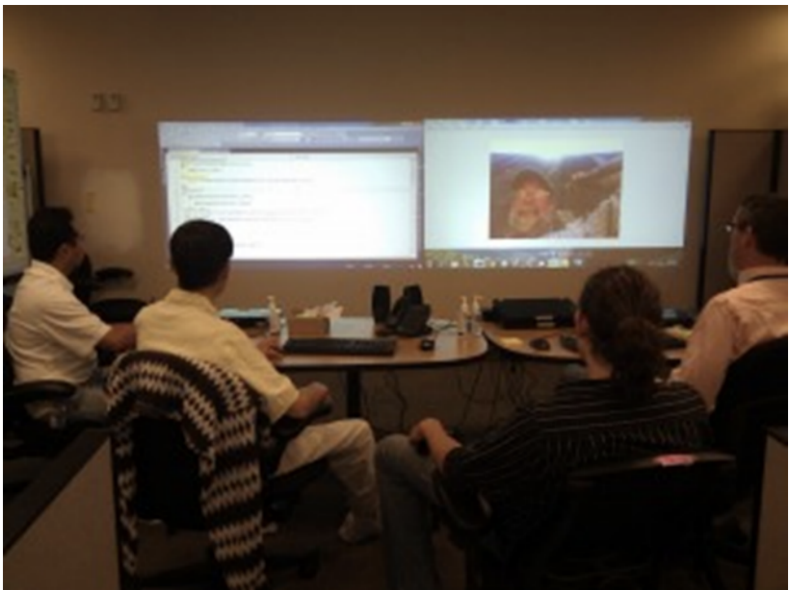

***Figure 3.1**: Practical impression of mob programming (Zuill & Meadows, 2016).*

3.1.5. Concept Releases

Concept-release is an incarnation of a product with a time aspect. A release consists of sprints, which on their turn contain backlog items. The release planning is coupled with multiple releases and sprints to transfer the content captured in User Stories. A backlog item delivers a feature or part of a feature. Backlog items can also be traced back to the source code. Test scripts are also related to the backlog items and source code but are not considered within the scope of this project. Traceability is also related to the domain concept, which states within which context the application is built, for example, “greenhouse application” or “hospital application”. Domain concepts are also related to User Stories, backlog items, and sprints.

3.1.6. Goal-Oriented Requirements Engineering

Goal oriented requirement engineering (GORE) can be a useful conceptualization to elicit, model, and analyze requirements which capturing alternatives and conflicts (Horkoff et al., 2017). In scientific literature, there has been an increasing research interest in GORE from 2008 to 2012 (Horkoff et al., 2017). There is solid evidence that GORE is feasible in general and when applied to inputs from industrial practice (Mavin et al., 2017). There are even papers where a Goal-Oriented Requirements Ontology (GORO) is presented (see *figure 3.2*), which is founded on the Unified Foundational Ontology (UFO) to represent relations and concepts in the GORE domain (Negri et al., 2017).

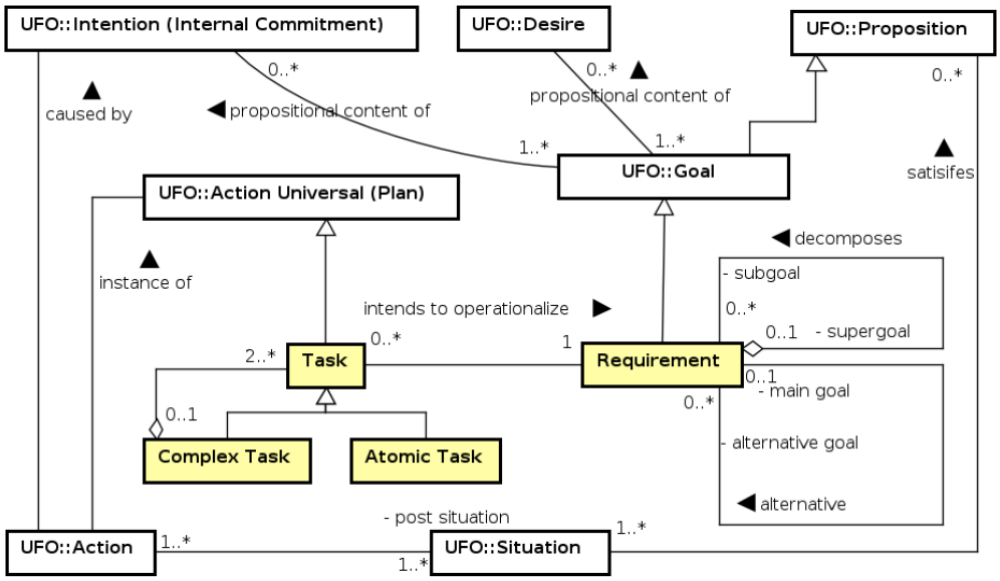

***Figure 3.2**: GORE fragment where the relations are highlighted (Negri et al., 2017)*

Although GORE looks promising, the number publications have been declining over the last few years (Horkoff et al., 2017) and there is also a lack of evidence that GORE can function in practice (Mavin et al., 2017) since practitioners seem to have little incentive to apply GORE. Furthermore, practitioners also seem to have difficulty with the exact concept of a goal, whereas a requirement is better understood (Mavin et al., 2017).

3.2. Industrial transitions in Requirement Engineering

3.2.1. Network Organisations

The phenomenon of network organizations is not uncommon, moreover, the number of network organizations is rising which leads to more scientific interest within the subject of using existing technologies to address the problem of communication between different stakeholders within a project. Schön, Thomaschewski, and Escalona (2017) investigated the current methods of requirement engineering by reviewing 27 studies about agile requirement engineering. In this study, they extracted some interesting characteristics about the approach of involving stakeholders in an agile development process. Most of the studies describe a direct involvement (77%), there are also many studies which describe a process to involve stakeholders (78%) and iterations during the development process is a common aspect in the majorities of studies (70%). The use of artifacts in requirement management is described by 93% of the studies and the requirement that documentation is understandable without further knowledge is described in 63% of the studies. From the artifacts described in the studies, *User Stories* (56%) is the most common one, followed by *Prototyping* (41%) and *Use Case* (26%). From this study, Schön, Thomaschewski, and Escalona concluded that building a shared understanding of the user perspective is not very well established. Furthermore, they identified four methodologies, Human-Centered Design, Design Thinking, Contextual Inquiry, and Participatory Design and identified

User Stories, prototypes, use cases, scenarios and story cards as artifacts for documentation of requirements in an agile environment.

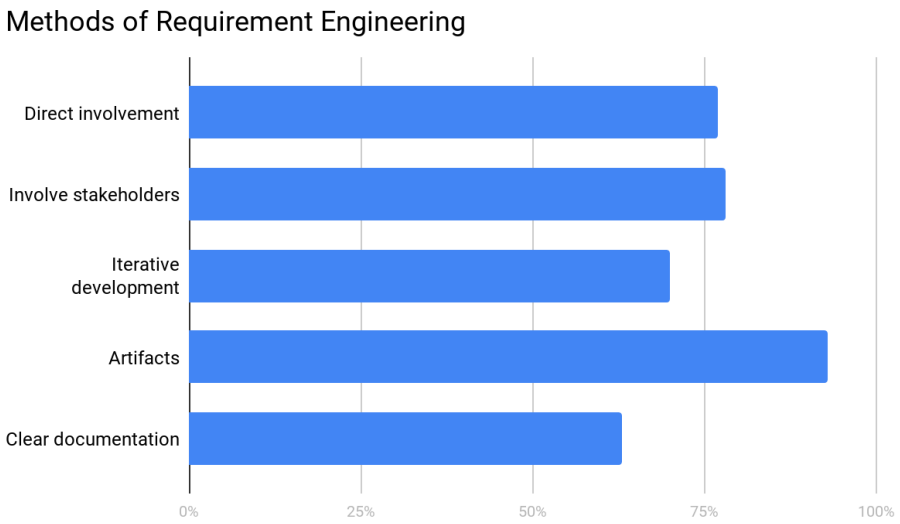

***Figure 3.3**: distribution of commonly used methods in Agile development (Schön, Thomaschewski and Escalona, 2017).*

For network organizations, like Gitlab, the sense of good communication becomes more prominent since workers have to overcome physical distances. To decentralize software development, new tooling becomes a necessity because communication between managers and developers could not always be done face-to-face. Within large-scale software projects the overall view becomes more abstract since it is impossible for one person to oversee every detail.

3.2.2. Decentralized Software Development

Another phenomenon within a more separated development approach is the usage of decentralized software development tooling. Layman, Williams, Damian, and Bures (2006) conducted an industrial case study regarding extreme programming within the context of distributed software development (DSD). extreme programming is characterized by its short programming horizon and its iterative methods. They concluded that customers and developers discussed feature details more common. Within these discussion changes in User Story and specifications were made over the entire project. Furthermore, technical questions were resolved more quickly and efficiently. They identified three communication practices:

- Email lists
- Project management tools
- Intermediary development manager

The main success factors of distributed development projects where:

- Development manager who advocates in two groups and forms a communication bridge between those groups.
- Short, asynchronous communication loops that can serve as a surrogate for synchronous meetings.
- A customer authority for resolving issues related to requirements.
- Visibility of the process to customers in order to guide the project synchronously with the development process.

For large scale organizations, the demand for decentralized projects rises. De Alwis and Sillito (2009) studied why software projects move from a centralized to a decentralized version control system (like Gitlab). They found the following reasons, decentralized development groups provide first-class access to all developers, they support atomic changes where it is more easy to automatic merge these changes, support can be improved for experimental changes and support is a disconnected operation.

While development teams are decentralizing and the customers play a more central part in the development process, the need for a central tool of measurement becomes a more prominent problem. While User Stories have a lot of promising properties, they are not widely implemented in most of the development environments, which is likely caused by the lack of scientific evidence of the effectiveness of User Stories is practice.

3.3. Prospects in Software Architecture

The field of software architecture has also gained popularity since the nineties amongst practitioners since it forms a effective basis reusability and consistency in the development process (Perry & Wolf, 1992). In order to visualize the architecture of applications, an architecture description languages (ADL) was used by Garlan (2000), where an ADL is a formal language to represent a software system. Malavolta and colleagues (2013) found that ADL's are mostly used to communicate about and analyze software systems, where the most popular of these languages is produced by the industry itself.

Malavolta (2013) and Clements (2010) indicate that three of the most used ADL's are UML, SysML, and AADL. While UML is the oldest and most popular of these three, it has not been developed to describe software architecture. SysML is a dialect of UML, so it shows similar diagrams. AADL is an architecture analysis language for performance-critical systems as found in the aerospace industry (Feiler, Gluch, & Hudak, 2006).

Since the industrial demands on complex applications are increasing, there is also research conducted to introduce alternative architecture languages. One of these languages, Utrecht Architecture Description Language (uADL), has a lot of promising features and could be a good candidate for future requirement engineering and software architecture. The uADL is designed to be simple but powerful and contains a set of different techniques to support multiple viewports (Jansen & van Rhijn, 2018). The techniques uADL supports are comparable with UML, SysML, and AADL but the number of unique elements and complexity of the notation is significantly lower, which could make uADL particularly useful in practice (Jansen & van Rhijn, 2018).

The different uADL-techniques can be ordered by six viewports, which are **context**, **functional**, **information**, **concurrency**, **development** and **deployment**.

The **context** viewport has to provides information about the context of the application, like entities who use the application or databases that store data from the application. The three techniques that can be used for this viewport are User Stories, Context Diagrams and Use Case diagrams. A Context Diagram is focussed on the external roles, related to the system (see *figure 3.4*) whereas a Use Case Diagram contains information about the actors.

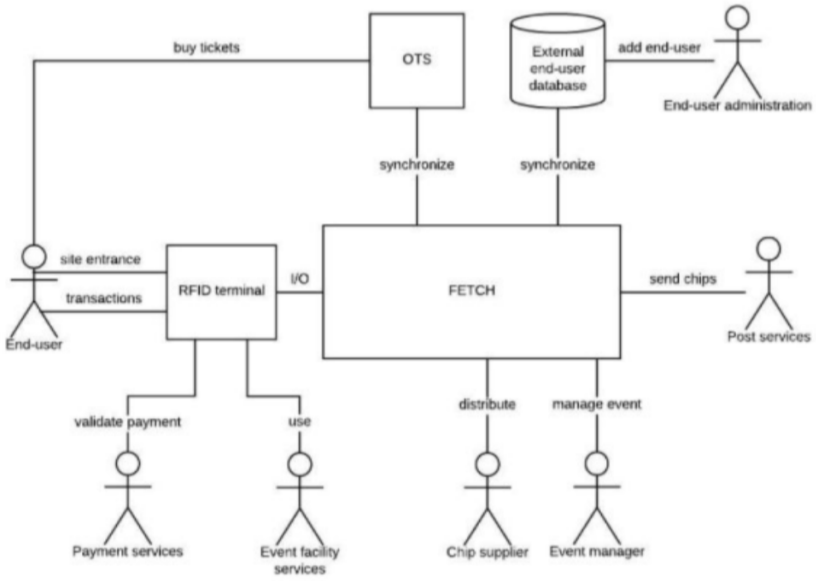

***Figure 3.4**: Example of a Context Diagram Diagram (Jansen & van Rhijn, 2018).*

Functional viewpoint techniques are Functional Architecture Model (FAM), feature diagram, scenario overlays and formalized FAM. A FAM represents the primary functionality of a software product, its main functions, and supportive operations see *figure 3.5* (Brinkkemper & Pachidi, 2010). A feature diagram presents the different modules of the FAM and the scenario overlays add scenarios of user behavior. A more formal way to describe the functional architecture of an application can be achieved by using a formalized FAM. scenarios can also be added to a formalized FAM, making it useful for constructing open Petri Nets.

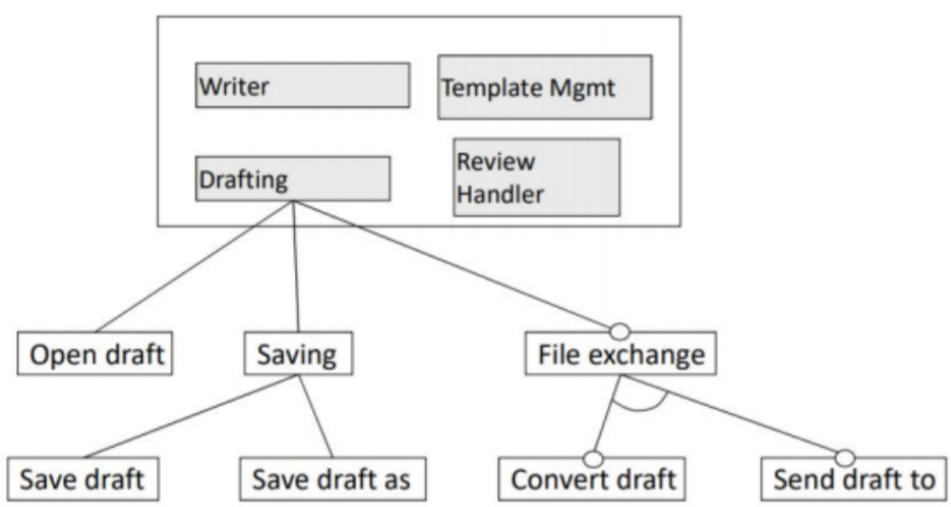

***Figure 3.5**: An impression of a FAM Diagram (Jansen & van Rhijn, 2018).*

The **Information** viewpoint only contains the Entity-Relation (ER) Diagram technique. An ER diagram depicts properties and cardinalities between the relationships of the entities that make use of the application.

Concurrency viewpoint consists of the open Petri net technique, where the features of a formalized FAM of an application as places and the overlays as a path of the Petri net.

The **development** viewpoint contains the more technically oriented Component & Connector diagram, which displays the software components as blocks and their connections as interfaces of an application.

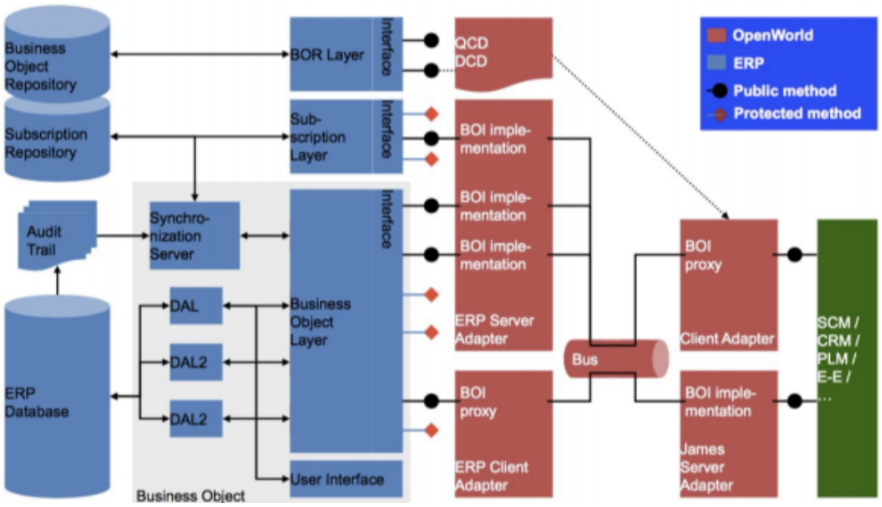

***Figure 3.6**: Impression of a Component & Connector diagram (Jansen & van Rhijn, 2018).*

Finally, the **deployment** viewpoint contains the product deployment stack technique where the dependencies are for an application are displayed, for example, the OS and virtual machine needed to run the run the application.

3.4. User Story Summary from Literature

In order to extract the requirements and architectural characteristics form the industrial trends and transitions, we have to take a step back and look at the greater picture of the software industry. Agile methods clearly prevail in the development process, although Agile methods have limitations on supporting distributed development and subcontracting, we see a trend emerging in the direction of network organizations and distributed development. Adaptations in Agile methods could be inevitable when these trends continue to grow. Another challenge can be the emergence of user-centered development, where upfront time estimations, modularization and prioritizing of tasks and usability testing are not well supported by Agile methods.

On the other hand, are continues release cycles gaining popularity, which is in line with an Agile pillar of putting more responsibility at a team. Also, the emergence of mob programming, where diverse teams cooperate closely to build a software product, takes a vision towards Agile adaptations.

The importance of architectural languages in the development process might also rise, with the emergence of both self-steering teams as distributed development teams. Since multiple workers with different backgrounds are involved in projects, which could become too big and versatile to be overseen by a single person, different viewpoint and ontologies between those viewpoints can be of help.

Certain key terms are reflected in all aspects of the development process (for example source code, test scripts, backlogs, user manuals, and conceptual models). In order to improve feature tracing and conceptual models, the reflection of key terms should be used to combine different views of the same features. A central way of communication should be adapted, analog to the architectural changes of a building, where technical drawing provide an overview of all technical aspect of the building. Related to these technical drawings, tooling should visualize conflict and address responsibilities within the development process.

3.4.1. Communication

ID	Description
UC-1	As a team-member, I want to communicate with other teams so redundant work is avoided.
UC-2	As a team member, I want to communicate with other team members so we can point everyone in the same direction.

***Table 3.1A**: Requirements related to communication.*

3.4.2. Architecture

ID	Description
UA-1	As a company, I want a reliable architecture method so distributed teams can cooperate.
UA-2	As a team member, I want an understandable view of the project so I can make informed decisions related to the project.
UA-3	As a product manager, I want to determine who can view which requirements so subcontractors can work on a shielded part of the project.

***Table 3.1B**: Requirements related to architecture.*

3.4.3. Requirements

ID	Description
UR-1	As a developer, I want to couple requirements to source-code so I can find existing code more easily.

***Table 3.1C**: Requirements related to requirements.*

3.4.4. Version Control

ID	Description
UV-1	As a team member, I want to scroll down in history so I can see why certain decisions are made in relation to updating a system.

***Table 3.1D**: Requirements related to version control.*

4. Software Development Kits

4.1. Existing Development Supporting Systems

Three different development supporting systems are selected for further investigation on where to implement the additional features of this study, based on their popularity of use in the industry. To measure popularity, the Alexa rank is used which represents the number of users visiting the product website per month. Applying the Alexa rank for development support systems, resulted in the selection of Bitbucket, GitHub, GitLab. All three systems use Git as a basis of an online version control system with additional tooling to support project management.

4.1.1. Bitbucket

Bitbucket is only commercially available and offers software on both client and server side. With a clientship of five million users worldwide, Bitbucket has an Alexa ranks of 937 (https://www.alexa.com/siteinfo/bitbucket.org). Since Bitbucket is a closed source commercial system, there is are no release binaries or release announcements available. The tooling and flexibility related to Bitbucket is very advanced, making Bitbucket a system used by large software companies.

4.1.2. GitHub

GitHub is the most familiar development support system and is mainly used by open source communities, small companies, and hobbyists. GitHub is free to use if the submitted code is deployed under an MIT license or another open license. In order to build software in a private environment within GitHub, a subscription fee must be paid. Because of its popularity, GitHub facilitates more than twenty-four million users and over sixty-nine million projects worldwide. GitHub has an Alexa score of 71 (https://www.alexa.com/siteinfo/github.com) and is by far the most popular development tool of these three. The popularity of GitHub might decrease in favor of GitLab since at the time of writing, GitHub has been acquired by Microsoft.

4.1.3. GitLab

GitLab is relatively new but has a lot of potention. It is very popular under small businesses and start-ups since GitLab provides a lot of free functionality for supporting the software development process. GitLab is compleatly open source and encourages communities to contribute to the development of their products. Because of the strong community they build-up with this way of working, the number of supporting tools grows rapidly as well as the number of users and projects hosted at GitLab. At the time of writing, the number of users has exceeded hundred thousand and the number of projects is above six hundred thousand. GitLab ranks at place 2866 at Alexa (https://www.alexa.com/siteinfo/gitlab.com).

4.2. Characteristics of Common Systems

A comparison is made between the three selected systems. This comparison consists of two steps, first, the basic usage statistics provide an overall picture which system is used most commonly by the industry. Secondly, a study is conducted to map characteristics of the development support systems, these characteristics of features are based on the requirements classification from the SWEBOK requirements classification (Bourque and Fairley, 2014).

4.2.1. Industrial Popularity

To provide an overview of industrial popularity regarding the selected SDK's, the number of users and projects from these systems is compared. The Alexa ranking also indicates the popularity of a system, by providing information about how popular a website is. Finally, the year since the system is operational can indicate how mature the system is.

Development Environment	Number of Users (in thousands)	Number of projects (in thousands)	Alexa ranking	Operational since
GitHub	24,000	69,000	71	2007
Bitbucket	5,000	Unknown	937	2010
GitLab	100	546	2866	2012

***Table 4.1**: an overview of industrial popularity regarding the selected SDK's.*

4.2.2. Feature Comparison

There are multiple functionalities extracted from the selected systems, these functionalities are compared using the SWEBOK requirements classification from Bourque and Fairley (2014). The SWEBOK classification contains the following aspects: Functional versus nonfunctional, high-level requirements versus emergent property, product versus process, requirement priority, requirement scope and volatility versus stability. With this comparison, the characteristics of the functionalities from the different systems can be extracted and classified.

Comprisement	**BitBucket**	**GitHub**	**GitLab**
website	**bitbucket.org**	**github.com**	**about.gitlab.com**
Commercial	yes	yes	yes

Open source	No	No	Yes
Main tasks	Issue tracking	Track and assign tasks	Powerful Issue Tracker: ● Group-level milestones ● Issue Boards ● Related issues ● Issue Weights
Third party integrations	Jira Software integration		● Time tracking ● Confidential Issues ● Burndown Charts
Main usage	Projects		Projects ● Move issues between projects
Delivery management	Built-in continuous delivery		Built-in CI/CD
Management functionality		Cards: reference every Issue and Pull Request	Deploy Boards
Development support		Notes: capture early ideas	● Todos ● Cycle Analytics
Requirements management			Epics
Integration with	● Confluence ● Jira ● Bamboo	● Slack ● Zenhub ● Travis CI ● Appveyor ● Codacy ● Codeship ● Code climate	● Jira
Custom integration	API	API	API

Table 4.2*: an overview of features of the selected SDK's.*

4.3. Conclusion

GitLab is chosen for further analysis and implementation because of the usability, its open character and its popularity amongst start-ups. The open character of Gitlab is visible in well supported online communities form Gitlab. It is possible to download most of the source code to build additional functionalities and to test the existing tooling. These properties make GitLab usable to test a new way of requirement management using User Stories.

4.3.1. Characteristics for Supporting Tooling

Certain key terms are reflected in all aspects of the development process (for example source code, test scripts, backlogs, user manuals, and conceptual models). In order to improve feature tracing and conceptual models, the reflection of key terms should be used to combine different views of the same features. A central way of communication should be adapted, analog to the architectural changes of a building, where technical drawing provide an overview of all technical aspect of the building. Related to these technical drawings, tooling should visualize conflict and address responsibilities within the development process.

5. Case Study

5.1. Interview approach

5.1.1. Goal Definition

Conduct a interviews with experts who are familiar with tooling related to Gitlab and User Stories. The goal of these interviews is to get insights in which functionalities are desirable in tooling to support User Stories.

5.1.2. Context Selection

- Off-line
- Professional
- Real problems
- General

5.1.3. Hypothesis Formulation

The software industry can provide practical insights to strengthen the literature review of this study.

5.1.4. Variables Selection

- Definition
- Governance
- Roles and Responsibilities
- Instruments
- Projects

5.1.5. Selection of Subjects

In order to get a diverse view on the industry, there are five companies selected for an interview, these companies differ on size and field of operation. Interviews are conducted with an employee from each of these companies, where also the role of each interviewee varies.

5.1.6. Choice of Design Type

For this study, a structured interview is chosen so subjects have the opportunity to explain reasoning and management choices in more details while the information related to the requirement engineering and architecture subjects are ensured.

In total, there are five interviews conducted to get an idea of the diversity of the software industry related to requirement engineering.

After the interviews had taken place, the audio files are transcribed and coupled with the literature.

