Linguistic Application Reconstruction: How Linguistic Links Support the Relationship between Requirements Engineering and Software Architecture

Sabine Molenaar (s.molenaar@uu.nl), 4075056

Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands.

August 2, 2019

A thesis submitted in partial fulfillment of the requirements for the degree of:
Master of Business Informatics.

First supervisor: Prof. Dr. Sjaak Brinkkemper
Second supervisor: Dr. Fabiano Dalpiaz

Contents

List of Abbreviations

AR	Architecture Recovery
ASR	Architecturally Significant Requirement
BDD	Behavior-Driven Development
BDT	Behavior-Driven Traceability
Epic	Epic story
FAD	Functional Architecture Diagram
FAM	Functional Architecture Model
GUI	Graphical User Interface
JTBD	Job-to-be-Done
LSI	Latent Semantic Indexing
MRQ	Main Research Question
PDD	Process-Deliverable Diagram
PDO	Product Domain Ontology
PoS	Part-of-Speech
QUS	Quality User Story
RE	Requirements Engineering
RE4SA	Requirements Engineering for Software Architecture
RQ	Research Question
SA	Software Architecture
SLR	Systematic Literature Review
uADL	utrecht Architecture Definition Language
US	User Story

1 Introduction

Software development is a fast-paced industry, with one of the more recent trends being the realization of continuous deployment. Deploying continuously also means ever-changing requirements, which is a considerable issue in Requirements Engineering (RE). One of the challenges is to make and keep requirements unambiguous, traceable and modifiable among others (Niu, Brinkkemper, Franch, Partanen, & Savolainen, 2018). However, this problem of changing requirements is not limited to RE, it also affects other software development activities, such as Software Architecture (SA). The requirements specification of a software product influences its architectural design, so if the former continuously changes, so does the latter (Nuseibeh, 2001).

The existence of relationships between software architecture and requirements engineering is well-known, but the nature of these relationships is not. The RE lab at Utrecht University has developed a model to visualize these relationships, illustrated in figure 1.