5.1.7. Instrumentation

- Interview protocol
- Audio recording

5.1.8. Validity Evaluation

Internal and external validity are taken into consideration. The interviews are semi-structured, so there could be some divergence of the subject. Another possible weakness is that the same interviewer conducted all interviews. The interviews, however, are recorded and transcribed so future analysis can be conducted on the results. There are also some control questions included to test the knowledge level of the interviewees with respect to User Stories and requirement management.

External validity threats can be found in the absence replications, this study is explorative. The number of interviewees is also not suited for hard conclusions. Generalizability across situations, is safeguarded by selecting a diverse set of companies of different sizes, maturity levels, and different industries.

5.1.9. Experiment Design

5.1.9.1. Research Approach

In order to gain a better understanding of User Stories in practice, a series of interviews are conducted at various companies where employees with different functions and backgrounds will answer questions related to User Stories in practice and possible functionalities to support User Stories.

5.1.9.2. Data Gathering

- Five different companies
    - The company will be anonymized so instead of using the name of the company, a company code is used.
    - To provide an estimation of the size of the company, the total number of employees is recorded.
- One employee per company
    - The employees will be anonymized but their role within the company is included, to give an idea of their perspectives.

Firm code	Sector	Number of Employees	Interviewee Role
Company A	Financial	27,000	Project manager
Company B	Management	500	UX designer
Company C	Administration	1,500	Project manager
Company D	Outsourcing	50	COO
Company E	Games / Education	20	CEO

***Table 5.1**: General information about the companies where the interview took place.*

The following nodes are used to structure the interview and the questions.

- Definition
  - *How is a User Story defined within the company?*
- Governance
  - Policy & Strategy
    - *Are User Stories part of a strategic decision?*
    - *Are there policies related to User Stories?*
  - Task & Responsibilities
    - *Who is responsible to govern User Stories?*
    - *What task are related to User Stories within the organization?*
- Instruments
  - *Is there some sort of instrumentation or tooling used related to User Stories?*
- Projects
  - *Is there any project where User Stories are used?*

5.2. Interviews

The information from the interviews is described per topic; Definition of User Stories, User Story Governors, Roles and Responsibilities related to User Stories and Projects.

5.2.1. Definition

There are multiple definitions for User Stories and related terms that are used in practice. In this chapter different terms are described in a practical context derived from the interview at the five companies.

5.2.1.1. Usage of the Term User Stories

The term User Story is used in all interviewed companies, however, there are differences in what the term represents and how often and in which context the term User Stories is used.

At company A the term User Stories is used to communicate between stakeholders of a project, whereas in company C User Stories are a way to record desirable features. At Company B User Stories mainly clarify the needs of a specific group of people sharing a role within an organization.

The frequency of use can also differ, for example in company D User Stories are only used in 20 to 10 percent of the projects, since most of the projects consist of combining existing features.

The perspective from which User Stories are being used, differs between organizations as well, Company A often writes functionalities from a user perspective, whereas company C is more focused on the product and its feature, where the User Stories are an extraction of "Feature Stories" (see "Related Terms"), translated into product backlog items.

Changes in the definition of User Stories are common in Company C, where the definition of a User Stories closely related to its practical use in a project. Also in Company D, there is some change in the definition of User Stories since the previously used waterfall method has been replaced by a more Agile approach, recently.

5.2.1.2. Terms related to User Stories

From the interviews, there are some terms closely related to User Stories in the requirement engineering process, in this section the most frequently used terms are explained from a practical perspective. These terms are sorted from global to detailed project level: theme, vision, epic story, feature, backlog item (also known as task).

5.2.1.2.1. Theme

Within the process of defining a project, a theme is often extracted. A theme is considered to be the basis of a project which can contain customers needs or tasks from the board at Company A. In some cases, a theme can also provide input for other artifacts, at Company A, the theme of a project goes hand in hand with a vision, which serves as input for defining Epic stories.

5.2.1.2.2. Vision

In a default Agile process, the definition of a vision can be tricky, the project manager at Company D even states that the lacking of a process for communicating a clear vision to customers is the main downside of scrum. Since the development process is continuous, it is not clear where the project will stop, making it difficult to clarify in advantage what the exact scope of a project will be and how much a project will cost. At company A, a vision of a project is defined by the stakeholders and PCG. where a vision contains high-level solutions with business cases, a scope, architecture, and sourcing. When using this definition a vision could be part of a scrum process, since the exact details are not entitled.

5.2.1.2.3. Epic Story

In most cases, central themes serve as input for epic stories. At Company B, An epic is constructed from a theme, which is used before a user story is composed. When a project manager at Company B wants to add value for a specific group of customers, an epic can also be constructed as an overlapping theme. At

company A, a theme can also be split into different epics, which contain more specific information about the business deliverables. The ambition of a project is part of the theme and epics where also the go-live event is included at Company A. Company B, uses Epics to compose User Stories for a project, it is also possible that two Epic are merged or spit in the course of the process.

In coupling an Epic Stories to models, is difficult according to the manager of Company B, since the term models within the context of User Stories is vague. In contrast, at Company D, Epics are mainly used to represent modules. Although the name Epic is not often used within Company D, there is often a grouping of User Stories, which is presented to the developers in designing and building modules, where each module has a central topic.

5.2.1.2.4. Features

Company C uses an alternative term, which is related to User Stories, Feature Story, which is comparable to a User Story but has no fixed format and is mainly focused on a feature, which can be used as input to define User Stories so a Feature Story can also be compared with an Epic Story.

Within Company A, there is also a feature-based approach of User Stories but here the features are derived from a business refinement and IT refinement. Members of different scrum teams will meet to define the deliverables in form of features. These features contain the results of a more detailed analysis by the IT teams and consists of customer journeys, visuals and a Probabilistic Safety Assessment (PSA). The business side is also reviewed, where legal issues, risks, and compliances are included.

The linkage of User Stories and features are clearly noted in the interviews at interviews Company A and Company C. At Company B, however, a link between User Stories and features is not clear, since features allow some vagueness in their practical application.

5.2.1.2.5. Backlog items (tasks)

In most cases, User Stories are distributed amongst different scrum teams, where backlog items are written to specify the different tasks related to the User Stories. These tasks help scrum teams in their planning but can also serve as a way to measure progress and time estimates.

The process of defining tasks and link them to User Stories is often done by hand, at Company A and Company C the tasks are defined by a product manager. Tasks are visualized on a kanban board, where a task can move to different stages (for example todo, in process and done). At Company D, the Tasks are composed by the team itself. There are no templates to describe a task and task are not shared by default between different projects. To align the development work in Company D, tasks are sometimes discussed during disciplinary meetings.

Tasks can also be linked to code changes, which is the case at Company D. When developers implement a feature at Company D, a task is related to the commit of this change. A tester can then check if the created software meets the requirements, described by the user story. If not, a new user story can be written to continue the development process.

5.2.2. Governance

The governance of User Stories is divided into activities related to User Stories, strategic decisions related to User Stories and key performance indicators and other measures related to User Stories.

5.2.2.1. Activities related to User Stories

There are different activities related to User Stories in a practical context. The first stage, writing the User Stories, can be done in different ways, for example at company D there are a lot of complicated projects where the customer has a dedicated requirements engineering team in place which defines the User Stories to ensure that the project requirements met the expectations. At the other companies, the product manager is responsible for writing User Stories. The User Stories are closely related to features at Company A, in other companies, like Company B and D, these linkages not clearly defined in the process of composing User Stories. At Company C the number of User Stories for some projects is very large, making it impossible for one person to oversee, so there is a strong dependence on tooling like Team Foundation Server (TFS).

After the stories for a project are written, Company E, Company D, and Company C bundle the stories into Epics, which are discussed with developers to design modules, after which the User Stories are imported in a supporting system, like Jira. Since the stories from Company C are already written in TFS, the stories bundled by product managers, which results in Epics Which are also used as input to define modules.

After the Stories are bundled in Epics from which modules are defined, the stories are distributed amongst the scrum teams, where tasks are defined and estimates are made.

Estimating User Stories is done by making use of planning poker, at Company A. With planning poker, tasks are given a value which represents the number of mediate development hours it will take to solve this task. Every team has also a number of expected development hours available for a sprint, so tasks can be divided amongst the teams, with respect to the expected workload of a task and available development hours of a team.

After the tasks are done, the built software is tested and made ready for release. At Company C there is a continuous development cycle where there is a release every night. Because of the high release frequency and high standards of delivering software, Company C also has a test automation team with is responsible for testing. Next to this testing team, there is also a team to ensure the quality of the code and decide if the code will be released in case of failing tests. At other companies, there is less pressure for a high release frequency, so a quality assurance team is often not necessary.

5.2.2.2. Strategy for managing User Stories

Most companies differ in managing User Stories, at Company A every scrum team is responsible for their own stories and the Jira tool keeps track of every story. Although there is not a specific strategy in place for managing User Stories, Company A tends to distribute the responsibilities over the teams and tools

related to User Stories. Other companies, like Company B, ensures that there are more people involved in the managing process, a product manager governs the product portfolio from multiple teams. There are others stakeholders than the product manager involved in the process of writing User Stories, for example, a test team checks for holes or inconsistencies in the stories and tests the described concepts. Company C also has a strategy team in place to oversee all Epics, User Stories are written by a product manager but customers can review User Stories and provide feedback.

Some companies also organize meetings to discuss User Stories when a project is being implemented. At Company D, for example, there is a monthly meeting between product owners and developers to discuss the work that is still open for a project and how these tasks and stories are scheduled in the sprint planning, scrum poker is often used during these meetings.

Implemented functionalities are often discussed during demo meetings, at Company D, these meetings take place every month.
Overall there is not much attention for discussing functionalities across projects, at company D, however, there are meetings for personnel of the same disciplines where the squad framework is being used. With the squad framework, specific fields of interests are divided into chapter sessions which can be helpful for internal knowledge distribution.

At the larger companies, like company A, B, and C, there is documentation available about the strategic decisions related to User Stories. At smaller companies, like Company E, documentation is not available since every team member closely works together and is familiar with the process. Company D did take the effort of writing documentation to ensure the quality of the work.

5.2.2.3. Measurements related to User Stories

There are multiple points of measure in a project, at the start, there is often an estimation of the project size, time and cost of a project. Since the complexity of User Stories takes a prominent role in estimating the costs of a project, most of the estimation work is done by developers. Story points are used to make estimations in a structured manner at company A. In principle, story points are meaningless, but could be seen as mediate development hours, where experienced developers can realize more than one story point per hour, whereas inexperienced developers realize less the one point per hour. At company A, story points do not serve as a KPI but can provide signals when something does not go to plan, within the Scrum team, for example when a team does not deliver their story points at the end of the sprint.

Company C also does not have specific KPIs in place for User Stories, but features are used to indicate if the stories are well estimated and implemented. The project's complexity is not extracted from User Stories at company C, and the estimates are also made in a later stadium of the process when more information is available. Time estimations are added just before the start of the project, to have an up-to-date status of related factors like shifts in developers. At a team level, company C also uses story points to keep track of the scrum planning.

Company D takes a more structured approach, where User Stories are measured in development hours. A senior developer is expected to resolve 125% of the issues of a mediate developer and a junior developer is expected to resolve at least 80% of a mediate developer. After a project is done, there is a feedback moment to evaluate how well the project's targets were met. When a project exceeds its budget, the customer is contacted to find a suitable solution. Often management failures are paid by company D and additional functionalities (like scope creeps) are mostly paid by the customer.

5.2.3. Roles and Responsibilities

At most of the interviewed companies, the product manager plays a central role when it comes to responsibility related to requirement management. At some companies, like company D, the User Stories are already written by the customer, so the role of a product manager shifts from writing the stories into checking and integrating the stories. At company C, the product owner is responsible for writing the and managing User Stories.

At other companies, like company B, the distinction of responsibilities is not so clear since it resides at different people throughout the process. Product managers manage a portfolio of products and are responsible for a line of products, product managers keep a close look at the industry and receive input from business analyst for future adjustments. Product owners are responsible for a specific product within a scrum team and are concerned with a more detailed level of that specific product. An UX-designers designs the product and then the testers and developers are responsible for the implementation of new features.

It also possible that the responsibilities are fixed within a project, but can shift between projects. For example in company D, there is no overall responsibility for user stories, since every project is different. In principle, every team is responsible for all tasks and User Stories related to their projects, but within a team, these responsibilities can change for every project.

5.2.4. Instruments

In most cases, the instruments related to the requirement engineering process consist of the standard toolset of the supporting system that is being used, in most cases Jira, TFS or Gitlab (in case of company E). Company A and C make use of a kanban board to visualize tasks and to communicate with developers and scrum masters regarding the print planning. Company D also uses planning poker to estimate User Stories.

At Company A there is also room for different scenarios, for example, if the story states that a user can log in, then the different scenarios could be “log in by email and password” or “log in by SMS code” or “login via another app”.

In addition to Jira, Confluence is also often used to document additional information regarding the story. Also from Atlassian Software is Hipchat, a tool to support ongoing discussions about a feature, which is often combined with bitbucket and Jira since cade references can be made.

Company B also uses Kibana to automatically create reports in analyzing log files. When software runs, a lot of signals are automatically logged. These log files contain a lot of useful information to improve services or to optimize the software in terms of efficiency or usability.

Company B also uses the Agile Inception Deck, were a couple of activities can help teams in visioning and planning process with predefined templates.

Excel or substitutes like sheets and numbers are also used in a more informal way, company D for example, used Excel to collect User Stories for a project before themes and Epics are composed.

5.2.5. Projects

Next communication during a project, User Stories can also serve as a way of measurement, or example when testers want to evaluate the implemented features. Company D also involves User Stories during the revision of a project. Sometimes, however, User Stories might change during the development phase of a project, for example when a customer requests a change in a feature. Tracing changes of User Stories might be very useful for the revision-phase of a project but is not supported by any of the tools used at company D. Moreover, a project manager has all the know-how about the scope of changes during a project, but after the project is finished these changes in functionalities and the argumentation regarding those changes is often not well documented.

At company C a quality engineering team often receives feedback from customers and other users, which is documented. The main points of improvement consist of gaps between time estimates and output. In more extreme cases, projects are not deployed despite finishing with the activities as described by User Story, this could be the case when a story is not validated and the functionality is for example out of date at the time of release.

In other companies, like company A, reviewing the project is part of the demo- and refinement sessions. In the specific case of company A, there is no evaluation regarding the User Stories, specifically. Sometimes the Story Point can be of use if a task takes much longer than expected.

5.3. User Story Summary from the Interviews

5.3.1. Definition

Since the definition of User Stories can differ regarding the context of usage, tooling must be in place to support the making use of stories from multiple roles and in multiple contexts. For example, there could be a default screen to compose a User Stories which can be called from different parts of the application (so people working on documentation can also have a button to create User Stories). Furthermore, the user interface must be easy to use for people who do not have a lot of experience in writing User Stories or are not familiar with the concept at all (for example a question mark can be displayed so a user can get information about how to write User Stories).

ID	Description
UD-1	As a product manager, I want to write a project theme so that the direction of a project is clear from the start.
UD-2	As a project manager, I want to write a projects vision so that the direction of the project is clear from the start.
UD-3	As a product manager, I want to write epic stories so that User Stories can be grouped.
UD-4	As a product owner, I want to write feature so that a User Stories can be split up into workable pieces.
UD-5	As a product owner, I want to write backlog items so that a User Stories can be split up into workable pieces.
UD-6	As a team member, I want to add User Stories from different contexts so that I can add User Stories from different roles.
UD-7	As a team member, I want to get help in writing User Stories so that I can get familiar with writing User Stories.

***Table 5.2A**: Requirements related to User Story definition.*

5.3.2. Governance

ID	Description
UG-1	As a product owner, I want to link User Stories to features so that developers can get specific information.
UG-2	As a product manager, I want to filter User Stories so that I can group User Stories into Epics.
UG-3	As a product manager, I want to automatically cluster User Stories so that I can group User Stories into Epics.
UG-4	As a product manager, I want to link Epics to modules so that a project is divided into specific parts.
UG-5	As a scrum master, I want to estimate development tasks so that a scrum planning can be made.
UG-6	As a product owner, I want to write tasks from user stories so that a User Stories is divided into specific parts.
UG-7	As a product manager, I want to link User Stories to test plans so that the testing process is structured.

UG-8	As a product manager, I want to transfer managing information related to User Stories so that I can align with other product managers.
UG-9	As a project manager, I want to communicate recommendations in writing User Stories so that product managers can work according to our strategy.
UG-10	As a product manager, I want to make estimations with story points so that project estimations can be done in a standardised way.
UG-11	As a product manager, I want to add time estimates of a project so that other stakeholders have insights on the costs.
UG-12	As a product manager, I want to adjust User Stories during the project so that scope changes can be picked up.

***Table 5.2B**: Requirements related to User Story governance.*

5.3.3. Roles and responsibilities

ID	Description
UO-1	As a product manager, I want to include User Stories written by customers so that a customer can share their needs.
UO-2	As a product manager, I want to check a list of User Stories so that I can ensure our quality standards.
UO-3	As a product owner, I want to write User Stories so that additional requirements requisites are included in the project.
UO-4	As a product manager, I want to manage a portfolio of products so that I can take responsibility for a line of products.
UO-5	As a UX designer, I want to designs a product so that value can be created.
UO-6	As a tester, I want to test a product so that quality can be ensured.
UO-7	As developer, I want to implement a product so that value can be created.
UO-8	As a project manager, I want to change responsibilities of team members so that roles can change between projects.

***Table 5.2C**: Requirements related to roles and responsibilities.*

5.3.4. Instruments

ID	Description
UI-1	As a product manager, I want Excel integrations so that I can add simple features related to requirement engineering.
UI-2	As a developer, I want to link user stories to source code so that documentation can be searched through requirements.
UI-3	As a product owner, I want to link user stories to documentation so that documentation can be searched through requirements.
UI-4	As a product manager, I want to visualize tasks on a Kanban board so that that planning is clearly communicated to the rest of the team.
UI-5	As a product manager, I want to write different scenarios related to the product so that different cases of use can be explored.
UI-6	As a product manager, I want to make estimations using planning poker so that estimations can be made in a familiar way.

***Table 5.2D**: Requirements related to instruments.*

5.3.5. Projects

ID	Description
US-1	As a product manager, I want to keep track of changes in User Stories so that a project can be reviewed more extensively.
US-2	As a product manager, I want to revise User Stories when a project is finished so that processes can be improved for future projects.
US-3	As a product manager, I want to review Story Points so that I can detect problems in an early stage of the project.

***Table 5.2E**: Requirements related to projects.*

6. Designing requirement functionality

This chapter is dedicated to the question of how the design of requirements functionality of development support systems would look like (**RQ-3**). The concluding User Stories for the previous chapters have been used as a starting point to compose a vision for software development after which the desired features are described as a first step towards realizing this vision.

6.1. Vision for Software Development

A starting point towards the vision of requirement management is the idea described by Jansen and van Rhijn (2018) where requirement engineering is a separate discipline which can be used as a starting point towards an architecture. In their paper, Jansen and van Rhijn discuss the related between Jobs and Epic from a requirement engineering perspective, which is reflected by modules and features from an architectural point of view. But for this research purpose, the provided uADL model is not enough to cover the full extent of the processes related to requirement engineering and software architecture.

User Stories and Epic stories are related composed and fine-tuned during the sprint cycles within the scrum model (Cohen, 2004) so User Stories in combination with a release can serve as input from a release plan. This release plan is then translated into different sprints where User Stories are divided amongst development teams. Within these teams, the stories are refined and estimations are provided by the developers. Tasks are also composed during the refinement sessions, which can be used as input for test plans and test scripts. In a test-driven development approach, these scripts serve the main input for writing code.

Another way cycle within the process of designing and building an application is by using Stories and Epics as the main input for designing an application, where a software architect discusses the design with developers and the product managers or product owners. During iterative meetings, the design takes shape, where functional and technical viewpoints of the desired functionalities are created. Both the technical and functional point of view serves as the main input for the development of the application.

A domain ontology can serve to link all these different artifacts and processes, so a centralized knowledge base can be created. The vision, as extracted from previous chapters is summarized in *figure 6.1*, where the uADL model is used in relation to other artifacts from the process of architecture and requirement engineering.

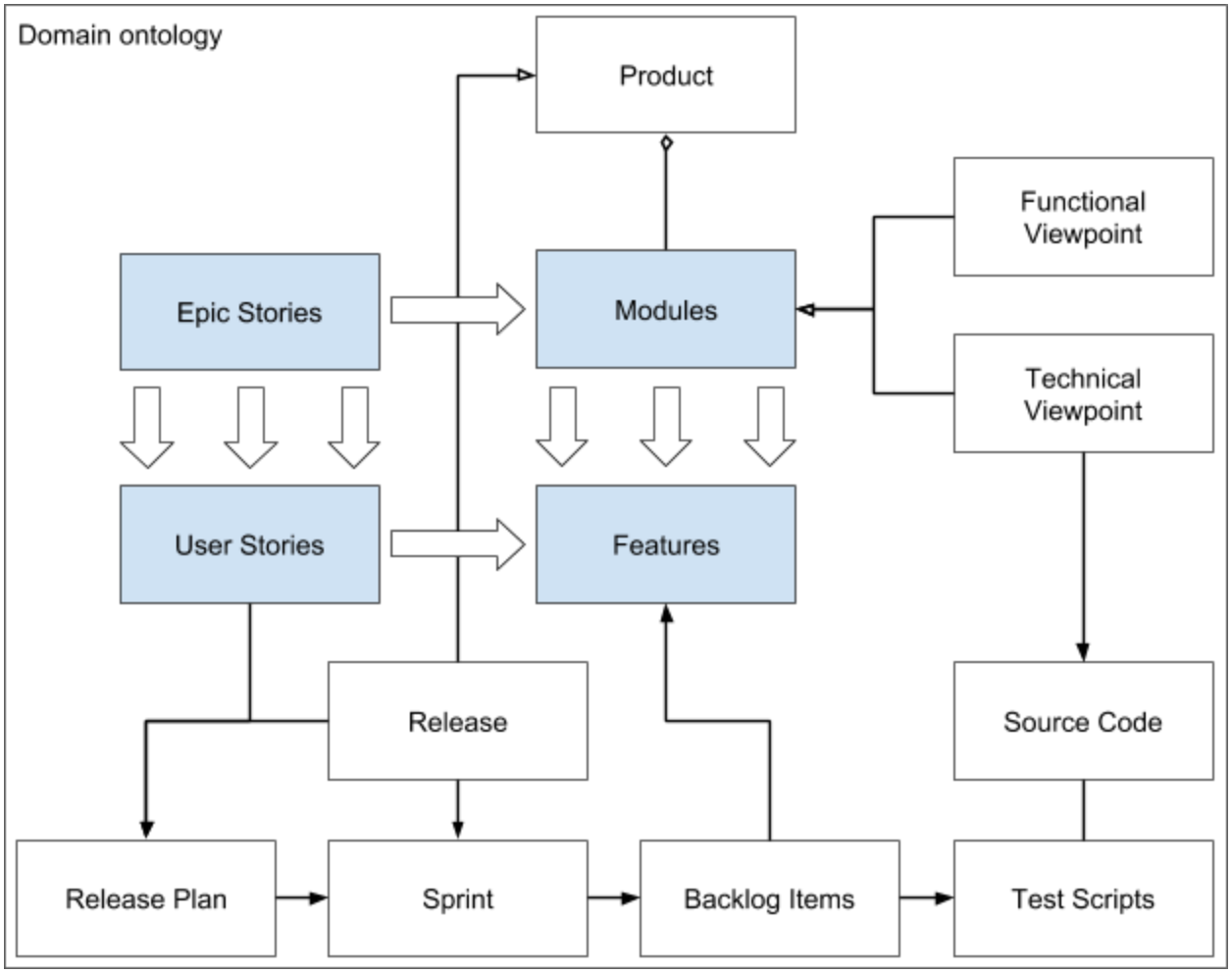

***Figure 6.1**: Vision of the role of User Stories within the development process, the uADL vision as described by Jansen and van Rhijn is displayed in blue.*

Visualizing the linguistic structure of User Stories could be of value in the process of extracting input for designing new features. From the requirement engineering perspective, User Stories can serve a central communication instrument to discuss new features, if User Stories can be used to visualize concepts and relations of requirement. By using linguistic analysis on User Stories, patterns can be shown and requirements become searchable by applying filters and adding color schemes (Slob et al., 2018).

Visualizing the linguistic concepts of User Stories is also relatively new in the industry, the initial interviews from the previous chapter showed that existing tooling does not have features to support the visualization of requirements. The concept of automatically linking different artifacts is also relatively new, some interviewees stated that tool like Jira does support the linkage of source code and User Stories, but this should be done by hand.

Integrating visualization an automated suggestions towards an architecture of an application might be useful in practice. The selected Gitlab environment is best suited to implements the features related to this vision, since additional tooling could be built easier than comparable development environments. Gitlab

has a strongly supported community and all code is open source. Because of the widespread use of Gitlab amongst companies around the world, testing these functionalities could resolve in a diverse and well-founded basis for future research about this topic.

There are also tools already in existence from previous studies, related to visualizing User Stories and extracting information from User Stories. Three existing tools are selected to be reused in the context of creating a new application to serve the features as depicted by the previous chapters:

- **AQUSA** (Automated Quality User Story Artisan), developed by Lucassen et al., (2015) provides some guidance in writing User Stories by applying quality checks on User Stories.
- **Visual Narrator**, designed and developed by Robeer et al. (2016), for an automated extraction of conceptual models from User Stories via NLP, see also Lucassen et al., (2017) for the underlying algorithm.
- **Interactive Narrator**, a tool for effective requirements exploration and discussion through visualization (Slob et al., 2018).

6.2. Desired Features

The desired features are summed at the end of each chapter as user stories. Most of the stories contain useful information, but there is some cluttering as expected with a real-life project. Some stories, for example, are duplicates, other stories contain unrealistic requirements or very specific requirements, and are therefore not very useful. In order to select themes for Jobs and Epics, the interactive narrator tool is used to visualize the conceptual model of the stories. The most important concepts are then selected and placed in a role-concept-matrix where references of the stories serve as values. The role-concept-matrix also shows some clustering of the User Stories, where the most prominent roles and concepts can be determined. These clusterings were then used as themes for Jobs and Epics.

6.2.1. Output Interactive Narrator

The output from the interactive narrator is shown in *figure 6.2*. Since the User Stories presented in this view also represents aspects closely related to the entire process of requirement engineering, the visualization can also be seen as a conceptual model for User Stories in general.

Figure 6.2*: Visual narrator output with most prominent with roles (red), themes (green), topics (blue) and relations (black) added manually.*

The most prominent concepts and roles are highlighted, where a red circle denotes a role. A green circle indicates a concept that depicts a general theme from the literature review, and a blue circle marks central themes within the network. An additional close-up is made to provide a better view of the largest cluster (see *figure 6.3*).

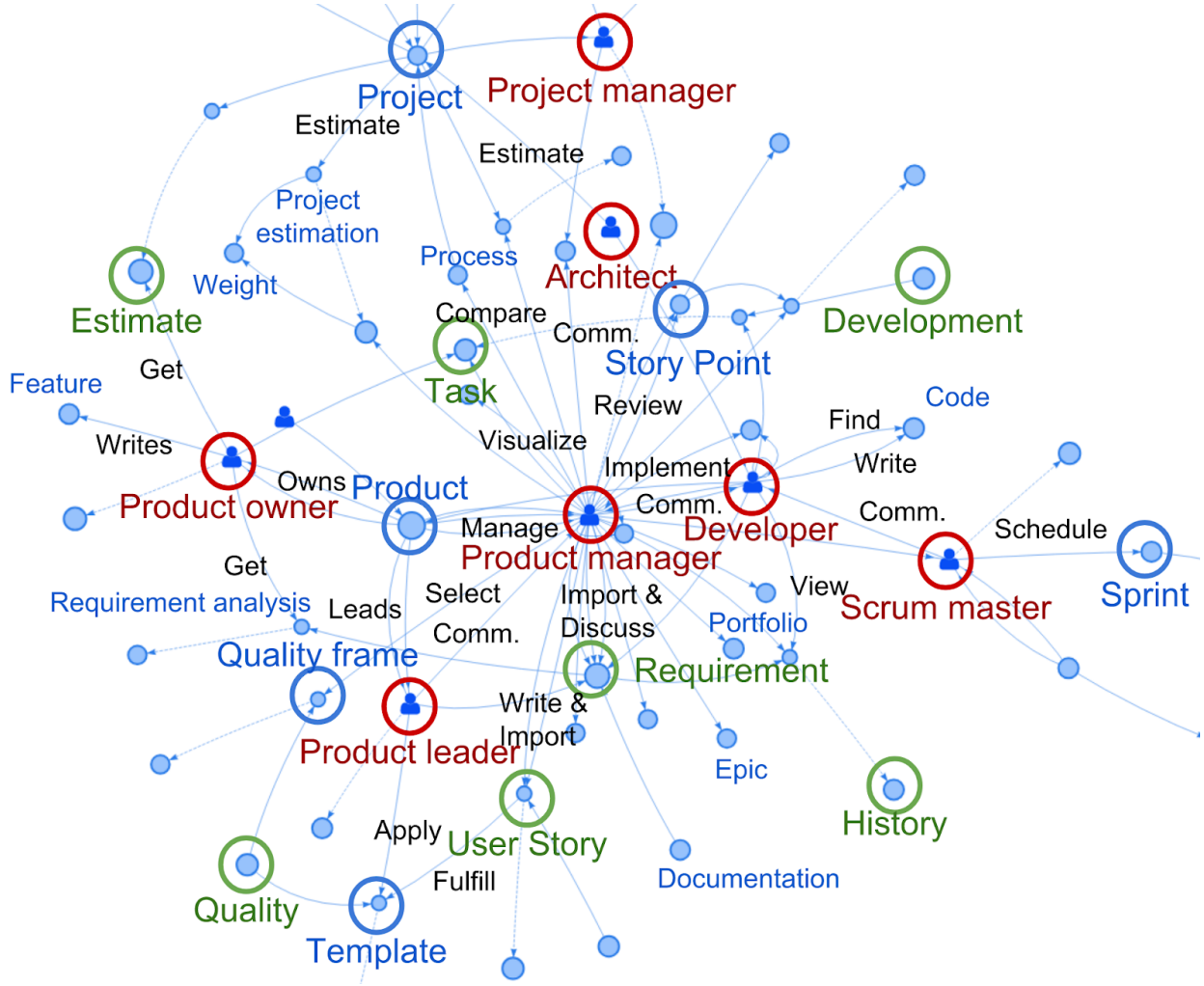

***Figure 6.3**: Close-up with product manager centered also with roles (red), themes (green), topics (blue) and relations (black).*

6.2.2. Role-theme matching matrix

All concepts that have been market green are displayed in the Role-theme matching matrix, where all roles are also included. The story references are presented as values of this matrix and can also serve as data points to indicate how similar different roles of concepts are. For example when the exact same stories are associated with two concepts, then these concepts can be joined.

Theme / Role	Product manager	Product leader	Product owner	Expert	Developer	Scrum master	Software architect
Quality	UQ-1, UQ-2, UD-3, UO-2	UT-1	UD-4				
History	UV-2				UV-1		
Development	UC-6, UP-5		UG-1, UG-6		UA-1, UA-2	UC-5, UG-5	UC-10
Requirement	UP-2, UP-3, UP-4, UI-1	UP-6	UE-5, UI-3, UO-3	UF-5	UI-2		
Estimate	UF-6, UG-10, UG-11		UE-1, UE-2, UE-3, UE-5, UD-5				

***Table 6.1**: Clustering of User Stories by roles (horizontal) and themes (vertical).*

Some manual adjustments are also made to *table 6.1* since themes with a lot of resemblance in terms of User Stories and roles can be easily identified. The choice is made to merge the concept “task” and “development” into “development” since the roles and related User Stories were almost equal. In the same way, “User Story” and “Quality” were also merged into “Quality”, which is more expressive the “User Stories”.

6.2.3. Composing Jobs and Epic stories

Now that the User stories are clustered by the interactive narrator and manually checked to select returning themes from the literature, Jobs and Epics are extracted. Some additional steps were undertaken in order to compose Jobs and Epics from the clustered User Stories:

- User stories from the Role-theme matching matrix were manually checked to remove stories with duplicate meaning.
- Conflicting stories were also manually detected and resolved. When conflicts between the literature and interviews were detected, the stories from the interviews were overruled.
- The resulting User Stories were also fed back to the concepts and roles of the Role-theme matching matrix in order to check if the stories actual depicts the proposed concepts and role.
- The resulting themes serve as input for the Jobs.

- User Stories related to the resulting themes are sorted by hand, related to the content of the User Stories and subclusters from the Interactive Narrator, to serve as input for Epic stories.

Although the set of requirements does provide a good impression of the requirements needed for this project, it is not complete because additional information from the literature or interviews was very specific, not useful or not feasible (for example during one the interviews, the interviewee stated that it would be a great addition to build an oculus rift plugin to literally walk through the visualization). The aim of this study is to provide an overall picture of the subjects related to user stories and requirement management in a practical context. Future research can focus on User Stories in relation to project estimation, requirement change management etc.

Another point of interest is that there is already a lot of functionality available, which does not have to be implemented. However, the linkage between User Stories and these features can still add values to the Gitlab tooling arsenal. For example, there are possibilities to specify User Stories and tasks but there is no template to help a user write high-quality User Stories.

Finally, the life-cycle of User Stories within a project can be of great importance in designing tooling to support User Stories. The different stages, as extracted from the literature review, are displayed in *figure 6.4*. The first step is to write the User Stories, so a set of stories is eventually composed to serve as input for a project. During a management phase, the stories can be reviewed where a story can be split into two or more smaller, more specific stores, updated or merged with another story. After the stories have been reviewed in the manage phase, the development phase starts where the stories serve as input for a design, implementation, and validation. After implementation, the stories can be stored for future changes in the project. During the two main phases of the life cycle, a story can also be signed-off, for example when the decision is made to exclude a story from the project. It is also possible that a story should be reviewed once it is in the development stage, for example when an architect or developer encounters a problem while implementing or discussing the story.

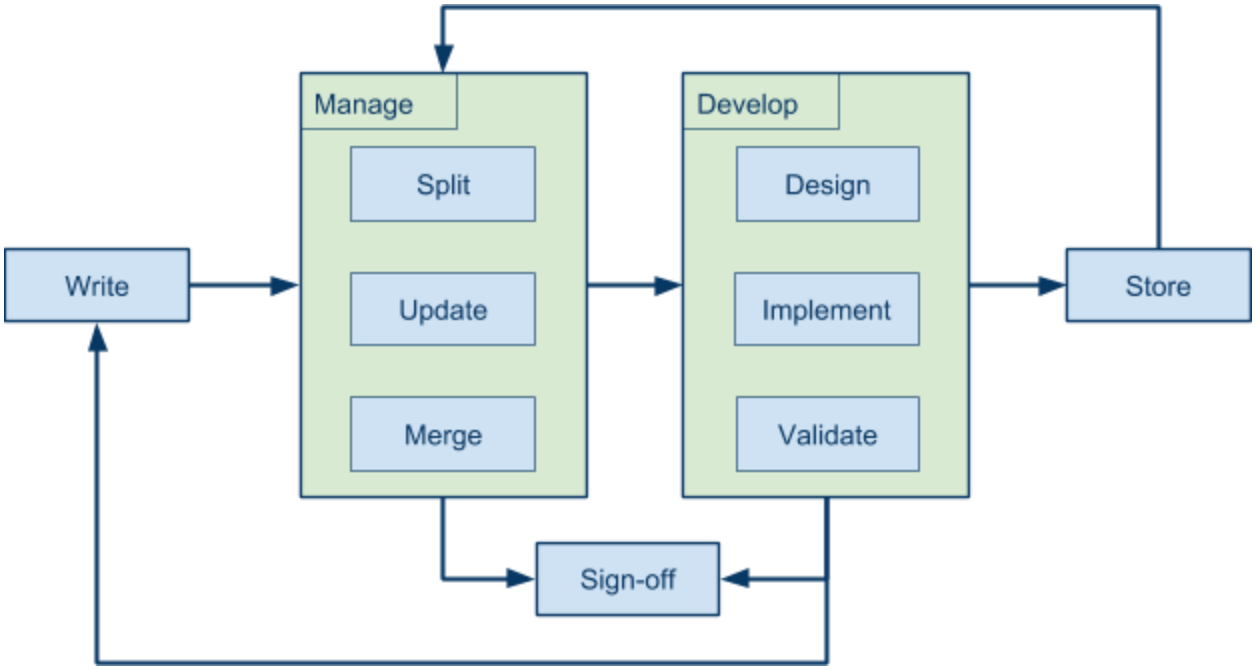

***Figure 6.4**: the life cycle of User Stories.*

The analysis from the previous section resulted in a very good overview of the requirements related to the vision as described earlier in this chapter, but the implementation of all requirements would be out of scope, given the limited time for this project. So in order to complete the process of designing, building and evaluating a tool, a selection is made from the concepts and User Stories.

Since there is already some tooling in existence to analyze User Stories, it is easy to reuse their functionality to implement a subset of features for the purpose of this study. The tools selected for this purpose were already discussed in the previous section, AQUSA, Visual Narrator and the Interactive Narrator. So in order to complete the cycle of this study, the Jobs and Epics themes related to the existing tool were used to compose the following set of Jobs:

ID	Job Story
JS-1	Help me manage requirements.
JS-2	Help me manage a project.
JS-3	Help me support development processes.

***Table 6.2**: Jobs stories for implementing the vision of User Stories in practice.*

The same selection criteria were used to compose the Epic Stories for this project, as seen in *table 6.3*.

Job ID	Epic ID	Epic Story
JS-1	ES-1	When a project is initialized, I want to gather requirements so that projects expectations can be met.
	ES-2	When a User Story is written, I want to check the quality so that software development and sprint execution go smooth.
	ES-3	When a project starts, I want to obtain a global view of all requirements so that development work can be optimized.
	ES-4	When a requirement changes, I want to keep track of the progress so adjustments can be made.
JS-2	ES-5	When a project is initialized, I want to make estimates so corporate assets can be allocated.
	ES-6	When requirements change during project execution, I want to track requirement changes so that decisions are clarified.
JS-3	ES-7	When user stories are distributed amongst development teams, I want to retrieve information from tasks so project estimates can improve.
	ES-8	When tasks are written, I want to govern the development process so that high-quality code is guaranteed.

***Table 6.2**: Epic stories for implementing the vision of User Stories in practice.*

The templates used to compose the Job and Epic Stories is extracted from the conceptual model as described by Lucassen et al., (2016). The related User Stories are included in table 6.3, where the Job en Epic Story ID's are linked to the finalized User Stories. The User Stories that have been updated in this final step have been given a different id, starting with UU (User Story Updated), the original ID is used when the User Story is not updated.

Job ID	Epic ID	Story ID	User Story Story
JS-1	ES-1	UP-2	As a product manager, I want to import requirements in Excel, Word, Visio, ScrumWorks, and Testlink so that I can work with familiar tools.
		UP-4	As a product manager, I want to discuss existing requirement with customers so I can fulfill the customers' expectations.
	ES-2	UU-1	As a product leader, I want to apply different User-Story quality templates so that existing development methods are supported.
		UU-2	As a product manager, I want to select a User-Story quality frame so User-Stories fulfill the preset quality templates.

	ES-3	UU-3	As a product owner, I want to get requirement analysis so estimates are more precise.
		UP-3	As a product manager, I want to compare of process flows so I can distinguish duplicate requirements.
	ES-4	UU-4	As a product leader, I want to ad-hoc walkthrough requirements so I can inspect the requirements.
		UF-5	As an expert, I want to make a judgment on requirements so that I can make adjustments.
		UP-5	As a product manager, I want to track changes in requirements so backtracking from future adjustments is possible.
JS-2	ES-5	UU-5	As a product owner, I want to get time estimates so estimates are more precise.
		UU-6	As a product owner, I want to get project size estimates so estimates are more precise.
		UU-7	As a product owner, I want to get effort estimates so manual estimates are more precise.
		UU-8	As a product manager, I want to assign story points so project estimates are concise.
	ES-6	UU-9	As a product manager, I want to view requirements history so I can see why certain decisions are made in relation to system updates.
		UU-10	As a developer, I want to view requirements history so I can understand the the reasons about previous implementations.
JS-3	ES-7	UU-11	As a scrum master, I want to schedule sprints so the development tasks are implemented according to the project estimates.
		UU-12	As a product manager, I want to communicate with the scrum master so development tasks can be prioritized.
	ES-8	UU-13	As a developer, I want to implement a development task so that a project can be finished.
		UU-14	As a developer, I want to write code so that development tasks can be implemented.
		UU-15	As an architect, I want to communicate with developers so I can check if the development tasks meet the technical design.

***Table 6.3**: User Stories related to the Job and Epic Stories from table 6.1 and 6.2.*

6.3. Design Decisions

The first step of starting a project is by devising a name for the tool, this way it becomes more easy to refer to the tool and it also makes the project come more to life. Since the key concept of this project is scrutinizing the lexical structure and architectural relations of User Stories, the tool is called the Architectural Lexical Arrangement Scrutinizer (ALAS), which is also a reference to a story written by Grimm brothers (see also *Appendix D*).

6.3.1. Stories to Features

The stories composed in the previous section are used to define the different windows and features of ALAS. Every Job Story, with underlying Epics, is discussed in a separated subsection.

6.3.1.1. Manage Requirements

For epics story ES-1 and ES-2, on gathering requirements and quality checks, an editor has to be build so User Stories can be composed and the quality of these stories can be ensured for further analysis. This story editor should contain a way of importing user stories, where different tools like Excel, Word, Visio, ScrumWorks, and Testlink should be supported as input stream (UP-2). Since all of these tools have an export function where User Stories can be saved as a raw text file, the editor can comply with this requirement by including text import, for example by dragging and dropping. The editor should also contain an overview of all stories, for example with colored templates to make the stories more readable in order to discuss the content with other stakeholders (UP-4).

To ensure the quality of the User Stories, the Story Editor should also include feedback as provided by AQUSA (UU-1), since this feedback should only be visible when the user wants to correct a story the feedback messages itself should be hidden but every story containing an error or warning should be highlighted. So the decision is made to use the enumeration of the Story Editor to indicate if a story contains warnings or errors. Where a warning is displayed as yellow and an error as red. The user should also be able to change the templates of the story editor (UU-2), which is supported by AQUSA.

For Epic Story ES-3, on estimation, an overview of the project structure with some additional information regarding the size of the project should be included. This overview then serves a starting point to guide the discussion on estimating the size of a project (UU-3). For example, an architect can give a time estimate for every feature in every module. Included information in this hierarchical view should be the main structure of the project, for example by displaying suggestions for modules and products of the project and include User Stories and roles that are related to the features, modules, products. Als the process flows should be visible, where duplicates can also be detected (UP-3).

Epic ES-4, to keep tract of the process, can be achieved by visualizing the User Stories in a graph, just like the Interactive Narrator. This way a product leader can do an ad hoc walkthrough to inspect the requirements (UU-4), experts then also have a central view to making judgments about the requirement clusterings (UF-5). The experts should also be able to make changes in the requirements in the story

visualizer and change the level of abstraction in order to give more input for a discussion. When discussing the features of a clustering of stories, it could also be valuable to mark certain concepts to serve as input for Epic or Job Stories. This overview should also be updated when the stories change, so the product manager can see if the requirements have been updated (UP-5).

Epic ID	Window	Story ID	Feature descriptions
ES-1	Story Editor	UP-2	• Drag and drop text files.
		UP-4	• Overview of all stories. • Template highlighting.
ES-2	Story Editor	UU-1	• Warning and error indication. • Feedback by AQUSA.
		UU-2	• Change the story template.
ES-3	Story Hierarchy	UU-3	• Display the structure of the project. • Display the related User Stories and roles for every element.
		UP-3	• Display process flows.
ES-4	Story Visualizer	UU-4	• Visualization of the User Stories to expose the linguistic structure of the sentences, like Interactive Narrator.
		UF-5	• Update the requirements when changes are made. • Include markings for selecting themes for Epic and Job Stories. • Change the level of abstraction.
		UP-5	• Connect the visualization to the User Stories, so the visualization is updated when the stories change.

***Table 6.4**: summary of features per User Story for Job Story JS-1.*

6.3.1.2. Manage Project

Another story of epic on estimation is ES-5, where the estimation serves as input to allocate assets on a corporate level. In order to get a good insight into project size estimates, the story hierarchy should support interactive shifts in project structure and scope. For example when the development team is discussing a certain module and comes to the conclusion that it is better to shift one feature to another module or even change a feature to a module (UU-5). Next, to the project size estimates, the development team should also be able to estimate the time needed for a project. In order to achieve a well-structured effort estimate, the team needs insights into the User Stories related to a feature or module (UU-7). Next,

to the effort estimates, the team should also be able to make time estimates. Every product and module should be discussed in terms of related features, so when a user selects a module or product, the related features and User Stories should appear (UU-6). In some teams, assigning story points helps to estimate the project, so for this purpose some additional notes related to a product, module or feature should be included (UU-8).

Epic ES-6 focuses on tracking changes of a project, the version control of Gitlab is well suited for this purpose. For a product manager, this means that the requirements history can be visualized out-of-the-box when the stories are saved on Gitlab as text-files. When someone makes changes in the User Stories, these stories should be automatically synchronized with Gitlab so historical data can be reconstructed by Gitlab (UU-9). Developers can also use the version control from Gitlab so they can view historical data from finished projects that need to be updated (UU-10).

Epic ID	Window	Story ID	Feature descriptions
ES-5	Story Hierarchy	UU-5	• Allow changes in the project structure.
		UU-6	• Only display modules and features related to a selected product. • Display all products, modules and features when nothing is selected • Only display the related features when a module is selected.
		UU-7	• Display the related User Stories when a product or module is selected.
		UU-8	• Include an additional text field to include story points estimations.
ES-6	Gitlab	UU-9	• Automatically synchronize changes in User Stories with Gitlab.
		UU-10	• Save User Stories and other data as text files in order to use the version control of Gitlab.

***Table 6.5**: summary of features per User Story for Job Story JS-2.*

6.3.1.3. Support Development Process

The path from designing requirements to implementing them is the theme from Epic ES-7. After the design decisions have been made and the project estimates are in place, the features of the project need to be included to Gitlab (UU-11). Since the requirements and other project information is already synchronized with Gitlab, the tasks can be easily included in the scheduler of Gitlab. Saving the stories in Gitlab also permits using the stories in the rest of the project, for example when a project manager wants to discuss the stories with a scrum master (UU-12).

The final Epics of this project (ES-8) is focused on the development process, where features from Gitlab are used to support the implementation of User Stories. When developers implement tasks, related to a User Story, these tasks can be labeled Done. this labeling should be synchronized with ALAS in order to trace the development process (UU-13). In order to write code with respect to the project, the developer can use the out-of-the-box features from Gitlab to deploy and review the source code. When this code is displayed on Gitlab, ALAS should also include a feature of viewing the projects structure and source code so developers can check the code on a project level (UU-14) and architects can review if the source code meets the technical design requirements (UU-15).

Epic ID	Window	Story ID	Feature descriptions
ES-7	Gitlab	UU-11	• Save stories and descriptions in Gitlab.
		UU-12	• Interactively exchange information between Gitlab and ALAS.
ES-8	Gitlab	UU-13	• Trac the statuses of tasks.
		UU-14	• Include a code editor to review source code on a project level.
		UU-15	• Visualize the project structure as saved on Gitlab.

***Table 6.6**: summary of features per User Story for Job Story JS-3.*

6.3.2. Moving Concepts in Hierarchy View

This section is an addition to Epic ES-4 and ES-5, on labeling concepts extracted from User Stories and interacting with the products, modules, and features of a project. The features described the previous section about upgrading or downgrading products and modules, are more elaborated in this extra section.

There are multiple movements possible, for example, a user can move an Epic to the Jobs column, which means the moved Epic Story will become in a Job Story. A user can also select a concept while making the move, which results in a change of relationships between the selected and moved concept.

It is possible to select multiple concepts of different types, for example, both a Job and an Epic can be selected at the same time, but not multiple Epics. A selection has also another purpose since the lists of child concepts are filtered to only show the concepts related to the selected concept. When both a Job and an Epic are selected, all concepts in the Epics-list are related to the selected Job, the list of User Stories contain all Stories related to the selected Epic.

With respect to all possible changes, the relation-shifts between concepts are explained in *table 6.4*.

Symbol	Description	Relation Change
C	Becomes the child of the selected concept	Create relation with the selected concept (only there is no existing relation)
G	Becomes grandchild of the selected concept	Make relation with the selected concept (only there is no existing relation)
S	Becomes sibling of the selected concept	Remove relation with the selected concept (when present)
P	Pibling, becomes a parents sibling	Remove relation with the selected concept (when present)
D	Downgrade	None
U	Upgrade	None

***Table 6.4**: possible concept movement actions*

Note that a moved concept cannot become a parent of a selected concept since the moved concepts cannot be selected.

Since there are three types of concepts (Jobs, Epics and User Stories), three dimensions of every relation shift event (from concept to concept, selected concept) containing respectively three, three and six possible states, there are 54 possible changes in total, which are displayed in *table 6.5*.

Selected			To		
			J	E	S
From	None	J	x	D	D
		E	U	x	D
		S	U	U	x
	J	J	x	C	C
		E	S	x	G
		S	S	C	x
	E	J	x	S	C
		E	U	x	C
		S	P	S	x
	S	J	x	D	D
		E	U	x	D

		S	U	U	x
	J & E	**J**	x	C_J	C_E G_J
		E	S_J	x	C_E G_J
		S	S_J	S_E C_J	x
	J & E & S	**J**	x	D	D
		E	U	x	D
		S	U	U	x

Table 6.5*: changes in concept states as a result of user action.*

As you can see from *table 6.5*, the state changes are the same in the situation where there is no selection or only User Story or when a Job, Epic and User Story are selected.

The table also contains an x-symbol, which denotes that a given action does not imply a change of states, for example, the user can't move a Job to the Job-column.

Finally, when both a Job and an Epic concept are selected, a movement can result in two state changes, for example when the user moves a User Story to the Epics column. In this case, the User Story will change from child to sibling in relations to the Epic and becomes a child of the selected Job. The concept type in relation to a state-change is denoted in subscript when multiple states change in one action.

7. Implementing Requirement Functionality

The central question of this section (**RQ-4**) is how to implement the functionalities described by the user stories in the previous sections. These functionalities are strongly interconnected with the Gitlab API. To understand how the tools for managing user stories coincide with Gitlab, I will first discuss the architecture of Gitlab and then how ALAS is integrated in Gitlab.

7.1. Look and feel of ALAS

The ALAS-tool contains four static elements that remain constant throughout the navigation of the tool:

- **Header**: contains the name of the tool with a link so u user can navigate back to the home screen at any moment.
- **Breadcrumbs**: At the top of the screen is a breadcrumb path included, just like Gitlab, so a user knows at any moment where he has navigated to and can go back some steps.
- **Left sidebar**: Navigation bar which also depicts the process of requirement management (from data as User Stories to a information with a story visualizer to a project outline in the hierarchy window).
- **Footer**: Displays the general information about the tool and its purpose. There are also links included in the footer to navigate to additional information pages about the tool, it origin and how to use it.

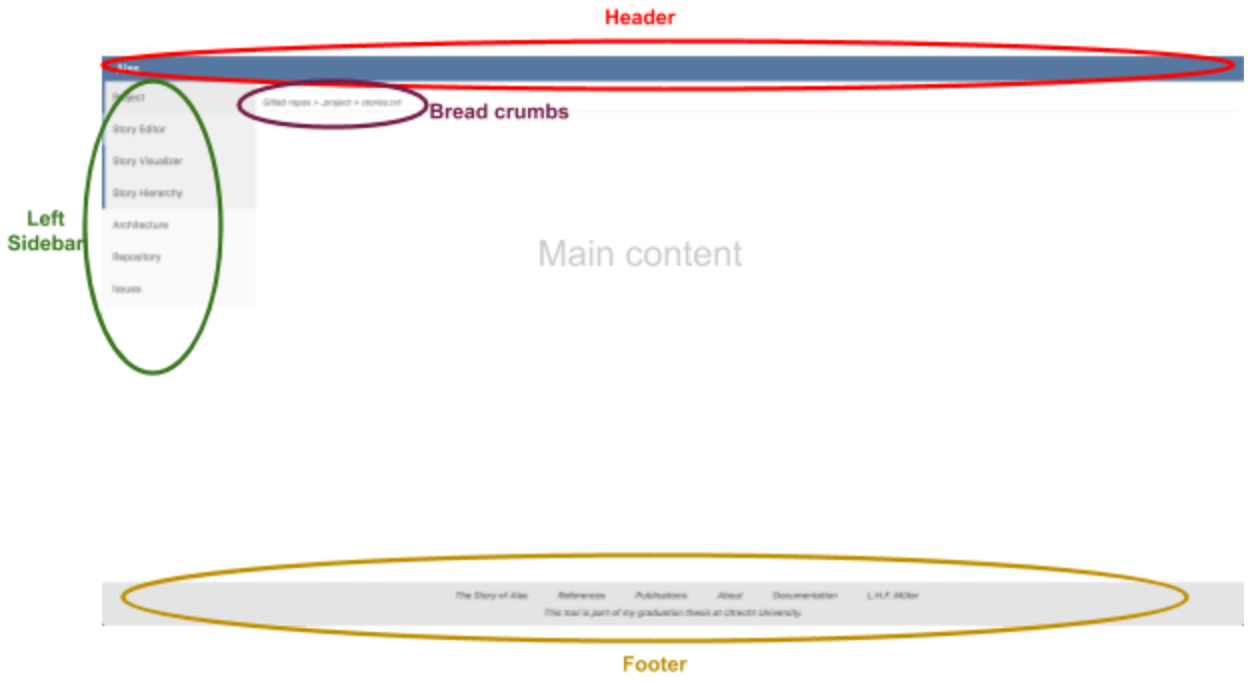

***Figure 7.1**: Static layout of ALAS.*

7.1.1. Story Editor

The first view the user will see when opening a project is the story editor. User Stories can be edited in the story editor view, which embodies an editor especially configured for supporting user stories.

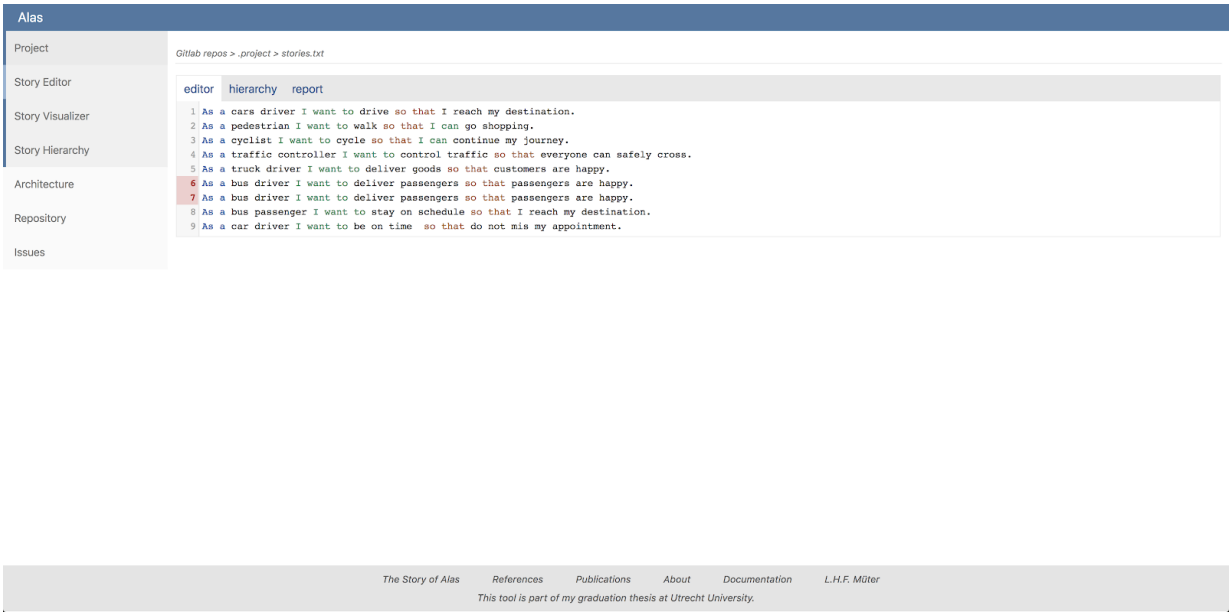

***Figure 7.2**: impression of the story editor.*

Some of the features included in the story editor:

- **Template highlighting**: when a story is pasted or written in the editor, some chunks of texts are highlighted to indicate the story template. For example when a user types “As a” or “As a” the text automatically gets a blue font-style.
- **Jump-feature**: when the TAB-key is pressed, the cursor will jump to the next section of the story template. So if a user has edited the role of a story and wants to make some changes in the benefits part of the story, he only has to press the tab key twice to jump to the benefit part (indicated by “so that”).
- **Template write**: The default user story template is created when a user presses the enter key on a new line, which can save a lot of time in combination with the jump-feature.
- **Copy-paste**: With copy paste shortcuts (by default ctrl-x, ctrl+c, and ctrl+v), text can be cut, copied and pasted to the clipboard of a user. This features can also be used via a menu which is activated by a right-click in the editor.
- **AQUSA**: The integration of AQUSA provides a real-time feedback to the user when a story does not follow the quality requirements as set by AQUSA. So when a user has written a User Story, AQUSA will automatically check this story and will give a warning or error indication in the left line-bar of the editor, red for an error and yellow for a warning. The user can click on this indication to see the error(s) or warning(s).

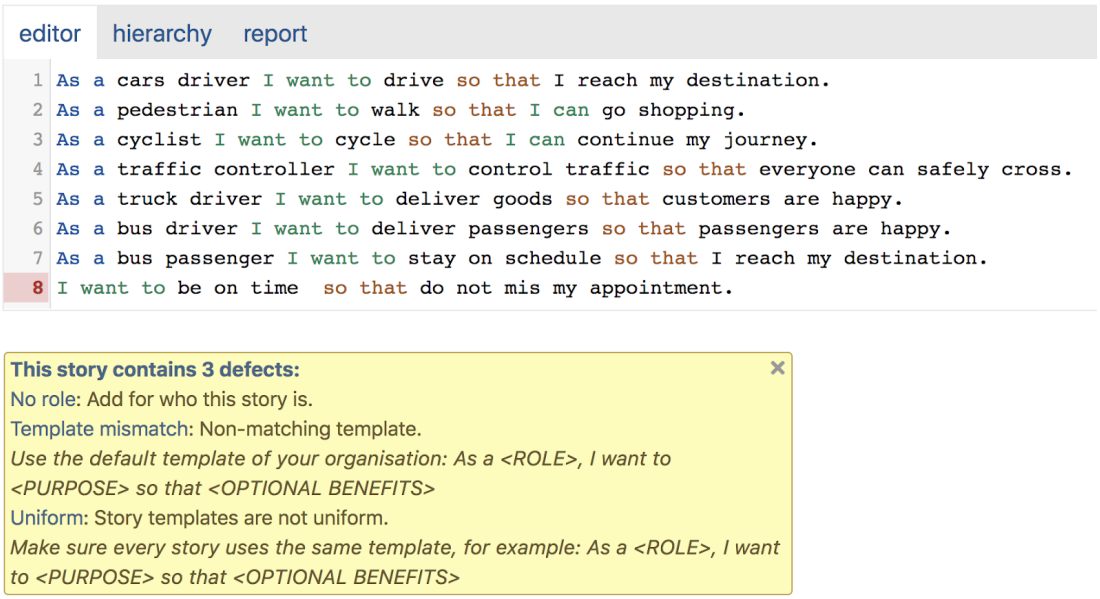

***Figure 7.3**: Story Editor feedback on a User Story.*

7.1.2. Story visualizer

In order to display the content of the User Stories, the visual narrator is used. User Stories are analyzed so concepts relations are extracted and displayed in a network graph. The ALAStool also indicates if a concept can serve as a theme for a Job or an Epic, by highlighting concepts in orange or green, respectively. The user can also remove this marker by selecting the concept and press the remove-marker-button. Themes markers can also be set by selecting a concept and pressing the mark-as-Job or mark-as-Epic button.

The user can also navigate by dragging the concept graph or by using the arrow buttons on the left side of the window.

Zooming-features are also included, so the user can zoom in or out with the plus and minus buttons right side of the window, or by using the scroll wheel of a mouse.

Finally, a user can also change the level of the graph, by shifting the change-level-slider. This feature is can be of use when the number of User Stories is very large. Only prominent concepts are displayed when the slide-bar-indicator is shifted to the right, which makes the resulting image less sensitive but more specific. On the other hand, when the slide-bar-indicator is shifted to the left, all concepts will be displayed, so the representation is more sensitive but less specific.

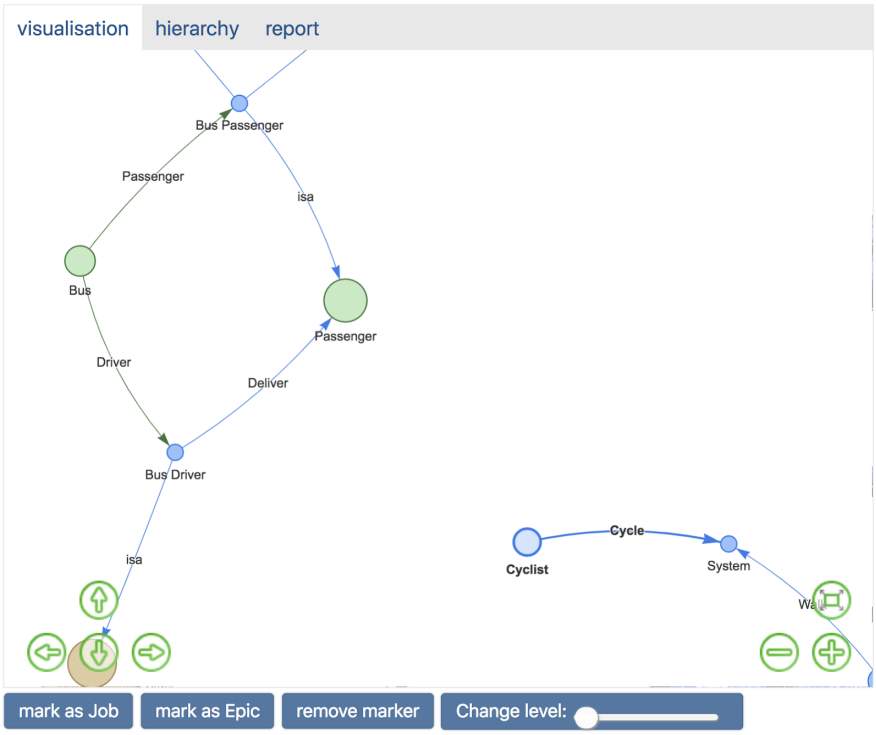

***Figure 7.4**: Impression of the Story Visualizer.*

7.1.3. Story hierarchy

Project outline where products, modules, and features extracted from the user stories come together in the Story Hierarchy view. The information about the concepts and related from the story visualizer is used as input, and additional information about the Jobs and Epic themes is added.

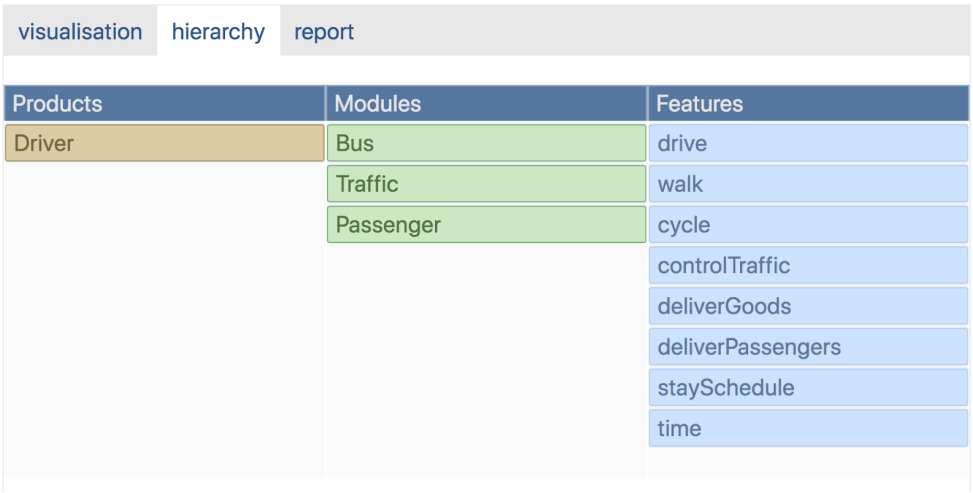

***Figure 7.5**: Impression of the Story Hierarchy.*

Details about related roles and User Stories are also shown when a user selects an item. The information related to a selected item is displayed in a pop-up window where the theme and description can also be edited, these changes are automatically synchronized with text-files hosted on Gitlab.

Since Jobs, Epics and User Story clusters can form an interconnected hierarchy, related Job, Epic themes or User Story clusters are shown when an item is selected. This matrix view also contains features to move Job themes to Epic themes or feature themes to Epic themes, this information is automatically updated in the story viewer window.

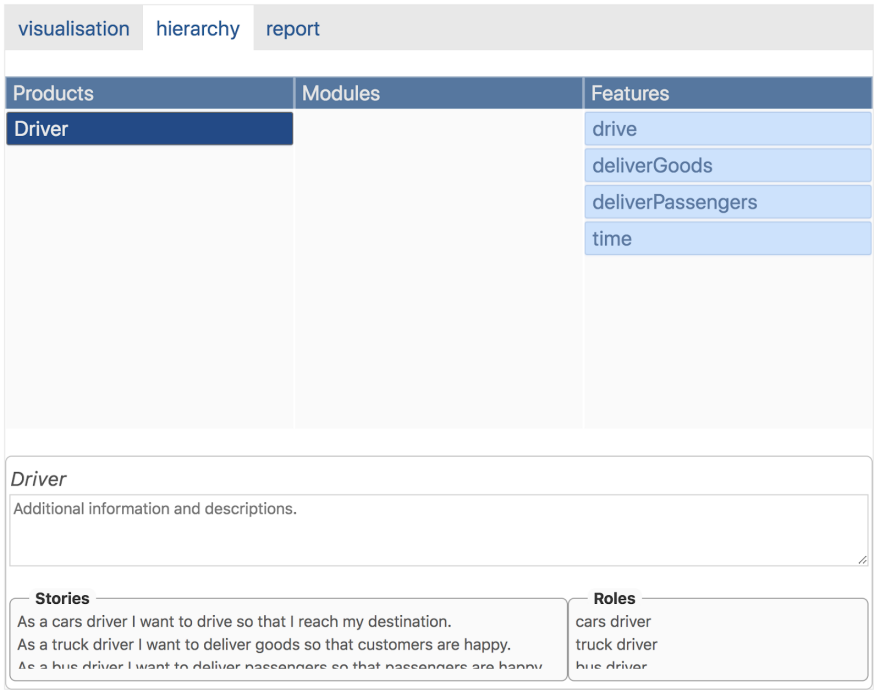

***Figure 7.6**: Filtering and details show when a hierarchy item is clicked.*

The features of moving Job themes (indicated by the "product" column) to Epic themes (indicted by "modules") or User Story clusters (indicated by "features") contain more functionality then changing the markings as Job or Epic theme. For example, when the product "Driver" is selected, then only the related features and modules are displayed. When a user wants to upgrade the feature "time" to a product of its own, he can drag the time- feature to the product column. When the time-feature becomes a product, then the relationship with the driver-product is no longer needed, since dependencies between products are not desirable. So if a user upgrades the time-feature to a product, then the relation between the driver-product is removed. Similarly, when the time would be a product, it can be downgraded to a module or feature related to another product or module.

7.2. Architecture

Since the features provided by ALAS are stored on Gitlab as text files, the interaction between the end user, ALAS and Gitlab can be visualized as in *figure 7.6.* The reason for this design decision is to easily

integrate the version control features of Gitlab and to more flexibility to ALAS. For example, an existing project could be added by simply pasting text files into a special folder (.project) in the project root. ALAS will automatically read the contents of this folder and will display its contents as described in the previous section.

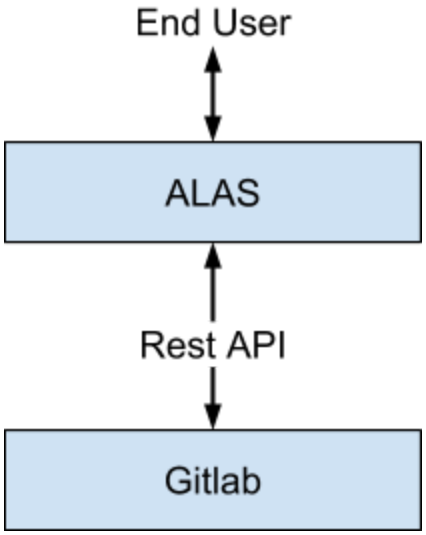

***Figure 7.7**: Interaction between the end user, ALAS and Gitlab.*

7.2.1. Architecture of GitLab

Gitlab is built in Ruby-on-rail, an easy to learn programminglanguage that is widely spread amongst web applications. The open character of Gitlab is also presented in the well-documented architecture which will be discussed next. Gitlab is also supported by a large community since most of the source code is freely available.

Gitlab contains multiple modules, each with a specific task and a set of features, see *figure 7.7*. For this project, the following modules are of interest:

- **NGNIX** (front end) is responsible for communicating with the end users, all connections are handled in this mode. The new supporting features will give the end user additional fields, so information has to be transferred through the NGNIX-module.
- **Unicorn** (security manager) handles security and permissions, so information from the database has to go through Unicorn in order to reach the customer.
- **Redis** (load balancer) can be compared with a communication board, to executes tasks in other parts of the application. The new requirement features will make use of some of the existing functionality of Gitlab, so information has to pass through Redis.
- **Gitlab pages** (documentation), provides information regarding the use of the Gitlab environment. Since the requirement features have to show information regarding the use of the new functionality (which also has to be editable for organizations) there must be a coupling to information from the Gitlab pages.

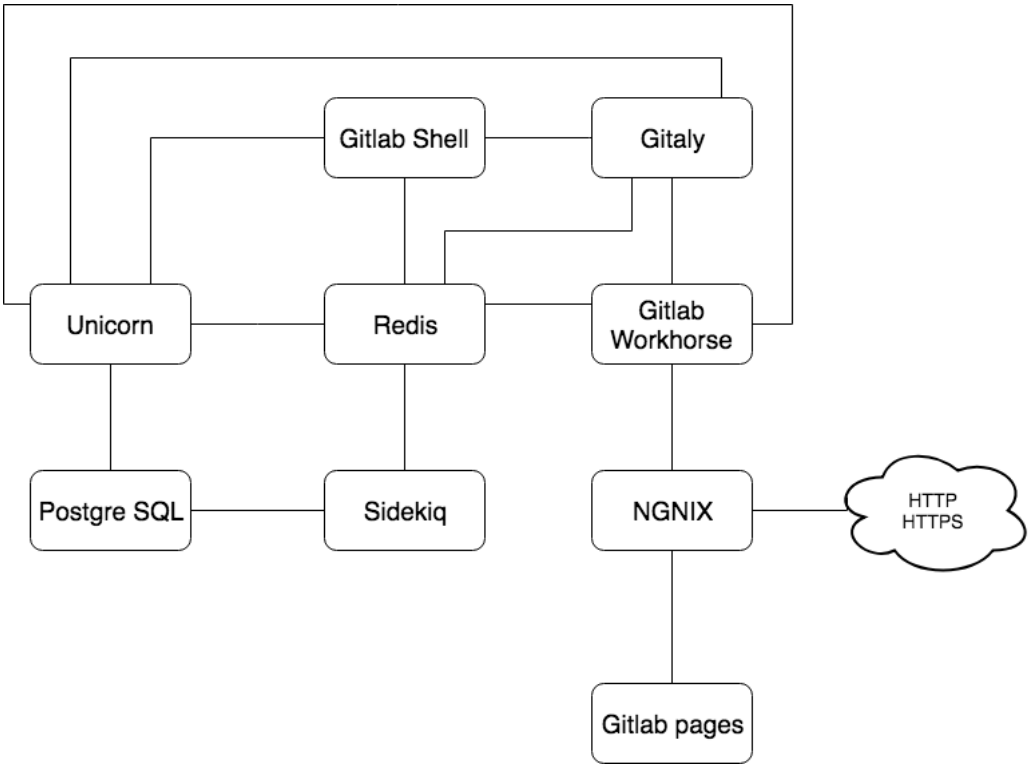

***Figure 7.8**: Gitlab Architecture.*

The functionality of every module can be reached by a REST-API, which makes it possible to write additional functionalities that can interact with Gitlab.

7.1.2. Architecture of ALAS

The global design of ALAS fits the architecture of Gitlab as shown in *figure 7.7*. The functionalities of ALAS itself can be subdivided into the following modules:

- **Story Quality** contains features to check if a User Story is written according to a preset template, a template can be added or selected by the project manager.
- **Story Input** is dedicated to creating new user stories, which can be done in two different ways (writing a single User Story via Gitlab graphical user interface and by bulk by uploading an Excel- or CSV file). This module communicates with the User Story Quality module to provide feedback on the written User Stories.
- **Story Viewer** is dedicated to viewing User Stories, comparable with the visualization of the Interactive Narrator. In this module, a user can add an overlay to this visualization to get estimates and process flows. This additional information is extracted from different modules.
- **Story Hierarchy** is a module to display an change in the structure of the Epics and Jobs related to the User Stories.

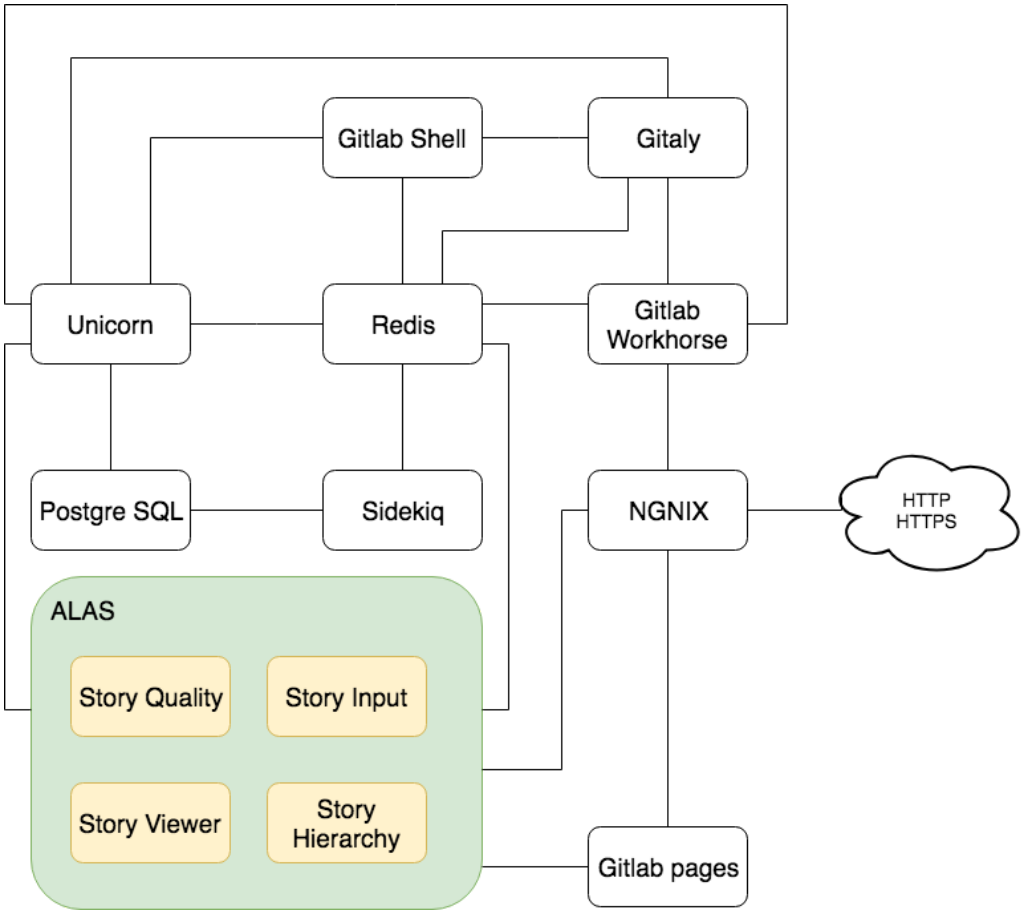

***Figure 7.9**: the architecture of Gitlab with ALAS.*

The different interactions between ALAS and Gitlab are displayed in *figure 7.8*. Unicorn makes sure the central entry point is kept secure, so in order to use the REST-API of Gitlab, a special id has to be requested from the graphical user interface. Redis is the central load balancer that has to decide how many call and data is send to ALAS, in case of large chunks of data, the Redis service can decide to close the connection. NGNIK is the front end where most of the user interaction takes place, at this moment ALAS has its own user interface but still has to interact via the NGNIX, for example when a user wants to make changes in the raw data files of ALAS. the Gitlab pages are mostly used for documentation, ALAS can also interact with this service, for example, to display information is the repository view.

7.3. ALAS modules

ALAS is built upon different modules that interact independently on each other. Because a lot of different tools and projects are integrated to obtain the functionality required for the complex story analysis. A flexible design is chosen to change or even replace functional units in the long run.

7.3.1. Story Quality

All functional units of the Story Quality module are displayed in *figure 7.10*. The central component in story quality is AQUSA. The data from the story editor is sent to AQUSA via an API-call, this way changes of AQUSA will not affect the rest of the application and thus provides additional flexibility.

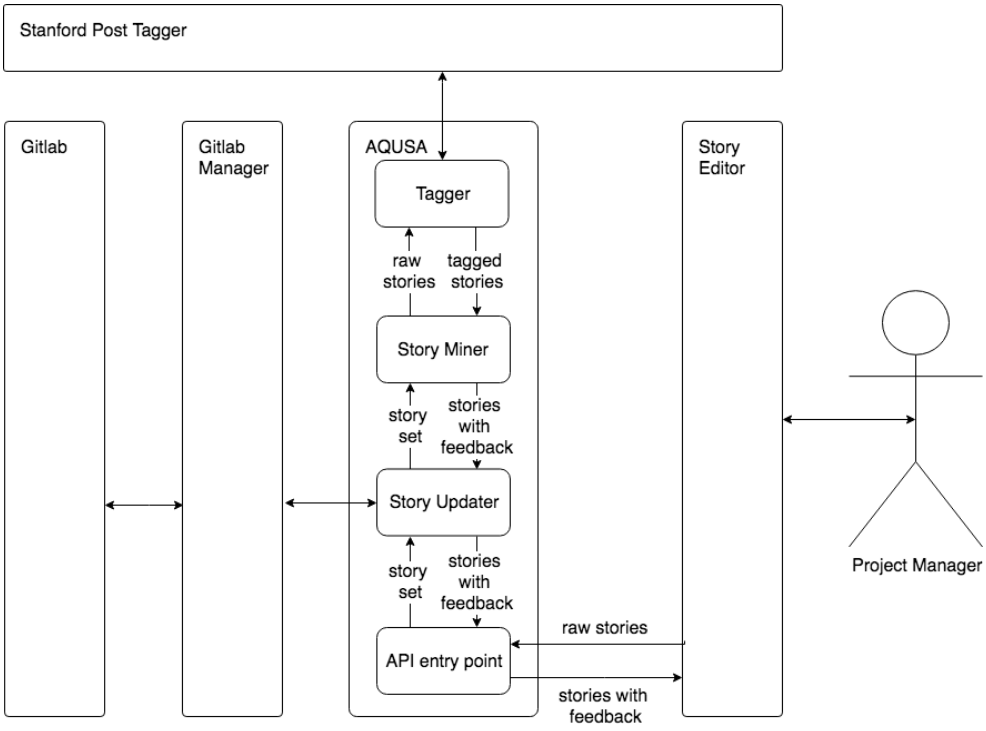

***Figure 7.10**: Story quality with AQUSA.*

Within the AQUSA-module, the story updater sends the raw story set to the story miner and receives analyzed stories with feedback. The stories are then saved on Gitlab via the Gitlab manager. The story miner contains the core functionality, the stories are tagged after which rules are applied to check if the stories meet all requirements. The miner on its turn interacts with the Stanford POST tagger via a tagger class to extract the sentence structure. This information is sent back to the story miner. There is also additional functionality in place to interact with the Stanford POST tagger, but this is included in the tagger class. The Stanford POST tagger itself is hosted as a different service so other modules or services can make use of this tool in the future.

7.3.2. Story Input

The main component of the story input module is a text editor (codeMirror) which is set-up to display user stories with a code highlighter.

The story editor contains three sub-modules which all communicate with codeMirror:

- **Event manager**: for capturing specific events from the editor, for example when a line number is clicked, or when the <tab> or <enter> key is pressed within the editor. When an event is fired, the event manager will call the corresponding functions, in the javascript classes where a different rest-call is made to the backend via the jQuery-library.
- **Story feedback manager**: for communicating with AQUSA. The raw stories from the text editor are sent to AQUAS and feedback is received, when an error or warning emerges the feedback manager creates writes the message into a hidden HTML-field, the line of the story where the error occurred is highlighted, and a link is activated so when a user clicks on an error-indicating line number, the hidden HTML-field will appear.
- **Story template manager**: where templates are stored in order to generate an empty template when the user presses <enter> on an empty line. When a user presses <tab> on a non-empty line, the template manager tries to discover a matching template and finds the next part of this template. A cursor position is returned corresponding with the next part of the template. The editor then moves the cursor into given position.

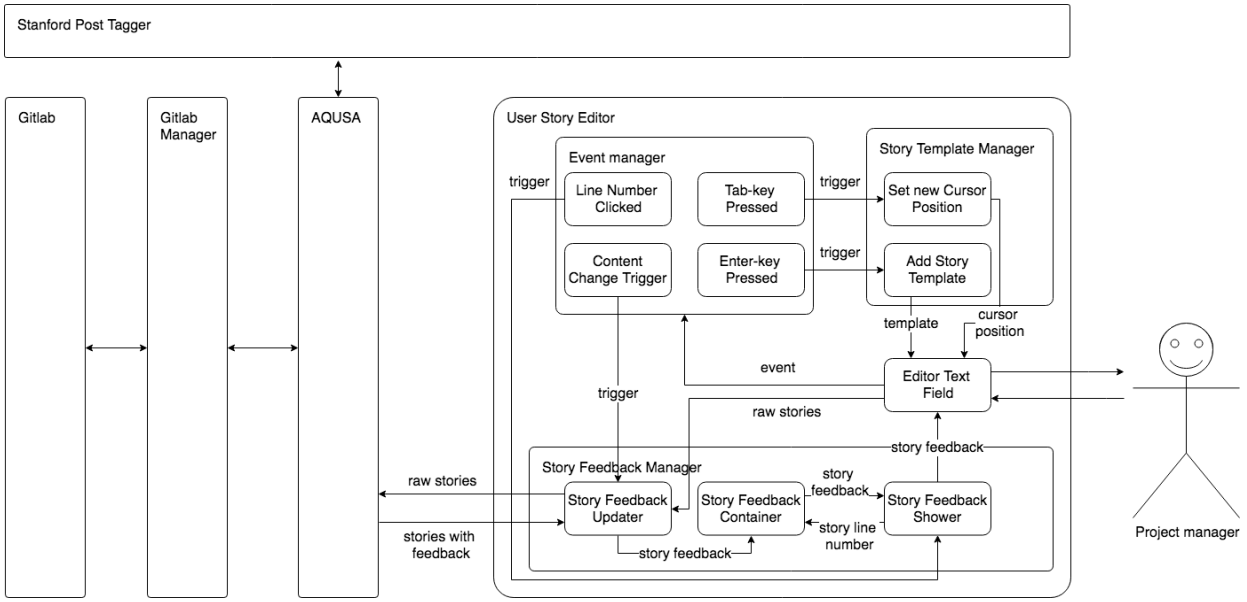

***Figure 7.11**: Story input with Codemirror text editor.*

7.3.3. Story Viewer

The story viewer module uses the visual narrator to extract the information from the stories. The visual narrator is contained in a different functional unit, to ensure flexibility. The visual narrator in its turn uses the spacy tagger as part of a python library so it does not need to be hosted differently, like the Stanford POST tagger in AQUAS.

The three main sub-modules of the story viewer are:

- **Visual narrator**: which is an existing tool for generating a report with sentence structure and statistics and for generating an ontology based on user stories. The Visual narrator interacts with the Gitlab manager to automatically synchronize the analysis with Gitlab.

- **Gitlab manager**: the main features of the Gitlab manager are creating new files if a requested file is not found, updating existing files when changes are detected or delete files. The Gitlab manager also parses the API-calls from ALAS into API-calls fit for Gitlab, and also holds some flexibility in form of parameters for example when a frontend developer wants to compare the content of a file without updating the existing file on Gitlab. Since the Gitlab manager is also a self-serving functional unit, it can be changed or even replaced in the future when Gitlab updates its API.
- **User Story Visualizer**: interaction between Visual narrator and visualizer is done by an API-calls. The visualizer contains four different modules, the *event manager* for handling user events when interacting with the visualizer, the *concept graph manager* for drawing the draws the concept graph (the actual visualization of User Stories), the *data manager* to interact with Visual Narrator via API-calls, and the *Ontology visualizer* to receive analyzed information from the Interactive Narrator and translate this information into Nodes and relation so the information can be interpreted and visualized by the concept graph.

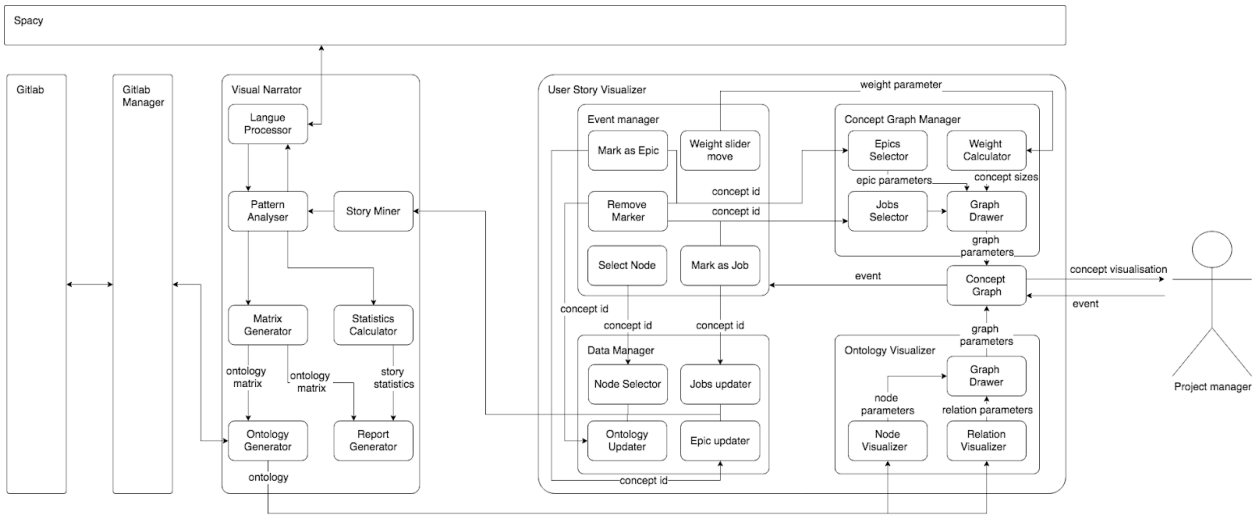

***Figure 7.12**: Story viewer and story analysis with the Visual Narrator.*

7.3.4. Story Hierarchy

The story hierarchy uses the information stored by the story visualizer, via the Gitlab manager, described in the previous section. When the output files from the story viewer do not exist, the story hierarchy can also make a call to the visual narrator to generate an ontology as described in the previous section. The story hierarchy on itself contains three sub-modules:

- **Hierarchy Visualizer**: responsible for showing the ontology derived from the ontology data, there is also a feature to show and hide the details window.
- **Event manager**: to receive events from the hierarchy visualizer and handle action accordingly.
- **Data manager**: contains features for handling the data, data is directly updated to Gitlab via the Gitlab manager. Textual analysis is not done by default since the underlying concepts and relations from the story viewer and the story hierarchy are equal.

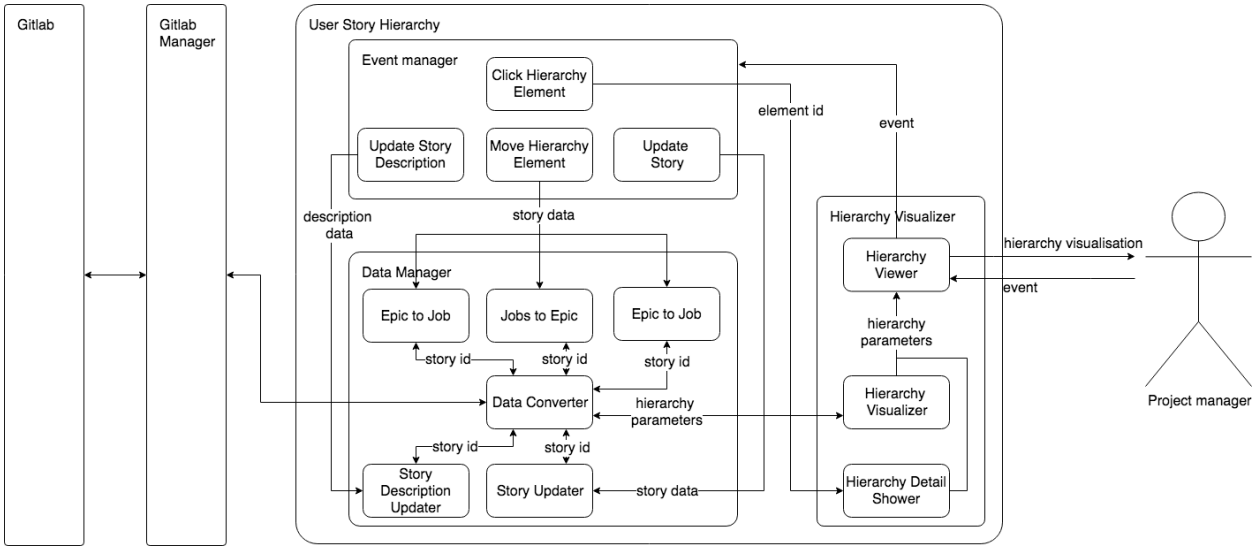

***Figure 7.13**: Story hierarchy view.*

7.4. Tooling and Dependencies

The following dependencies are used to implement the features in the Django application.

7.4.1. Langues

There are different programming languages used for building the ALAS tool, the different languages are selected for specific purposes or general purposes. The general purpose language serves throughout the tool and is used in a diverse manner:

- **Python**: is the main programming language, for the backend of ALAS. Python is mainly used in the back because the integrated tools (AQUSA and Visual Narrator) are also written in python. Python serves a lot of purposes but the main advantage of this language is it's easy to learn syntaxes and the variety of scientific libraries that are build in or supported by python.
- **Javascript**: a lot of front-end features from ALAS are build with javascript and related libraries. Closely related to are also typescript, which is a more powerful langue that can be converted into javascript. The main purpose of using javascript is reducing the numbers of calls to the servers and to do some easy calculation and data manipulation on the client side, thus saving more server power.
- **CSS/HTML**: used for interaction with the browser, HTML (hypertext markup language) is used to carry the content of the ALAS tool and CSS (cascading style sheets) are used to display this content.
- **XML/json**: are both used to exchange data with the server directly, both are formatted text files, json is more easy to use and XML provides a more structured approach for handling the data. Data transfer between the frontend and backend is done by REST-API interfaces, which is the most standard approach of transferring data through the web.

Languages used for a specific purpose are not so commonly used in the tool and often serve one specific purpose:

- **Scala**: is used by Gitlab, it is a powerful langue and especially suited for functional programming. Scala can be used by a Java Virtual Machine (JVM) but also as javascript (via Scala.js). Although the syntax differs a lot from both java and javascript, the language is very useful for doing complicated calculations with little lines of code.
- **Ruby on Rail**: is the main language of Gitlab, it is also used a lot on the internet. By it object-oriented design, it is easy to learn and stable langue.

7.4.1.1. Libraries

- **jQuery/jQuery-UI**: through jQuery and jQuery-UI are two different javascript libraries, they are closely related and use the same engines. jQuery is a library for interacting with the Document Object Model (DOM) with is a structured representation of an HTML-page as generated by the browser. jQuery-UI contains flexible UI-elements and is particularly useful for creating visual elements for user interaction. Advantages, among others, jQuery and jQuery-UI are very well maintained, easy to use, well documented, widely used and flexible.
- **CodeMirror**: is an extensive text editor, the main purpose of CodeMirror is to deliver the basics of the Story Editor text field, with build in feature to support custom text highlighting and flexible event handlers to implement features of specific user events, this library is very well suited for this purpose. Other advantages, CodeMirror is free to use without any copyright restrictions, it is well maintained and used by large corporations like Google.
- **mxClient**: forms the basis of an editor to draw application architecture. The main purpose with ALAS is to visualize the generated architectural template for the Story Hierarchy and to let users interact with this template.
- **VisJs**: is a library for displaying network graphs and also includes a lot of features for data manipulation of a functional level. Because of these two features, VisJs is used throughout ALAS, and also form the basis of the frontend data handlers. Strong aspects about this library are its callback principles and efficient algorithms to sort and filter data.
- **Require**: for loading javascript on the fly. Since there are many libraries and a lot of different scripts used by ALAS, loading times can become a problem. Downloading everything at once would take a lot of time, Require helps to reduce the loading time by only downloading the scripts that are necessary of a specific action.
- **Font Awesome**: is used for cosmetic purposes, it contains a collection of icons and fonts. The advantage of loading default symbols makes the interaction with the user more intuitive.
- **Showdown**: is used to display markup language. When clicking on the repository view, the README-file is automatically loaded, which is easy to get insights into the project. Since most projects have a readme-file in markup, the showdown parser can be of use to visualize the README-content.

7.4.2. Frameworks

- **Django**: is an easy to use server framework for python. Since it is used a lot by the industry it is well maintained a secure. It also provides a lot of flexibility and out-of-the-box functionality for fulfilling many purposes of a server.

7.4.3. Tools

- **AQUSA**: an application created by Gram Lucassen to check the quality of user stories.
- **Stanford NLP lib**: used by AQUSA to tag sentences for natural language processing.
- **Visual Narrator**: is an application to extract an ontology from a set of user stories.

7.4.4. Environment

- **Apache**: serves the different tools and servers on the backend, it is an open-source HTTP server that operates on UNIX and windows. Because of its stability, it is commonly used by the industry.
- **AWS**: is the web frame from Amazon which is used to host ALAS's servers. The infrastructure of AWS is very reliable and is the preferred choice in a lot of companies.

8. Evaluating Requirement Functionality

After implementing the functionalities described in the previous sections, multiple evaluations are to measure how well the ALAS tool reaches the requirements. So the central research question in this chapter is how to evaluate the functionalities for development support systems (**RQ-5**). Four expert-interviews are conducted to test how useful ALAS would be in practice. The main subjects of these interviews where ease of use, usefulness, correctness, and completeness. Another test is conducted to check the output of ALAS by comparing the source code structure of finished software projects with the linguistic structure of the User Stories, generated by ALAS.

8.1. Testing the Proof of Concept

8.1.1. Goal Definition

The goal of testing conceptual tooling to support User Stories in practice is to evaluate the role of User Stories in practice, using ALAS. A subject like communication and estimation can often play an important part in the successes of software projects. Also, the role of User Stories within a project can be evaluated, for example by estimating if the stories are used to their full potential. Finally, the goal of this evaluation is to check how well ALAS can contribute to getting the required information out of a User Story set.

8.1.2. Context Selection

Because of the practical context of this evaluation and to get an idea about the industrial needs in project management, the interview are conducted face-to-face at the companies location. All interviewees are professional and have years of experience in project management with User Stories. Real life problems of existing projects are discussed to get a general insight into how projects are managed and how User Stories are used within this context.

8.1.3. Hypothesis Formulation

The software industry can provide practical insights to strengthen the evaluation of the ALAS-tool and to review this study.

8.1.4. Variables Selection

With respect to the evaluation the role that ALAS can play within software projects, four variables are defined:

- **Ease of use**: where the tool is tested how intuitive the user interface is, for example, a tool that has a steep learning curve might be more efficient to use in the long run but will not be picked up easily by the industry.

- **Usefulness**: to check how well the information extracted from the User Stories would fit the industrial needs. This variable also includes the intuitivity of the representation of this information.
- **Correctness**: in order to check if the information represented by ALAS would fit the information extracted from the User Stories by the expert. Since analyzing a set of stories is usually done by hand, the extracted concepts and relations should reach human quality in order to be useful.
- **Completeness**: in line with the correctness variable, the completeness variable measures what content or visualization is still missing.

8.1.5. Selection of Subjects

The same professionals have been interviewed as in chapter five, the variety of professionals and company sizes provides more insights into how ALAS would perform in different environments. Amongst the interviewees are software developers, product owners, scrum masters and project managers, were some professionals can play different roles.

8.1.6. Choice of Design Type

Since the tools will be tested in a real working condition, a test design is not possible. Instead of setting up a controlled experiment, the tool is tested with an evaluation form to check how the tool performed. An unstructured interview is also conducted in order to get additional feedback on assumptions made in the design and the workflow of the tool.

8.1.7. Instrumentation

The list of instruments during the interviews:

- Questionnaire / interview protocol
- ALAS tool
- An example project to showcase the features of ALAS
- Presentation of the example project
- Audio recording

8.1.8. Validity Evaluation

Internal and external validity are taken into consideration when designing this test. Since the interviews are semi-structured, there is room for divergence of the required nodes. Another possible weakness is that the same interviewer conducted all interviews, thus experimenter bias could be threatening the results. To address these problems, all interviews are recorded and transcribed so future analysis can be conducted on the results to check the strength of the founded results. There are also some control questions included to test the knowledge level of the interviewees with respect to User Stories and requirement management. The situation between both interviews has also remained constant in every company.

There are also threats within the external validity of the conducted interviews, for example, there are no replications since the study is more explorative. The number of interviewees is also not suited for hard

conclusions. The generalizability across situations, however, is safeguarded by selecting a diverse set of companies of different sizes, maturity levels, and different industries.

8.1.9. Experiment Design

8.1.9.1. Research Approach

In order to check if the ALAS tool can add value in the process of requirement management or architecture, four interviews are conducted with the same companies and interviewees as section five. The main goal of these interviews is to get an idea of how useful ALAS is the setting where the selected professionals operate. Next, to the usefulness of ALAS, the interview is also focused on the correctness and completeness of the tool, for example, which features are still missing and do the existing features produce the expected results, given a case.

8.1.9.2. Data Gathering

Four different companies are selected to obtain the data from the unstructured interviews. The company will be anonymized so instead of using the name of the company, a company code is used, this company code corresponds to the company codes used in section five. The total number of employees is also recorded to provide an estimation of each company size. One employee is interviewed per company, thus four interviews were conducted in total. The employees will be anonymized but their role within the company is included, to give an idea of their perspectives.

Firm code	Sector	Number of Employees	Interviewee Role
Company A	Financial	27,000	Project manager
Company B	Management	500	UX designer
Company C	Administration	1,500	Project manager
Company D	Outsourcing	50	COO

***Table 8.1**: General information about the companies where the interview took place.*

The following nodes are used to structure the interview and the questions (these nodes correspond with the variables described in section 8.1.4.):

- Ease of use
- Usefulness
- Correctness
- completeness

8.2. Interviews

During the interviews, all different screen of the tool was being discussed, to get an overall picture of how the tool will perform in practice we looked at four variables, ease of use, usefulness, correctness, and completeness. These variables are applied to every view of the tool e.g. the story editor, the story visualizer, and the story hierarchy. The most distinguishable citations from the transcribed interviews are included, after which a Likert scale overview follows to sum the impressions from the interviewees.

8.2.1. Ease of use

8.2.1.1. Story editor

Overall the interviewees were positive about the ease of use from the tool, there was some additional feedback presented for an additional feature to make the purpose of ALAS more clear and more intuitive.

Firm code	Citations
Company A	• "<enter> is a very handy feature which could save a lot of time" • "most of the stories we encounter are already written so those need to be imported"
Company B	• "The story editor is valuable"
Company C	• "Looks good, at this moment you are imposing a default template, which is good. I think companies can work with a fixed template."
Company D	• "[...] role synonym, for example during typing, on the basis this is what you want." • "Add it to the Jira management tool, this would be a good addition, when you create backlog item you have to pass through here." • "spelling errors could also be included"

***Table 8.2**: citations per interviewee related to ease of use of the Story Editor.*

Lessons learned

- Integrate to Jira, for example, the title field.
- The TAB function is nice.
- The ENTER feature is also good, but users should be able to change the template.
- Add a spelling check.
- Include additional import features, for example, file upload.
- Add role synonym detection, so roles super "superuser" and "administrator" can be mapped.
- Add suggestions for a role when the user is typing.

8.2.1.2. Story Visualizer

Experts from different fields and perspectives diverged in their opinion about the ease of use from the story visualizer, on the one hand, experts were very enthusiastic about the visualization and could see easily see the added value. Other expert expressed their concerns related to this screen, stating that the story graph will be difficult to read when the number of stories rises.

Firm code	Citations
Company A	● "This is a very good screen, it looks like am seeing a part of my own brain" ● "really handy to communicate on a different level regarding these stories" ● "Even though I do not understand the content of the stories, I immediately start to see connections and some hierarchy from these different colors and dot sizes"
Company B	● "You can see what belongs together" ● "It is hard to understand what you can do with this information"
Company C	● "With hundreds of user stories this view would be very complicated, formulating hierarchy and structure would be easier in a matrix. "
Company D	● "The concepts of stories are visualized, which could be handy for when there are a lot of stories, this view provides a different dimension on the stories" ● "Real project visualization, becomes too much and then everything becomes important you want to know the most important parts. If you would split the stories into concepts you can mark concepts, for example as epics"

***Table 8.3**: citations per interviewee related to ease of use of the Story Visualizer.*

Lessons learned

- The opinions regarding this tool diverge.
- On the positive side:
    - ALAS can function as a mind map.
    - You can see different dimensions.
    - This view could be useful to discuss 'hidden' structures in the stories.
    - Relations between stories are getting clear.
    - Markers can be useful to order different concepts or mark them as a theme.
- Some companies where positive:
    - This view becomes complicated when more stories are being displayed.
    - It is not clear what you can do with the information from this view.
    - Viewing this information with a matrix would be better.
    - Everything can become important when the number of stories increases.

- Feedback:
    - Add an additional layer to define the selection criteria, now only Job and Epic exists, but this does not fit a lot of companies.

8.2.1.3. Story Hierarchy

The story hierarchy was overall well received, although some experts express their concerns about the flexibility of this columns by stating that companies should be able to change the columns and meaning behind these columns in order to be useful in practice.

Firm code	Citations
Company A	• "The step of getting from the view towards your hierarchy view is a bit more tricky since I think you should do an additional step where more technical information is added. So I see this as two different steps (1) visualizing the stories and supporting the process of writing user stories or (2) generating code or tasks given a set of stories. I think you should be focussing on the first one."
Company B	• "A features are not tree structures, some features come together in different modules"
Company C	• "This is very handy, you can already see the features and modules, the project structure." • "I would suggest to remove the story visualizer and focus more on this window, after all, why would you try to enrich people with this diagram while most people will shift to the hierarchy view to see the same information." • "Not everyone works with features and modules, this example is a bit confusing for me. You should define an action which is represented by a feature, a product is also clear but a module is a bit vague, perhaps you should split this concept, perhaps schedule."
Company D	• "this view is clearer because I am used to thinking in of epics with a feature"

***Table 8.4**: citations per interviewee related to ease of use of the Story Hierarchy.*

Lessons learned

- Resembles the project structure.
- Add a layer of technical information.
- Make this view more flexible, by letting the user define custom headers.
- Features are not tree structures, that way things can get easily messy.
- Focus more on the is view and less on the visualizer.

8.2.2. Usefulness

8.2.2.1. Story Editor

The experts were overall satisfied with the usefulness of the story editor. Some expert shared that the story editor would be better suited as part of an input screen in Gitlab or Jira. Another returning theme was the inclusion of Dutch language processing and spell checks.

Firm code	Citations
Company A	• "It looks like a good way to edit user stories"
Company B	• "Valuable"
Company C	• "Looks good, at this moment you are imposing a default template,m which is good. I think companies can work with a fixed template. "
Company D	• "Dutch language processing would also be interesting" • "detecting double stories would very handy also with small changes in stories" • "spelling errors could also be included" • "backlog item when creating a title it should be fired" • "Jira management tool, this would be a good addition, when you create backlog item you have to pass through here."

***Table 8.5**: citations per interviewee related to the usefulness of the Story Editor.*

Lessons learned

- Looks good
- As a Jira plugin, it would work better, for example, AQUSA should be fired when the header of a backlog item is stored
- Include spell checker
- Train the language process models on Dutch user stories.
- Detecting double stories is very useful.

8.2.2.2. Story Visualizer

The practical use of the story visualizer should be more elaborated according to some of the experts, the practicality of analyzing sentence structures could be useful, but this representation was debatable by some of the experts. Although most of the experts agreed that the representation of this information is on its way unique.

Firm code	Citations
Company A	• "It starts to get meaning, and you already are coupling action on this view." • "A lot of tools are supporting data but I don't know any tools which create images from user stories" • "data is not useful, data with context is information, and information with goals are actions"
Company B	• "The practicality of this application is not clear"
Company C	• "I don't want to see this in a graph" • "The elements are not much but this schema becomes very busy, so I would make the visualization more schematic. "
Company D	• "It did not add value, remove this view to make a clear line in the process of creating an architecture form user stories."

***Table 8.6**: citations per interviewee related to the usefulness of the Story Visualizer.*

Lessons learned

- The purpose of this view and how to use it should be better represented in the view.
- Extracting information from data is a useful aspect of this tool.
- Creating images from stories is unique.
- People want to couple actions to the concepts, additional input fields should support this.
- This view should be optional, some companies like to work with it and others see it as a distraction.

8.2.2.3. Story Hierarchy

The information presented in the hierarchy window can be very useful, but the column names and its meaning can differ between companies and even projects. The experts, therefore, stated that including a different layer of complexity would make this view even more useful.

Firm code	Citations
Company A	• "This feature already exists, it can be useful but is not very special"
Company B	• " it is not very clear what "modules" means, make the column names more flexible"
Company C	• "Also very useful but the column names indicate more about architecture, change the names to requirement management associated, like Epics, Jobs and story clustering."
Company D	• "Very useful, a good way to structure the user stories." • "Since this feature already exists in other tools, make the view more

	concise with existing features or include unique elements.”

***Table 8.7**: citations per interviewee related to the usefulness of the Story Hierarchy.*

Lessons learned:

- The column names should be changeable by adding a new layer of abstraction where the user can define the column names and configure to the tool in how to detect suitable themes.
- The column names suggest a more architectural impression of the structure while the information presented is still in the requirements gathering phase, so the column names should be something like “Job”, “Epic” and “Story clusters”.
- The clustering information can be transformed in the architecture window.

8.2.3. Correctness

8.2.3.1. Story Editor

Firm code	Citations
Company A	• “most of the stories we encounter are already written • “since your tool also includes drag and drop and copy-paste features, guess your editor can be useful in practice”
Company B	• “Usually, 20 User Stories make it to the start of a project, normally 5 of these stories express the themes of the project and serve as Epics”
Company C	• “The most difficult aspect of this approach would be our existing architecture, it would be a good addition to be able to add the existing legacy architecture, but it is very complicated since the architecture of every application can differ. ” • “The feature of changing the levels and filtering could be useful, this would help to get more structure if you have a diagram you want more structure so a search field could also be useful to find concepts.”
Company D	• “This example is too simple.”

***Table 8.8**: citations per interviewee related to the correctness of the Story Editor.*

Lessons learned

- The project structure might be composed earlier in the process.
- The number of User Stories should be limited at the initialization phase of a project.
- Code or features might be in existence, the requirements should be merged with existing features.

8.2.3.1. Story Visualizer

Firm code	Citations
Company A	• "you visualize unsorted data so it gets meaning and people start to think about it. "
Company B	• "Difference between an epic and user story depends on the team" • "Within the product, it does not matter who is working on the user story, story points should compensate for differences in work speed" • "Comparable workload can differ between teams" • "There are a lot of risk factors related to the workload"
Company C	• "Overall the visualizer does not add much value, the hierarchy is very powerful. Secondly, you should add more flexibility, especially in the hierarchy view, the entities should be definable by the user. "
Company D	• "we do this in Jira if you could tag it would be useful but you need a step in between" • "If I see a word different times, if I see payment this can mean different things" • "The linguistic structure of user stories and relation between stories are different things" • "The most important reason to couple a concept marker to an Epics is when you are four sprint's future, your stories in the first sprint, other programmers do not know what is defined in the first sprint."

***Table 8.9**: citations per interviewee related to the correctness of the Story Visualizer.*

Lessons learned

- Stories are not always displayed correctly.
- Linguistic processing could improve when for example taggers are trained on stories datasets with more data.

8.2.3.1. Story Hierarchy

Firm code	Citations
Company A	• "The step of getting from the view towards your hierarchy view is a bit more tricky since I think you should do an additional step where more technical information is added"
Company B	• "For one feature, there could be ten user stories, so it is not a hierarchy"
Company C	• "The cohesion with modules and Epics is more statistically which should be translated to a more practical context."

Company D	• "In practice, you would first define the epics or jobs"

***Table 8.10**: citations per interviewee related to the correctness of the Story Hierarchy.*

Lessons learned

- This view is too limited, architecture should be added from an existing project to get a better-suited architecture.

8.2.4. Completeness

8.2.4.1. Story Editor

Overall, the experts were satisfied with the features presented in the story editor. However, some suggestion where made to make the story editor more flexible and more useful in practice.

Firm code	Citations
Company A	• "Stories we encounter are already written so those need to be imported."
Company B	• "Valuable"
Company C	• "You write the user stories in a very simple way, I know for practice that the stories are more complicated. For example, an "and" is not permitted, but in practice, you see this a lot. If you take more difficult concepts, with more actions like click and drag, my advice would be to find related terms, it is not possible to use the same terms." • "Overall you can also add some settings, for example, to specify the user story templates, and which product you want to use (features, business components etc.) so if you can create a template with components where the user will couple different names to this entities. "
Company D	• "detecting double stories would very handy also with small changes in stories" • "spelling errors could also be included" • "backlog item when creating a title it should be fired"

***Table 8.11**: citations per interviewee related to completeness of the Story Editor.*

Lessons learned

- Spell checking can be included.
- Dutch sentence analysis can be added.
- More flexibility in selecting templates can be added.
- Detecting double stories can improve by integrating synonyms.
- The story editor could be more interactive, for example by presenting suggestions while typing.

8.2.4.2. Story Visualizer

The suggestions of including features of the story visualizer diverged between experts. Some experts suggested to remove this window from the left sidebar and present it more like an additional option. Another expert would suggest to focus fully on this window and include a lot of additional features to present more dimensions to the information, for example by generating a 3-dimensional representation of the data.

Firm code	Citations
Company A	• "Everything starts functional, and the owner of this problem should tell this story (storytelling) which already possible with this tool. So the two steps should be (1) this is the problem (2) these stories are added or updated, the best thing would be to add stories, for example, an architect should be able to add technical criteria. The stories I have to send you all used the correct template but additional constraints and decision soon become prosa."
Company B	• "You don't want to document to mutch" • "Everyone in the team should have a clear meaning of the whats and whys" • "Flexible planning is like shooting on a moving target, you never know what to expect"
Company C	• "searching and filtering are very important aspects in this view, in which cases will I reach my goal."
Company D	• "If you would split the stories into concepts you can mark concepts, for example as epics. Epics and modules are marked automatically" • "mark as a Job, we use Magento it is a platform, if you would provide all user stories you would have many jobs in practice you will have one Job if you can tag it would be very valuable."

***Table 8.12**: citations per interviewee related to completeness of the Story Visualizer.*

Lessons learned

- Add more flexibility, for example, ways to detect nodes or let the user define rules to auto select nodes. Also, add filtering and search options

8.2.4.3. Story Hierarchy

The completeness of the story hierarchy was difficult to gauge because of the diverse views of the experts regarding this view. Overall, the expert agreed that the basis of this contained a lot of useful features, but there are many different features that should be included to increase the usefulness in practice.

Firm code	Citations
Company A	• "In advance, you can, for example, think of some restrictions for this view, so the descriptions can be functional but you can also think about the psychology of humans regarding information processing."
Company B	• "At the start of a project, epics and goals are being conceived, the backlog starts to get a shape. The problem is not where to start but understanding what the purpose of the project is, and checking if you are on the right track with your team. Also, customers and other stakeholders should be involved with the project. The best way is to start very small and build the project step by step."
Company C	• "The tool can make proposals, but as a user, you should be able to change this, then the tool could recompute the visualization. This recomputation could be better suited for the architect. " • "This view is very valuable, I have to admit, but in this form, there are still things missing in terms of flexibility."
Company D	• "It would be useful to click on an epic and see which stories are related to this cluster. In tools like Jira epics are linked to stories and eventually to the code, then you have the complete story, I would make a Jira plugin from this. "

***Table 8.13**: citations per interviewee related to completeness of the Story Hierarchy.*

Lessons learned

- Make column names more flexible.
- Shift stories into job and vice-versa.
- Include more features to shift between architecture and story clustering.
- Link the presented information more to existing artifacts in Jira or Gitlab.
- Further integrate this window in a Gitlab project where changes are pick-up more easily.

8.2.5. Interview summary

The sentiment of the interviewees can be summed by a Likert scale, see *table 8.14*. Overall the interviewee's were most existed about either the story visualizer or the story hierarchy, which indicates that the information presented by both of these views is value for the process of acquiring requirements. The story editor has overall received the highest appreciation, but this can be influenced by the fact that every expert is already familiar with writing user stories, so a set of tools to support writing these stories is well suited with their daily practice.

Firm code	Story Editor	Story Visualizer	Story Hierarchy
Company A	+	++	0
Company B	++	0	0
Company C	+	-	+
Company D	0	0	+

***Table 8.14**: Likert-scales per screen for every company, extracted from the interviews.*

8.3. Code Analysis

Next to the interviews, ALAS is also evaluated by comparing the structure of existing projects with the output from the ALAS-tool. After a short of every tool, the source code is reversed engineered to get insights into the structure of the project. This representation of the project structure is then compared with the out of ALAS.

8.3.1. Existing projects

For this analysis, three finished projects are selected with respect to the presence of source code and User Stories. The projects, Camper+, Frictionlessdata, and Recycling System were built by students at Utrecht University for a course in software architecture and are publically available. One selection criteria was that the source code and User Stories where presented, that the projects have a working website and that the projects are completed, that is there is a minimal viable product in place that includes all features described by the user stories.

8.3.1.1. Data Description

The data is provided by a professor at Utrecht University, who made a selection of finished projects for a students assignment. For these projects, a selection was made to make sure all information was present for making a useful analysis. The selected projects are all built by students and are publically available.

8.3.2. Camper+

8.3.2.1. Project Description

Camper+ provides tools for camp management, the tool can be used and is available online (see https://camperapp.herokuapp.com/). The source-files are accessible via GitHub, where the corresponding user stories can also be found. There are forty stories associated with Camper+ and the code can be roughly divided into three themes (child, parent, and administrator), see *figure 8.1*.

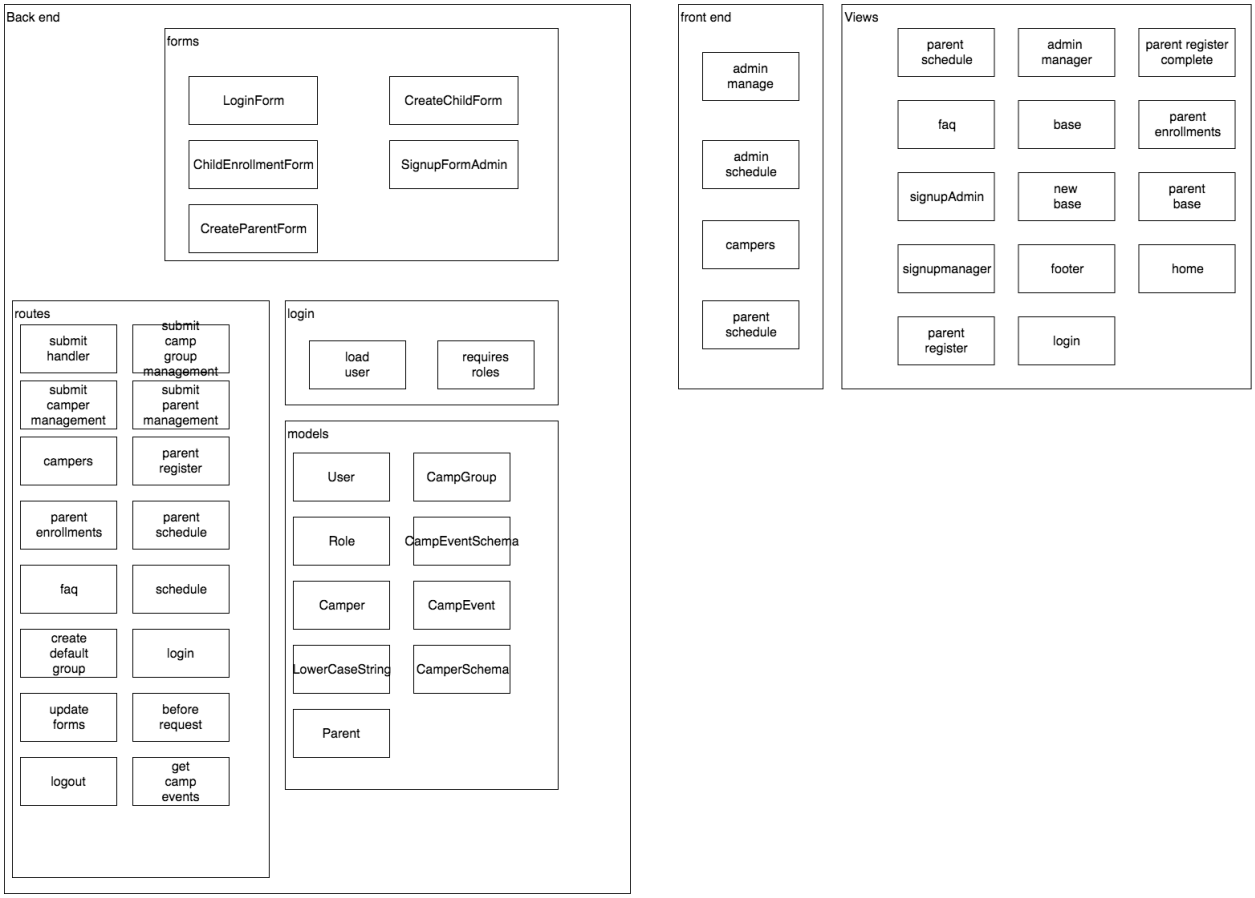

Figure 8.1*: project structure of Camper+, extracted from the source code.*

8.3.2.2. Structure generated by ALAS

When analyzing the User Stories related to this project, ALAS found the following themes for products:

- Administrator
- Camp

In total, the following modules were generated by ALAS:

- Consent
- Counselor
- Information
- Track
- Child
- Kid
- Form
- Camper
- Parent

- Features

On a high level, ALAS constructed the following linguistic structure:

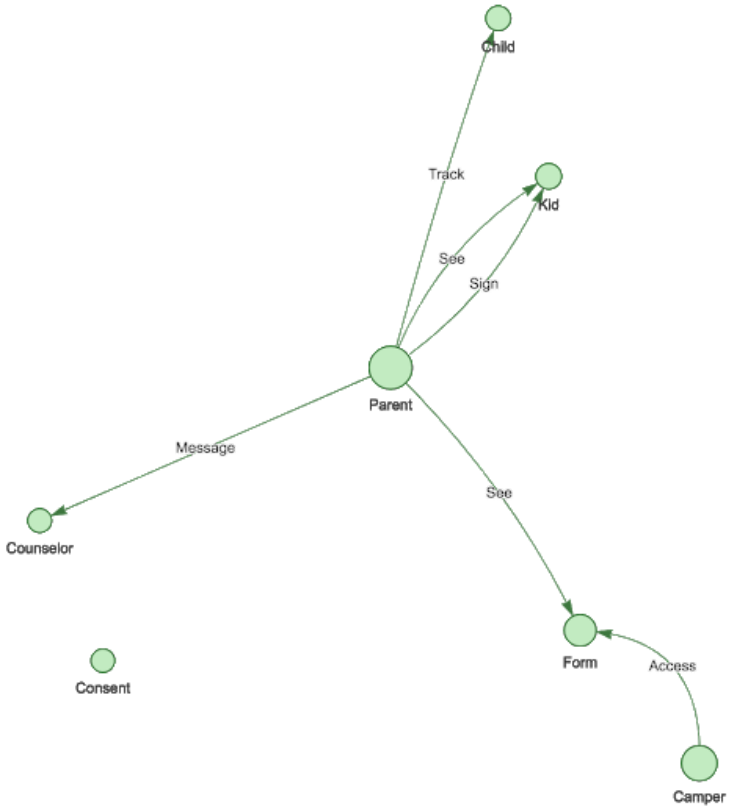

Figure 8.2*: High-level overview of the concepts and relations, extracted by* ALAS *from the user stories from the Camper+ project.*

While you can already the most prominent structures of the stories, a detailed view from ALAS is also generated to provide more insights into the clusters of features related to the Camper+ project.

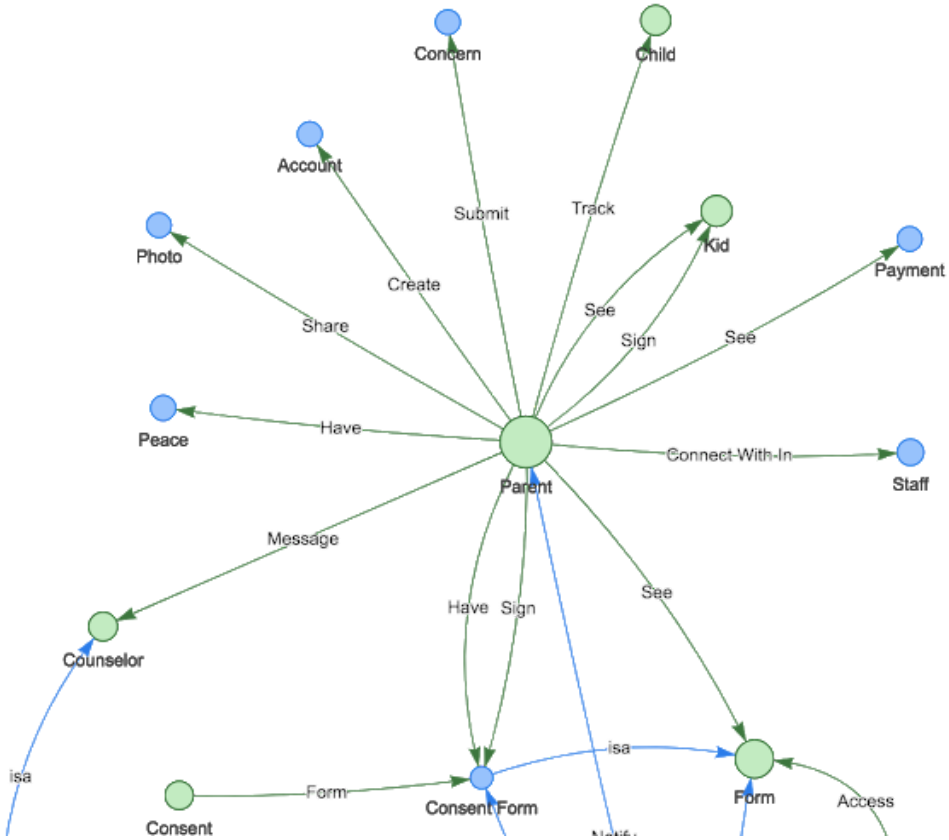

***Figure 8.3**: Detailed view of the concepts and relations of the Camper+ project, extracted by ALAS from the user stories from the Camper+ project.*

8.3.2.3. Evaluation

There are two main groups of users interacting with Camper+, guest, and organizers. Since the application is build to support camp management, scheduling activities (especially for children), request information and handling reservations are central themes in the Camper+ project. These themes are reflected in the source code, for example in modules like camper event and camper schema. Most of these themes are also presented in the structures of ALAS, for example, Information, Child, and parent can be distinguished. Also, the two main user groups are represented by ALAS, admin can be interpreted as the organizers of camp-events and camp is a close representation of the guest attending these events. Although the structure of the project is represented, there are also modules in the source code that have not been licked-up by ALAS, for example, modules related to login and register. There are also different concepts that might be merged, for example, kid and child.

8.3.3. Frictionlessdata

8.3.3.1. Project Description

Frictionlessdata provides lightweight containerization formats for data with a minimal yet powerful foundation for data publication, transport, and consumption. This tool is also available online, https://frictionlessdata.io/. The number of related User Stories is 73, which are subdivided in themes. These themes are also presented in the architecture of the tool (see *figure 8.4*).

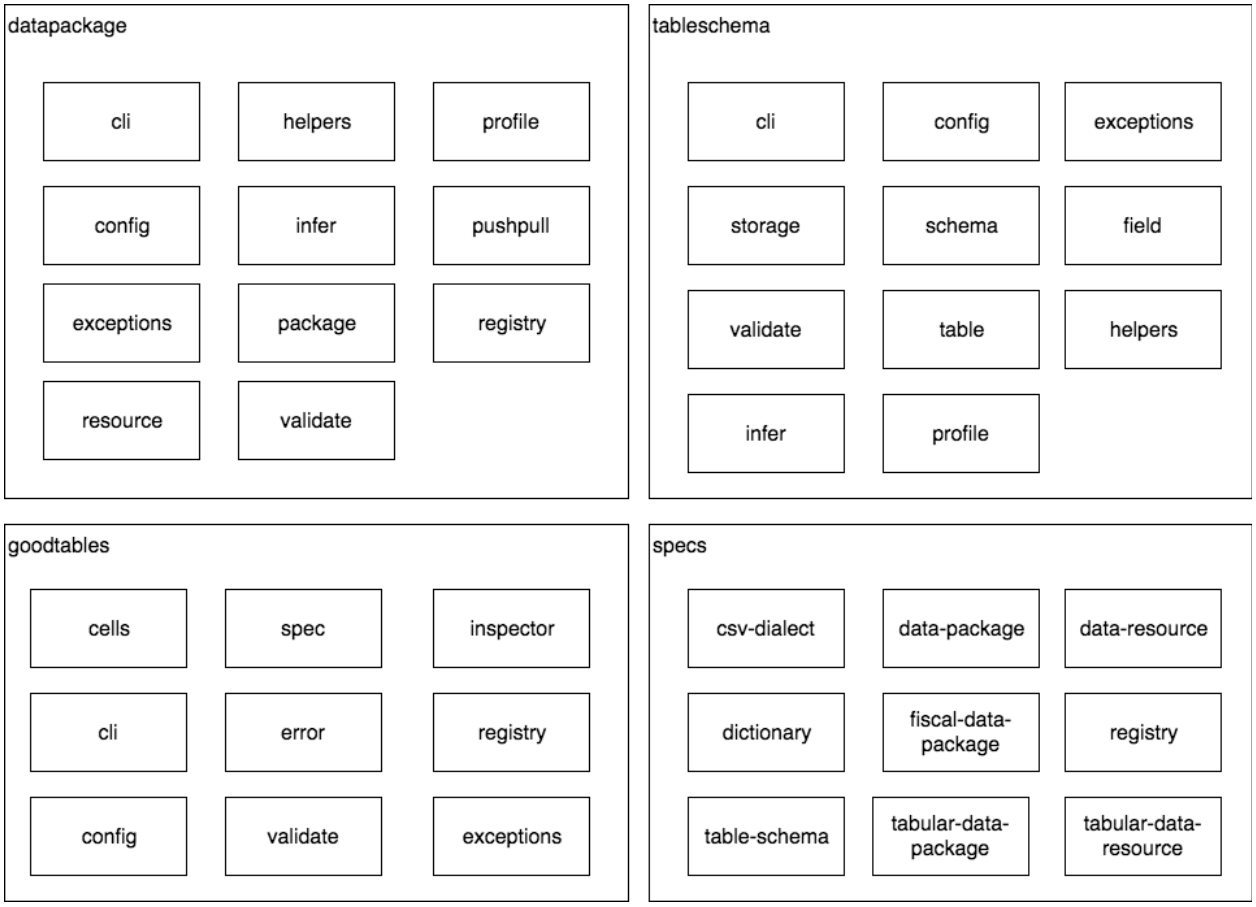

***Figure 8.4**: project structure of Frictionlessdata, extracted from the source code.*

8.3.3.2. Structure generated by ALAS

For the User stories, ALAS has extracted one theme for a product:

- Data

The themes that might serve as input for the modules, as generated by ALAS:

- Researcher

- Package
- Features
- Analysis
- Dataset
- Publisher

A subcluster of the concepts is also included:

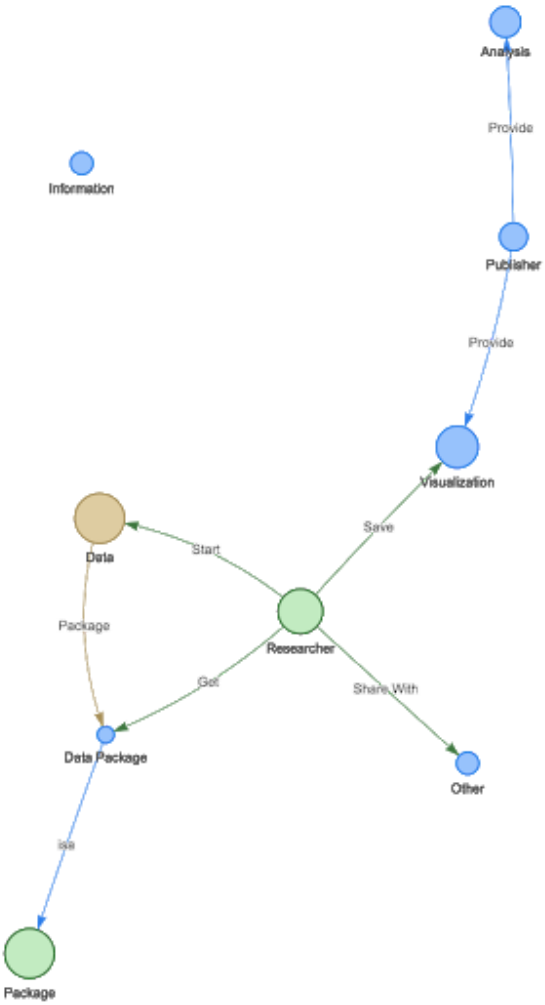

Figure 8.5*: a global view of the concepts and relations from the Frictionlessdata project, extracted by* ALAS *from the related user stories.*

8.3.3.3. Evaluation

ALAS has made a fitting suggestion of a theme on a product level, equal to the Camper+ project. The distinction of modules is also presented by ALAS, for example, package is presented as a module theme by ALAS and is also closely related to a module in the source code (datapackage). Other modules, like

specs, goodtables and table schema are not represented by ALAS as module themes. However, the suggested modules do make a logical
Grouping for architecture for this application. Now, the modules are divided into types of analysis, which is more like a tech-driven design. However, if we look at the features where users will be interacting with, the provided themes by ALAS can make more sense. Researcher and publisher can play an important role from a user perspective, thus if the application was build from a more user-centered point of view, then the modules themes of ALAS can be used as input for structuring the architecture.

8.3.4. Recycling System

8.3.4.1. Project Description

Recycling system is a tool to find recycling option in the postal code area, features like saving locations and getting information about safe disposal events are also included. The tool is available via https://warm-beach-37724.herokuapp.com/. In total there are 53 User Stories related to the functionalities of this tool, where the architecture is also represented by prominent themes from these stories (see *figure 8.6*).

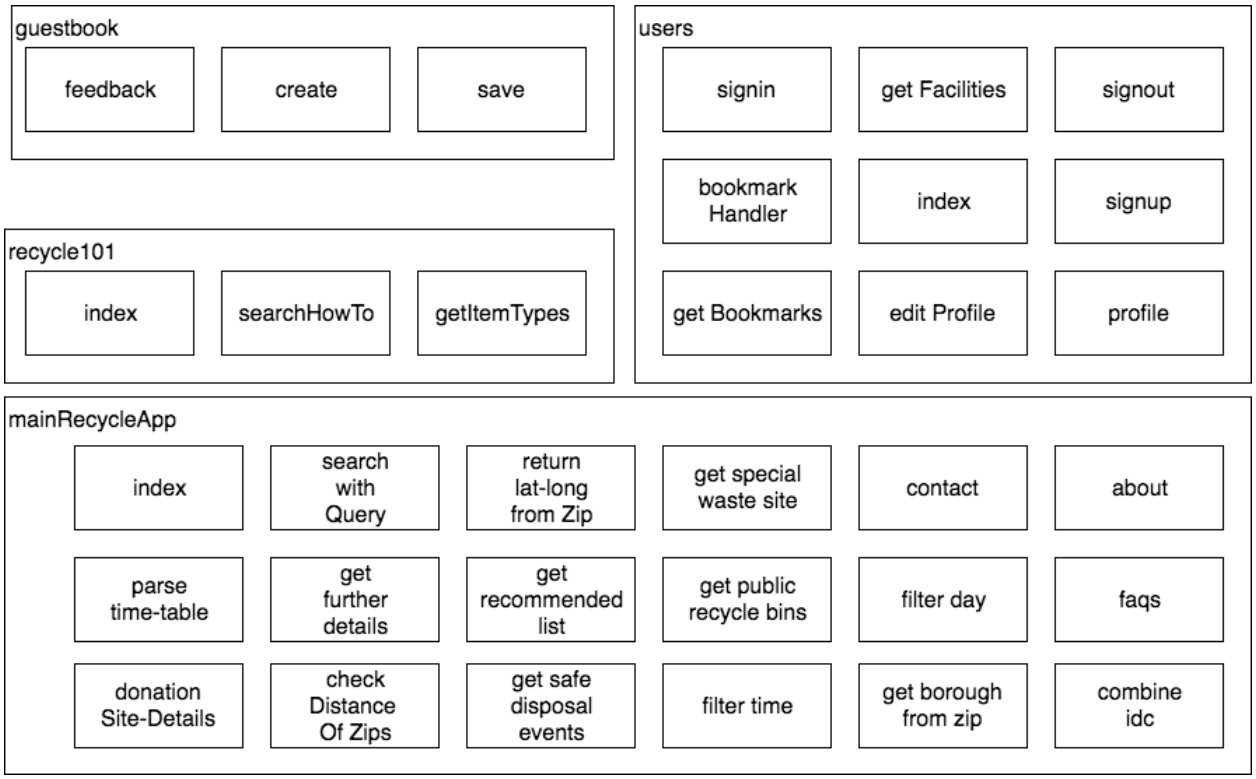

***Figure 8.6**: project structure of Recycling System, extracted from the source code.*

8.3.4.2. Structure generated by ALAS

The only product-theme presented by ALAS regarding the Recycling System is:

- User

The generated module themes are:

- Hour
- Information
- Admin

A high-level representation of the sentence structure from the Recycling System's User Stories is also generated by ALAS:

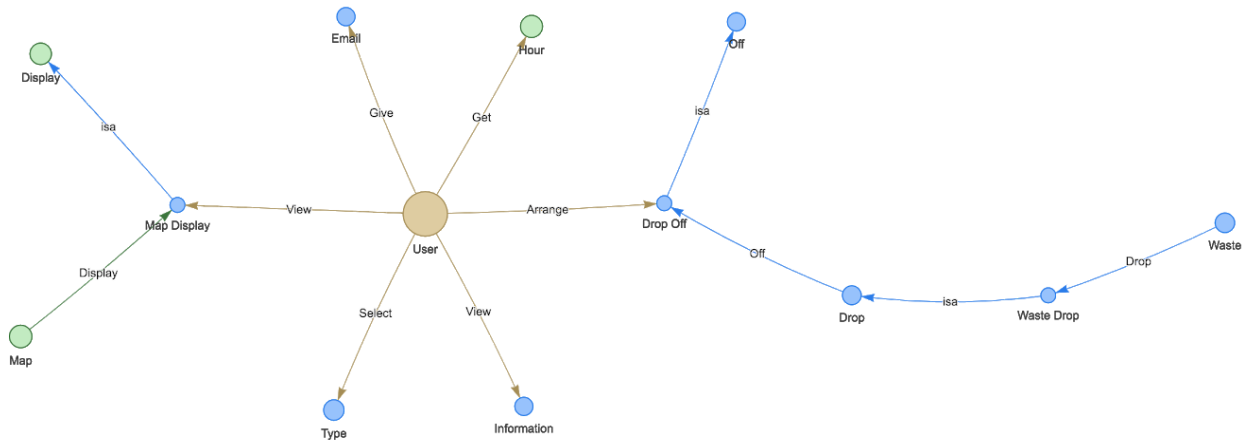

***Figure 8.7**: linguistic structure of Recycling System's User Stories, generated by ALAS.*

8.3.4.3. Evaluation

Although the user does play a central role in this application, the theme suggestion "user" is very suited for this product. As the name suggests, the central theme of the application is more related to recycling. Some sub-clusters do contain concepts and related that suit the architecture fine, but the central themes of modules are also not very useful. A closer at the raw User Stories, reveals that more of the stories start with "As a user I want to...", which might explain the generated theme to picked the recycling system tool. Moreover, the AQUSA analysis of these stories also reveals a very poor quality, since more than fifty percent of the stories contain an error or warning (see *appendix c*). Although the stories of this project might be not very well suited for textual analysis, it should be taken into consideration that the quality of User Stories in practice might also contain errors.

8.4. Conclusion

Overall, ALAS performs well in practice, the evaluation interviews reveal many interesting insights into how the tool might be improved or how to increase its usability. All experts agreed that ALAS contains a good basis of structuring a set of well-formed User Stories. The different views on the User Story data (story visualizer and story hierarchy) both provide a lot of additional insights, were different backgrounds of companies and users can play a role in how useful a specific representation of the User Stories is.

There are also improvements possible in terms of suggesting project structures from User Stories, where additional challenges in extracting contextual information from User Stories have to be overcome.

9. General Conclusions

The question of how requirement management and architecture functionality for software product development be implemented in an integrated development environment can be answered is related to the milestones as discussed in this chapter 1.2.2. I will walk shortly through every step of the project and describe the main conclusions.

9.1. Phase One (Problem Investigation)

In chapter 4 SWEBOK is used to extract the characteristics of an SDK (MS1-1-1), these characteristics have been selected and weighed to create a feature matrix (MS1-1-2). This feature matrix is used to make a comparison between three different SDK (Gitlab, GitHub and Bitbucket), where Gitlab has been selected for implementing features related to User Stories. The decisive factors were the usability of Gitlab, its open character and its popularity amongst start-ups.

9.2. Phase Two (Solution Design)

In order to get a clear overview of the desired features regarding the support for User Stories in a practical context, a literature study is conducted on User Stories (MS2-2-2) and emerging industrial trends (MS2-2-1). The literature on User Stories points to quality control, template usages, project estimates, and history tracking. From the industrial trends are pointing User Stories in the direction of supporting communication, software architecture, requirements, and version control. Next to the literature studies, there are also five interviews conducted to gain more knowledge of how User Stories are used in a practical setting and how well the literature is reflected on different businesses (MS2-2-3). The hypothesis for these interviews, that the software industry can provide practical insights to strengthen the literature review of this study, is considered to be confirmed since the requirements gathered by these case studies exceeds the number of requirements by both literature studies. The interviews are then related to the literature in section 5.2, where different insights of companies are compared and placed within the literature studies (MS2-2-4). From the literature and interviews data, are requirements extracted in the form of User Stories (MS2-2-5). These requirements have been used to compose a vision is where the role of User Stories within the process of development becomes more prominent and is more integrated with other processes (MS2-2-6). Because of this centralized role of User Stories, an ontology can also be created from User Stories which can then be used to automatically link other artifacts of a project.

9.3. Phase Three (Design Validation)

Now the requirements take shape in a vision, it is time design features are going to get implemented in Gitlab (MS2-3-6). Designing new features from a set of User Stories formed a practical challenge, equal to the problems arising in real-life software development, for example, which stories to select and how to structure the project to get a minimal viable product. To tackle these problems, the visualizations of the Interactive Narrator have been used, as explained in chapter 6. With the visualizations of this tool, the

clusters of story-concepts became clear and served as the main input for composing Job and Epic Stories. When designing the tool to support User Stories, the Gitlab style guides and an expert from Gitlab were consulted, to provide feedback on the design and meaningful information about Gitlab and its developing community (MS3-3-1 and MS3-3-2).

9.4. Phase Four (Solution Implementation)

The architectural views and the took and feel of ALAS are described in chapter 7. The implementation itself consisted of preparing a local environment to get the development started (MS4-4-1), for example installing supporting tools, setting up a local Gitlab environment, writing code, and testing the implemented features (MS4-4-2). This phase also contains a step to look for ways in evaluating the implemented features (MS4-4-3), were eventually two measures are decided, by analyzing existing projects and by conducting evaluation interviews.

9.5. Phase Four (Solution Evaluation)

The interviews again resulted in a lot of valuable information regarding the performance of the tool (MS5-5-1). The Story Editor screen proved to be very useful since it includes checks for duplicated stories, which turned out to be a very common practical problem and forced the user to correctly apply User Story templates, which is also a very common problem when working with User Stories. The Story Visualizer and Story Hierarchy view both received mixed comments, some companies where very enthusiastic about one of these screens, while other companies preferred the other screen. Interviewees did agree that the information extracted from the stories could be very useful in practice. The ease of use from the tool was also well rated by the experts because of its intuitive design. The analysis of the finished projects also showed some interesting results on the performance of the tool. By extracting the structure of User Stories, the tool was able to predict the global architectural structure of a project. An important condition, however, is that the User Stories of such a project should be composed according to the quality standards of AQUSA.

9.6. Future research

This study has brought a lot of different perspectives on User Stories together, which resulted in many new insights but also in new ideas to explore in future research. Four of those ideas are shortly discussed:

- The backlog of companies is often unstructured, contains a lot of requirement but not in the form of stories. There are interesting themes hidden in these data-sets for example when a problem keeps occurring but is incidental enough to stay under the radar. A linguistic approach to extract information from these large chunks of data would be very useful for companies.
- Now that there is a tool in existence to check the quality of user stories, there are also opening possibilities to explore existing projects and even more questions like what are the most common flows in writing user stories? And are there returning themes in different projects?
- Now the basis of user stories is explored and some insights about tasks are being explored, we can also focus on the larger picture to find relations between user stories and tasks.

- The idea of applying themes on user story clusters was considered to be very useful, as discussed during the interviews, however, the idea of predefined labels to label the cluster (in ALAS Job-theme and Epic-themes) was too frigid. Some companies proposed a different layer of complexity between defining themes and applying these themes to the story clusters, which could be further investigated.

10. References

Adolph, S., Cockburn, A., & Bramble, P. (2002). Patterns for effective use cases. *Addison-Wesley Longman Publishing* Co., Inc..

Ahn, Y., Suh, J., Kim, S., & Kim, H. (2003). The software maintenance project effort estimation model based on function points. *Journal of Software: Evolution and Process*, 15(2), 71-85.

Ali, M., Shaikh, Z., & Ali, E. (2015). Estimation of Project Size Using User Stories. In The *International Conference on Recent Advances in Computer Systems*.

De Alwis, B., & Sillito, J. (2009). Why are software projects moving from centralized to decentralized version control systems?. In *Proceedings of the 2009 ICSE Workshop on cooperative and human aspects on software engineering*(pp. 36-39). IEEE Computer Society.

Ammann, P., & Offutt, J. (2016). Introduction to software testing. *Cambridge University Press*.

Anwer, S., & Ikram, N. (2006, December). Goal oriented requirement engineering: A critical study of techniques. In *Software Engineering Conference*, 2006. APSEC 2006. 13th Asia Pacific (pp. 121-130). IEEE.

Anwer, F., Aftab, S., Shah, S. S. M., & Waheed, U. (2017). Comparative Analysis of Two Popular Agile Process Models: Extreme Programming and Scrum. *International Journal of Computer Science and Telecommunications*, 8(2), 1-7.

Barbosa, R., Silva, A. E. A., & Moraes, R. (2016, June). Use of Similarity Measure to Suggest the Existence of Duplicate User Stories in the Scrum Process. In *Dependable Systems and Networks Workshop, 2016 46th Annual IEEE/IFIP International Conference on* (pp. 2-5). IEEE.

Basili, V. R., & Perricone, B. T. (1984). Software errors and complexity: an empirical investigation. *Communications of the ACM*, 27(1), 42-52.

Boehm, B. (2000). Requirements that handle IKIWISI, COTS, and rapid change. *Computer*, 33(7), 99-102.

Bourque, P., & Fairley, R. E. (2014). *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press.

Brinkkemper S., Pachidi S. (2010) Functional Architecture Modeling for the Software Product Industry. In *Babar M.A., Gorton I. (eds) Software Architecture. ECSA 2010. Lecture Notes in Computer Science*, vol 6285. Springer, Berlin, Heidelberg

Charette, R. N. (2005). Why software fails [software failure]. *Ieee Spectrum*, *42*(9), 42-49.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., Stafford, J. (2010) Documenting Software Architectures: Views and Beyond, Second Edition. Boston, Massachusetts: *Addison-Wesley Professional*.

Coelho, E., & Basu, A. (2012). Effort estimation in agile software development using story points. International Journal of Applied Information Systems (IJAIS), 3(7).

Cohn, M. (2004). *User stories applied: For agile software development*. Addison-Wesley Professional.

Cohn, M. (2005). Agile estimating and planning. Pearson Education.

Dimitrijević, S., Jovanović, J., & Devedžić, V. (2015). A comparative study of software tools for user story management. *Information and Software Technology*, 57, 352-368.

Dingsøyr, T., & Lassenius, C. (2016). Emerging themes in agile software development: Introduction to the special section on continuous value delivery. *Information and Software Technology*, 77, 56-60.

Feiler, P. H., Gluch, D. P., & Hudak, J. J. (2006). The architecture analysis & design language (AADL): An introduction (No. CMU/SEI-2006-TN-011). *Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst*.

Fitzgerald, B., & Stol, K. J. (2014). Continuous software engineering and beyond: trends and challenges. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering* (pp. 1-9). ACM.

Garlan, D. (2000, May). Software architecture: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 91-101). ACM.

Github vs Bitbucket. (2018). Retrieved March 28, 2018, from https://www.upguard.com/articles/github-vs-bitbucket

Halpern, F. (2015). Next-generation analytics and platforms for business success: tDWI research report. *Available on: www.tdwi.org*

Heck, P., & Zaidman, A. (2014). A quality framework for agile requirements: a practitioner's perspective. *arXiv preprint arXiv*:1406.4692.

Hofmann, H. F., & Lehner, F. (2001). Requirements engineering as a success factor in software projects. *IEEE software*, *18*(4), 58.

Hoorn, van, A., Rohr, M., Hasselbring, W., Waller, J., Ehlers, J., Frey, S., & Kieselhorst, D. (2009). Continuous monitoring of software services: Design and application of the Kieker framework.

Horkoff, J., Aydemir, F. B., Cardoso, E., Li, T., Maté, A., Paja, E., ... & Giorgini, P. (2017). Goal-oriented requirements engineering: an extended systematic mapping study. *Requirements Engineering*, 1-28.

Jansen, N., & van Rhijn, J. (2018). utrecht Architecture Description Language. Unpublished manuscript, *Utrecht University*, The Netherlands.

Kassab, M. (2015). The changing landscape of requirements engineering practices over the past decade. In *Empirical Requirements Engineering (EmpiRE), 2015 IEEE Fifth International Workshop on* (pp. 1-8). IEEE.

Kattan, H. M., Oliveira, F., Goldman, A., & Yoder, J. W. (2017). Mob Programming: The State of the Art and Three Case Studies of Open Source Software. In *Brazilian Workshop on Agile Methods* (pp. 146-160). Springer, Cham.

Kavitha, C. R., & Thomas, S. M. (2011). Requirement gathering for small projects using agile methods. *IJCA Special Issue on Computational Science-New Dimensions & Perspectives*, NCCSE.

Kniberg, H. (2011). Lean from the trenches: Managing large-scale projects with Kanban. *Pragmatic Bookshelf*.

Knight, S., Rabideau, G., Chien, S., Engelhardt, B., & Sherwood, R. (2001). Casper: Space exploration through continuous planning. *IEEE Intelligent Systems*, 16(5), 70-75.

Landhäußer, M., & Genaid, A. (2012). Connecting user stories and code for test development. In *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering* (pp. 33-37). IEEE Press.

Layman, L., Williams, L., Damian, D., & Bures, H. (2006). Essential communication practices for Extreme Programming in a global software development team. *Information and software technology*, *48*(9), 781-794.

Larson, D., & Chang, V. (2016). A review and future direction of agile, business intelligence, analytics and data science. *International Journal of Information Management*, 36(5), 700-710.

Lehtola, L., Kauppinen, M., Vähäniitty, J., & Komssi, M. (2009). Linking business and requirements engineering: is solution planning a missing activity in software product companies?. Requirements engineering, 14(2), 113-128.

Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems*

documentation (pp. 24-26). ACM.

Lin, D. (1998). An information-theoretic definition of similarity. *In ICML* (Vol. 98, No. 1998, pp. 296-304).

Lin, J., Yu, H., Shen, Z., & Miao, C. (2014). Using goal net to model user stories in agile software development. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference* on (pp. 1-6). IEEE.

Lopes, L. A., Pinheiro, E. G., Silva da Silva, T., & Zaina, L. A. M. (2017, September). Adding human interaction aspects in the writing of User Stories: a perspective of software developers. In *Proceedings of the 31st Brazilian Symposium on Software Engineering* (pp. 194-203). ACM.

Lucassen, G. (2015). Introduction to AQUSA. Retrieved May 19, 2018, from https://www.slideshare.net/GarmLucassen/introduction-to-aqusa

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., Brinkkemper, S., & Zowghi, D. (2017, September). Behavior-Driven Requirements Traceability via Automated Acceptance Tests. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)* (pp. 431-434). IEEE.

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., & Brinkkemper, S. (2016). Visualizing user story requirements at multiple granularity levels via semantic relatedness. In *Conceptual Modeling: 35th International Conference, ER 2016, Gifu, Japan, November 14-17, 2016, Proceedings 35*(pp. 463-478). Springer International Publishing.

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., & Brinkkemper, S. (2016). Improving agile requirements: the quality user story framework and tool. *Requirements Engineering*, *21*(3), 383-403.

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., & Brinkkemper, S. (2016). The use and effectiveness of user stories in practice. In *International Working Conference on Requirements Engineering: Foundation for Software Quality* (pp. 205-222). Springer, Cham.

Lucassen, G., Robeer, M., Dalpiaz, F., van der Werf, J. M. E., & Brinkkemper, S. Extracting conceptual models from user stories with Visual Narrator. *Requirements Engineering*, 1-20.

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., & Brinkkemper, S. (2015). Forging high-quality user stories: towards a discipline for agile requirements. In *Requirements Engineering Conference (RE), 2015 IEEE 23rd International* (pp. 126-135). IEEE.

Mahnič, V., & Hovelja, T. (2012). On using planning poker for estimating user stories. *Journal of Systems and Software*, 85(9), 2086-2095.

Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., & Tang, A. (2013). What industry needs from architectural languages: A survey. *IEEE Transactions on Software Engineering*, 39(6), 869-891.

Mavin, A., Wilkinson, P., Teufl, S., Femmer, H., Eckhardt, J., & Mund, J. (2017, September). Does Goal-Oriented Requirements Engineering Achieve Its Goal?. In *Requirements Engineering Conference (RE), 2017 IEEE 25th International* (pp. 174-183). IEEE.
Méndez Fernández, D., Wagner, S., Kalinowski, M., Felderer, M., Mafra, P., Vetrò, A., ... & Männistö, T. (2016). Naming the pain in requirements engineering: contemporary problems, causes, and effects in practice.

Myers, K. L. (1999). CPEF: A continuous planning and execution framework. *AI Magazine*, 20(4), 63.

Negri, P. P., Souza, V. E. S., de Castro Leal, A. L., de Almeida Falbo, R., & Guizzardi, G. (2017). Towards an ontology of goal-oriented requirements. In *Proc. of the 20th Workshop on Requirements Engineering (WER) at the 20th Ibero-American Conference on Software Engineering* (CIbSE).

Ntt-review (2018) Retrieved May 19, 2018, from https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr200807sf3.html

Panda, A., Satapathy, S. M., & Rath, S. K. (2015). Neural Network Models for Agile Software Effort Estimation based on Story Points. In *Proceedings of the International Conference on Advances in Computing, Control and Networking* (pp. 26-30).

Pedersen, T. (2010). Information content measures of semantic similarity perform better without sense-tagged text. In H*uman Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics* (pp. 329-332). Association for Computational Linguistics.

Perry, D. E., & Wolf, A. L. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software engineering notes*, 17(4), 40-52.

Robeer, M., Lucassen, G., van der Werf, J. M. E., Dalpiaz, F., & Brinkkemper, S. (2016, September). Automated extraction of conceptual models from user stories via NLP. In *Requirements engineering conference (RE), 2016 IEEE 24th internationa*l (pp. 196-205). IEEE.

Salah, D., Paige, R. F., & Cairns, P. (2014). A systematic literature review for agile development processes and user centred design integration. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering* (p. 5). ACM.

Salo, O., & Abrahamsson, P. (2008). Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum. *IET software*, 2(1), 58-64.

Schmietendorf, A., Kunz, M., & Dumke, R. (2008, May). Effort estimation for agile software development projects. In *5th Software Measurement European Forum* (pp. 113-123).

Schön, E. M., Thomaschewski, J., & Escalona, M. J. (2017). Agile Requirements Engineering: A systematic literature review. *Computer Standards & Interfaces*, *49*, 79-91.

SCRUM ALLIANCE (2015). THE 2015 STATE OF SCRUM REPORT. Retrieved May 23, 2018, from https://cdn.ymaws.com/scrum.site-ym.com/resource/collection/AE0D3360-3FF1-4769-AD24-C020E7B42AD4/state-of-scrum-2015.pdf

ScrumWorks Pro Download. (2018). Retrieved May 18, 2018, from http://www.collab.net/downloads/scrumworks

Survive Disruption with Rally and CA Technologies. (2018). Retrieved May 18, 2018, from http://www.rallydev.com/

Slob, G. J., Dalpiaz, F., Brinkkemper, S., & Lucassen, G. (2018). The Interactive Narrator Tool: Effective Requirements Exploration and Discussion through Visualization.

The #1 software development tool used by agile teams. (2018). Retrieved May 18, 2018, from https://www.atlassian.com/zh/software/jira

Tothenew (2018) How to Estimate Story Points in Agile? Retrieved May 18, 2018, from http://www.tothenew.com/blog/how-to-estimate-story-points-in-agile/

Turk, D., France, R., & Rumpe, B. (2014). Limitations of agile software processes. *arXiv preprint arXiv*:1409.6600.

Usman, M., Mendes, E., Weidt, F., & Britto, R. (2014). Effort estimation in agile software development: a systematic literature review. In Proceedings of the 10th International Conference on Predictive Models in Software Engineering (pp. 82-91). ACM.

Wake, B. (2003). INVEST in Good Stories, and SMART Tasks. Retrieved May 19, 2018, from https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

Wang, C., Pastore, F., Goknil, A., Briand, L., & Iqbal, Z. (2015). Automatic generation of system test cases from use case specifications. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis* (pp. 385-396). ACM.

Wang, X., Zhao, L., Wang, Y., & Sun, J. (2014). The role of requirements engineering practices in agile development: an empirical study. In *Requirements Engineering* (pp. 195-209). Springer, Berlin, Heidelberg.

Wautelet, Y., Heng, S., Kolp, M., & Mirbel, I. (2014). Unifying and extending user story models. In *International Conference on Advanced Information Systems Engineering* (pp. 211-225). Springer, Cham.

van de Weerd, I., & Brinkkemper, S. (2009). Meta-modeling for situational analysis and design methods. In *Handbook of research on modern systems analysis and design technologies and applications* (pp. 35-54). IGI Global.

Wilensky, R. (1982). Points: A theory of the structure of stories in memory. *Strategies for natural language processing*, 345, 374.

Wynne, M., Hellesoy, A., & Tooke, S. (2017). The cucumber book: behaviour-driven development for testers and developers. *Pragmatic Bookshelf*.

Zuill, W., & Meadows, K. (2016). Mob Programming: A whole Team Approach. In *Agile 2014 Conference*, Orlando, Florida.

Appendix A: Interview protocol First Interview

1. Interviewees' background

2. Organizations' background

3. Defining User Stories

3.1. Is the term User Stories being used within your organization?
If yes:
3.2.a. What is meant by User Stories within your organization? And can you provide examples to illustrate how the term is being used?
Of no:
3.2.b. Is there a specific reason why the term User Stories is not used within your organization? And if so, can you please elaborate on this?
3.3. Has the meaning of User Stories changed over time within your organization? And if so, how has it changed?
3.4. What are activities are performed within your organization related to User Stories?

4. Instruments

4.1. Are there any instruments, tools or app related to User Stories used in your organization?
If yes:
4.2.a. How did these instruments get introduced into the organization?
4.3.a. Has there been a shift in which instruments are being used?
4.4.a. Could you elaborate on how these instruments contribute to the success/failure of your organization's goals/processes?
If no:
4.2.b. Why not?
4.5. What type of functionality would be desirable in an instrument to support User Stories?

5. Governing User Stories

5.1. Is there a strategy for managing User Stories in your organization?
If yes:
5.2.a. What is your organization's strategy related to User Stories, and how is it documented?
If no:
5.2.b. What do you think is hindering your organization in developing a strategy related to User Stories?
5.3. Where does the overall responsibility for User Stories reside within your organization?
5.4. Is there a specific department that is responsible for User Stories in your organization?
5.5. Which roles, if any, are related to User Stories (e.g. Chief Knowledge Officer, Thought Leader, or Community-of-Practice leader)?

5.6. Is there some kind of measurement used related to User Stories, and if so, by which KPIs?

6. Projects

6.1. Is there any project where User Stories are used?
6.1.a. If so, how where User Stories used?
6.1.b. Did User Stories add value to the governance of the project?
6.2. How did these projects finish?
6.3. Is some evaluation done on these projects, is the role of User Stories also evaluated? If so what was the outcome?

7. Completion

7.1. Is there anything you would like to add regarding this interview?
7.2. Do you have any questions?

Appendix B: Interview protocol Second Interview

What is your first impression on the tool as a whole?

We are now going to discuss every screen, can you tell something about the editor?

Can you tell something about the story visualizer?

Can you tell something about the story hierarchy view?

Can you tell something about the architecture editor?

Does the workflow of the tool (from top to bottom) fit your organizations workflow?

What do you think about the vision of producing source code from the architecture view, which in turn is generated by the hierarchy and the story visualizer?

Appendix C: Recycling Systems' User Stories

User Stories with AQUSA analysis

As a user, I want to be able to click on the address and it should take me to a new tab with google maps	No ends	Conjunctions			
As a user, I want to anonymously view the public information on the website so that I can know about the recycling centers around my location before creating an account.					
As a user, I want to be able to enter my zip code and get a list of nearby recycling facilities so that I can determine which ones should I consider.	Conjunctions				
As a user, I want to be able to get the hours of each recycling facilities so that I can arrange drop-offs on my off days or after-work hours.	Template mismatch				
As a user, I want the flexible pick up time so that I can be more convenient to use this website.					
As a user, I want to be able to select different types of recyclable waste that I have and get a list of facilities that accept each type that also their hours, so I can find optimal route and schedule.	Conjunctions				
As a user, I want to add the donation centers as a favorite to my profile and view them later.	No ends	Conjunctions			
As a user, I want to be able to give my email id to get notifications for the newer events as they are posted.	No ends				
As a user, I want to be able to view the maps display of the public recycle bins around my area.	No ends				
As a user, I want to be able to view the maps display of the special waste drop off sites around my area.	No ends				
As a user, I want to be able to view the safe disposal events currently being organized around my area.	No ends				
As a user, I want the flexible pick up time so that I can be more convenient to use this website.					
As a user, I want to view the user documentation about the website so that I can know how to use the web app.					
As a user, I want to be able to create an account so that I can create my own profile.					

As an admin, I want to be able to add/remove recycling facilities information so that users get the most recent information.	Template mismatch	Conjunctions			
As an admin, I want to be able to read users' feedback/complaints so that we can add more features and keep improving the service we provide to them.	Template mismatch				
As an user I want to be able to check transaction history and keep a record of them so that I can go back when I needed.	Conjunctions				
As an user I want great UI and UX from the sites so that I can have pleasurable experience when navigating through the websites.					
As an user I want to be able to access the site and do all the stuffs on all my electronic devices like iPad, iPhones, Mac, Apple Watch, etc.	Template mismatch	No ends	Conjunctions		
As an admin I want to be able to block specific users based on IP address so that I can prevent spamming on the websites.					
As an admin I want a dashboard that monitors all status related to the sites so that I can have a sense of what people do on our sites and know the service status.					
As an admin I want all data to be encrypted so that important information won't be stolen during a server breach or an attack.	Template mismatch				
As an executive I want full access to data related to my company so that I can have a sense of the company's performance.	Template mismatch				
As an employee I want to access to route planning system during my work so that I can be guided through the neighborhood.					
As an employee from HR department I want to have full information of every employees working for this business.	No ends				
As a developer I want API from the websites so that I can integrate and implement certain features in my own iOS application.					
As an user I want the tempting rewards back so that I have a reason to use this website.					
As an user I want the personal information is security keep in the database of this website so that I will not suffer by identity theft or telephone harassment.					
As an admin I want to handle all users' activities so that I can be manage more efficiently.	Template				

	mismatch				
As an company I want this website is a easy to use so that I can upload or delete stuff step by step.					
As an employee I want to get the quick notification so that I can process case in first time.					
As an accountant of a company I want all activity fees are available online so that I can easily to bill statement.					
As a developer I want to use bootstrap in the process of developing so that I can easy to design my website.					
As a developer I want to learn some UI/UX lessons so that I can develop a awesome and beautiful features website.	Template mismatch				
As a user I want to view all the locations of the recycling centers on a map so that I can check which routes to take to drop off the waste.					
As a user I want to upload my week's schedule so that I can get recommendations about recycling centers that best fit my availability.	Template mismatch				
As a user I want to link my email account to my profile so that I can get a temporary password in case I forget it.					
As a user I want to contact the administrators so that I can give feedback or ask for any help.					
As an admin I want to add recycling center information so that I can keep the database up to date over time.					
As an admin I want to view user error logs so that I can fix/review any issues that are being faced by the users of the system.	Template mismatch				
As an admin I want to onboard recycling centers to the platform so that I can increase information accuracy.					
As a superuser I want to update the recycling center information so that I can provide the latest information about the recycling center.					
As a superuser I want to view users stats so that I can view on real-time how many users have visited my recycling center information and their recyclable waste.	Template mismatch				
As a superuser I want to reply to user questions so that I can answer any questions about my recycling center.					

As an admin, I wan to be able to have a dashboard that shows usage stats/locations so that I can identify the neighborhoods with largest number of drop-offs and try to add get more facilities involved.	Template mismatch	Uniform			
As an admin, I want to be able to communicate directly with facilities to keep them updated on features that we have in our website.	No ends				
As a user, I want to be able to browse through the list of facilities and see which ones are environment-friendly so I can know for sure that my waste isn't going to leave negative ecological footprint.	Template mismatch	Conjunctions			
As a recycling facility (representative), I want to be able to update my information and the type of material I accepted to avoid any miscommunication with users.	Template mismatch	No ends	Conjunctions	Brackets	
As a recycling facility (rep.), I want to have access to user stats/schedules so that I can adjust my hours and/or upgrade equipment/capacity in order to be able to accommodate larger amounts of recyclable materials.	Template mismatch	Conjunctions	Brackets	Indicator repetition	Uniform
As a recycling facility, I want to be able to communicate directly with site admin in order to convey any issues/concerns and have them fix it.	Template mismatch	Uniform			

Legend

Uniform	warning
Template mismatch	warning
No ends	warning
Conjunctions	error

Appendix D: The Story of ALAS

Once upon a time, there was a boy called Alas who ran away from home. After many years he saw his father again and told him he became a thief. His Father looked at an old crooked knotted tree in his garden and asked his son why he had chosen this path. The thief replayed "if you had trained me while he was still young, I would not have run away, I have grown hard and misshapen like that knotted tree, who should have been straightened tied while it was still young.

This story incorporates an important lesson, early decisions will have a huge impact later-on, likewise in software projects, where the most costly flaws are often made in the first phases of the process. So can we reduce or even prevent costly flaws at an early stage?

Three steps: first you start with your **hart** (*wishes, needs, desires, requirement*) then you use your **head** (*analyze, cluster, design*) and finally, you use your **hands** (*build, implement, test*).

With these steps in mind, I've studied the field of requirement engineering with, especially User Stories and interviewed project managers from multiple companies (varying from size, market and maturity).

In practice, we see a lot of variations on how these projects are executed, but in most cases shaping a project is done by hand by one or just a few people, which makes it very difficult to see patterns in thousands of user stories related to a project.

I have dedicated my time for the last six months to find solutions for tools to support the process of getting from a set of user stories to a full-featured design, ready to implement. For this work I have looked at existing literature and tools (for example AQUSA, Garm Lucassen and the Interactive Narrator Gover-Jan) but also came up with research of my own (task analysis) and eventually build the **ALAS** tool (*Architectural Lexical Arrangement Scrutinizer*) to combine all this knowledge to support and guide project managers and developers in the initial phases of a project, so a strong root will emerge in supporting a straight project.

All and well, but how did the real Alas came about? He proved himself as master-thief, took leave of his parents and went forth into the wide world, never to be heard from again.

The story of Alas is based on a tail "The Master-Thief", found in Household Tales, by the Brothers Grimm (published in 1812 and 1815), see
https://ebooks.adelaide.edu.au/g/grimm/g86h/chapter193.html

Refinement of User Stories into Backlog Items: Linguistic Structure and Action Verbs

Laurens Müter[1], Tejaswini Deoskar[2], Max Mathijssen[1], Sjaak Brinkkemper[1], and Fabiano Dalpiaz[1]

[1]RE-Lab, Dept. of Information and Computing Sciences, Utrecht University
{L.H.F.Muter, M.Mathijssen, S.Brinkkemper, F.Dalpiaz}@uu.nl
[2]Utrecht Institute of Linguistics, Department of Languages, Literature, and Communication, Utrecht University
T.Deoskar@uu.nl

Abstract. [**Context and motivation**] In agile system development methods, product backlog items (or tasks) play an prominent role in the refinement process of software requirements. Tasks are typically defined manually to operationalize how to implement a user story; tasks' formulation often exhibits low quality, perhaps due to the tedious nature of decomposing user stories into tasks. [**Question/Problem**] We investigate the process through which user stories are refined into tasks. [**Principal ideas/results**] We study a large collection of backlog items (N=1,593), expressed as user stories and sprint tasks, looking for linguistic patterns that characterize the required feature of the user story requirement. Through a linguistic analysis of sentence structures and action verbs (the main verb in the sentence that indicates the task), we discover patterns of labeling refinements and explore new ways for refinement process improvement. [**Contribution**] By identifying a set of 7 elementary action verbs and a template for task labels, we make firsts steps into comprehending the refinement of user stories to backlog items.

Keywords: Requirements engineering, user stories, backlog items, natural language processing, sprint tasks.

1 Introduction

User stories (USs) have made their way into the development process of companies [1] and their adoption is evolving to higher levels [2,1]. USs are the starting point for specifying software that is developed, according to the agile development paradigm, through a series of sprints. The USs are distributed to the development teams that refine the USs into a number of (usually 3 to 6) so-called backlog items (but also called tasks) to break down a US into specific executable tasks for developers to carry out during the sprints.

Software specifications have been thoroughly studied from the viewpoint of their linguistic structure. Researchers have proposed approaches for finding ambiguity [3,4] and other types of defects [5] in natural language requirements, for generating conceptual models [6,7], and much more [8].

Previous work has conducted linguistic analyses of USs and defined guidelines for writing a *good* specification in agile development [1,9]. The template structure of a US "As a [Role] I want to [Action], so that [Benefit]" is often misused and many real-world USs are poorly written requirements [10]. However, there is no study on the requirements-related artifacts that stem from USs in agile development and Scrum, i.e., backlog items or *tasks*.

Table 1: Example US that has been refined into 3 tasks

US: As a webshop visitor I want to add shipping addresses so that I can send presents to my friends	
Task-1	Create ShippingAddresses records for visitors
Task-2	Update validity check for Addresses
Task-3	Add data-item for LastShippingAddress to visitor

Table 1 shows the refinement of a US into three tasks. By reading the table, one can see that tasks are the bridge between user-centered requirements (USs) and development artifacts like code and test cases. It is not surprising that the tasks are the basic constituents of sprint backlogs, i.e., they define what functionality will be included in the next release of the product.

The contribution of this paper is a linguistic analysis of a large industrial product backlog that includes 195 USs and 1,593 tasks. We study the linguistic structure of the task labels as well as the main verb that indicates what actions the developers are expected to carry out. Based on the analysis, we distill guidelines for writing tasks in a clear and consistent way.

After describing our research approach in Sec. 2, we present our linguistic analysis of the sentence structure (Sec. 3) and of the main verb in a task (Sec. 4). Finally, we present conclusions and outline future directions.

2 Research Approach

We considered a large product backlog provided to us by a multinational software development company, located in the Netherlands, and having circa fifty employees. The company's main product is a web-based platform to manage contract and tender processes of companies in the procurement industry.

The initial data consisted of 2,702 entries (backlog items), each of which being labeled as Epic, Feature, Task, or Bug. In this paper, we focus on the tasks (1,593, 59.04%). Each backlog item has an attribute that defines the development status in the product development history of several releases: New (6.49%), To Do (3.74%), Approved (1.41%), Committed (1.33%), In Progress (1.26%), Done (85.29%), Removed (0.48%).

Our linguistic analysis started with running the Stanford Part-of-Speech (POS) tagger to determine the structure of the task labels; for example, "Define (VB) box (NN) type (NN) actions (NNS) and (CC) implement (VB) them (PRP). (.)"[1] indicates that "define" is a verb, "box" is a singular noun, "actions" is a plural noun, "and" is a conjunction, and so on.

[1] The individual tags refer to the Penn Treebank tagset [11].

We experienced that the POS tagger accuracy was not perfect, presumably because task labels are hardly written as grammatically correct sentences. Two major problems we encountered were words that can be tagged as either verbs or nouns (e.g., "update") and spelling mistakes (e.g., "crate" instead of "create").

We then looked at the first-occurring verb in each task label, trying to identify recurring patterns. After tagging the unique verbs, we employed classes of VerbNet to cluster the identified verbs in families of related verbs.

Finally, we extracted a linguistic template that fits most of the tasks and that can be used as a recommended template for task label writers.

3 Linguistic structure of task labels

The goal of this analysis is to identify the most common linguistic structures in the sentences that represent tasks labels. Because of the vast number of existing POS tags, we grouped the tags as shown in Table 2. For example, verbs tagged with different tenses (present/past) are grouped into the *verb* category.

Table 2: Grouping of POS tags employed in analysis

Group Tag	POS Tags	Occurrence	%	Unique first Words
verb	VB, VBD, VBG, VBP, VBZ	1,173	73.63	70
noun	NN, NNS, NNP, NNPS	322	20.21	65
adjective	JJ, JJR, JJS	27	1.69	13
adverb	RB, RBR, RBS	27	1.69	4
pronoun	PRP, PRP$	7	0.44	2
other		37	2.32	11
total		1,593	100	165

Despite the grouping, the Stanford POS tagger identified 968 different linguistic structures that represent the 1,593 tasks, thereby showing the various ways task labels are formulated by developers.

Table 3: The ten most frequent structures of task labels

Structure	Freq.	%	Example
VB, NN, NN	130	8.17	Create tender-settings component
VB, NN, NN, NN	67	4.18	Create Messages DB tables
NN, NN, NN	25	1.57	Admin licenses breadcrumbs
VB, NN, IN, NN	21	1.32	Add filters for KO
VB, NN, NN, NN, NN	20	1.26	Implement TenderPlan actions business logic
VB, JJ, NN, NN	18	1.13	Create disqualified offers card
VB, NN	27	1.67	Create TenderProcessDefinitionLevelRule
VB, NN, IN, NN, NN	15	0.94	Bind rules per section item
VB, NN, NN, IN, NN, NN	13	0.82	Create SQL Script for AcceptedById items
NN, NN	10	0.62	Update actions

POS taggers are trained with long newswire text and not with short, sketched sentences like task labels, so to further improve the accuracy we performed a manual amendment of some tags (especially verb instead of noun). The ten

most frequent structures are shown in Table 3. In the table, we use the following abbreviations: NN = noun, VB = verb, IN = conjunction, and JJ = adjective. The most frequent pattern is a verb followed by two nouns, for example: "Create tender-settings component" (VB, NN, NN). Several variations exist that add an adjective or a conjunction to the sequence of nouns. In the top-10 list, only two structures start with a noun, which usually indicates the architectural location of the task. Task labels starting with a noun will be analyzed in future work.

Given the variations in sentence structures as presented in Table 3, we distill a template that we propose as a guideline for writing task labels. The extended Baccus-Naur form (EBNF) grammar for the template (shown below) states that a task is expressed by a verb, followed by one or more `follow` elements, each being either a noun, a conjunction, an adjective, a "to", or a cardinal number.

```
task = verb, follow, {follow};
follow = noun | conjunction | adjective | "to" | cardinal number;
```

The pattern matches 42.4% of the tasks in the dataset (676 out of 1,593). Further research will reveal more detailed patterns in the label set in order to develop guidelines for task refinement.

4 On the choice of an action verb

Task labels describe an *action* for the developer to carry out in order to implement part of a software function, or to improve existing code. We have first analyzed the first *action verb* that occurs in a task label. To do so, we employed the Stanford POS tagger and extracted the action verbs from our 1,593 task labels. This resulted in 56 different verbs, which became 81 after some manual pre-processing of spelling errors and noun-verb conversion. The 20 most frequently occurring action verbs are shown in Table 4.

Table 4: Most frequent action verbs that occur in a task label

Rank	Action verb	Frequency	Rank	Action verb	Frequency
1	Create	578	11	Bind	11
2	Modify	125	12	Update	11
3	Add	85	13	Move	10
4	Implement	79	14	Show	10
5	Change	27	15	Delete	9
6	Extend	19	16	Get	9
7	Set	18	17	Redesign	9
8	Check	16	18	Setup	8
9	Load	14	19	Fix	8
10	Remove	13	20	Review	8

The most frequent action verb is *create*, which amounts to about one third of the entire task set. This figure is a strong indicator of the feature creep phenomenon [12]. On the other hand, a very related verb such as *delete* occurs only

in 1.5%. However, while analyzing the results, we observed that quasi-synonyms exist; for instance, the *remove* verb is a synonym of *delete*.

The observed relatedness of some verbs and the quasi-synonyms motivate to obtain a smaller set of action verbs for use in task descriptions. We resorted to VerbNet [13], a taxonomy of verbs that groups similar verbs in so-called *verb classes*. For example, the class *create* (CREATE-26.4) includes, besides the namesake verb, the similar verbs *coin*, *fabricate*, *construct*, etc. We identified verb classes in VerbNet that could act as containers for multiple verbs; moreover, we performed some adjustments to cope with the domain-specific jargon of software development. This resulted in the 7 families of action verbs listed in Table 5.

Table 5: Families of action verbs in task labels

Family	Members of the verb-family
Create	add, code, create, define, design, implement, insert, make
Update	add, adjust, change, edit, extend, fix, improve, insert
Merge	bind, export, insert, integrate, invite, link, list, offer
Delete	delete, redesign, refactor, remove, renew, replace
Validate	check, evaluate, research, test, verify
Control	accept, allow, apply, bind, cancel, check, configure, control, determine
Investigate	inquire, investigate, research, search

Our analysis of the data set leads us to distill the following recommendations regarding the use of elementary action verbs in task labels:

- Each task should start with an action verb.
- The family-verb defines the nature of the development action to be performed with the code.
- The starting action verb should be in the imperative mood.
- When a suitable member-verb exists in Table 5, that verb should be used.

Table 6: Elementary action verbs for task labeling

Verb	Explanation	Example
Create	add new features	Create new tender property.
Update	change existing functionality	Update all permissions screens.
Merge	merge existing functionalities	Integrate localization in datetime picker.
Delete	remove existing functionalities	Delete offer stored procedure.
Validate	test existing functionalities	Evaluate inserted event.
Control	manage existing functionality	Control of access to box content.
Investigate	investigate potential functionality	Research angular 2.0 validation and refactoring components.

When re-analyzing our data set using our guidelines, we found many well formed task labels but also several poorly defined tables. A poorly defined task would be “Box breadcrumb component”, which could be rewritten as “Create box breadcrumb component”. On the other hand,“ Update validity check for Addresses” from Table 1 is a well defined task, for “update” is a verb in Table 5.

5 Conclusions and directions

Our linguistic analysis of a large industrial product backlog resulted in preliminary guidelines for writing backlog items / tasks in a consistent manner.

Tasks play a key role in agile development, for they bridge the problem space (the requirements) and the solution space (the architecture and the code). The tasks refine the product requirements expressed as USs. A poorly formulated task is likely to lead to issues in the developed code and sprint velocity.

This research-in-progress paper simply paves the way for future work in the field. First and foremost, we have used a single dataset in our analysis. The guidelines are likely to need some amplification, and their impact on software development needs to be evaluated *in vivo*. In the long run, we hope this research will bring insights and theories to the "wild" world of agile development.

References

1. Lucassen, G., Dalpiaz, F., van der Werf, J.M.E.M., Brinkkemper, S.: The Use and Effectiveness of User Stories in Practice. In: Proc. of REFSQ. (2016) 205–222
2. Kassab, M.: The Changing Landscape of Requirements Engineering Practices over the Past Decade. In: Proc. of EmpiRE. (2015) 1–8
3. Berry, D.M., Kamsties, E., Krieger, M.M.: From contract drafting to software specification: Linguistic sources of ambiguity. Technical report, School of Computer Science, University of Waterloo, Canada (2001)
4. Bano, M.: Addressing the challenges of requirements ambiguity: A review of empirical literature. In: Proc. of EmpiRE. (2015) 21–24
5. Rosadini, B., Ferrari, A., Gori, G., Fantechi, A., Gnesi, S., Trotta, I., Bacherini, S.: Using NLP to detect requirements defects: An industrial experience in the railway domain. In: Proc. of REFSQ. (2017) 344–360
6. Lucassen, G., Robeer, M., Dalpiaz, F., van der Werf, J.M.E.M., Brinkkemper, S.: Extracting conceptual models from user stories with visual narrator. Requirements Engineering **22**(3) (2017) 339–358
7. Yue, T., Briand, L.C., Labiche, Y.: A systematic review of transformation approaches between user requirements and analysis models. Requirements Engineering **16**(2) (Jun 2011) 75–99
8. Bakar, N.H., Kasirun, Z.M., Salleh, N.: Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. Journal of Systems and Software **106** (2015) 132–149
9. Wautelet, Y., Heng, S., Kolp, M., Mirbel, I.: Unifying and Extending User Story Models. In: Proc. of CAiSE. Volume 8484 of LNCS. (2014) 211–225
10. Lucassen, G., Dalpiaz, F., van der Werf, J.M.E.M., Brinkkemper, S.: Improving agile requirements: The Quality User Story framework and tool. Requirements Engineering **21**(3) (2016) 383–403
11. Marcus, M.P., Marcinkiewicz, M.A., Santorini, B.: Building a Large Annotated Corpus of English: The Penn Treebank. Computational Linguistics **19**(2) (1993) 313–330
12. Jones, C.: Strategies for managing requirements creep. Computer **29**(6) (1996) 92–94
13. Schuler, K.K.: Verbnet: A Broad-coverage, Comprehensive Verb Lexicon. PhD thesis, Philadelphia, PA, USA (2005) AAI3179808.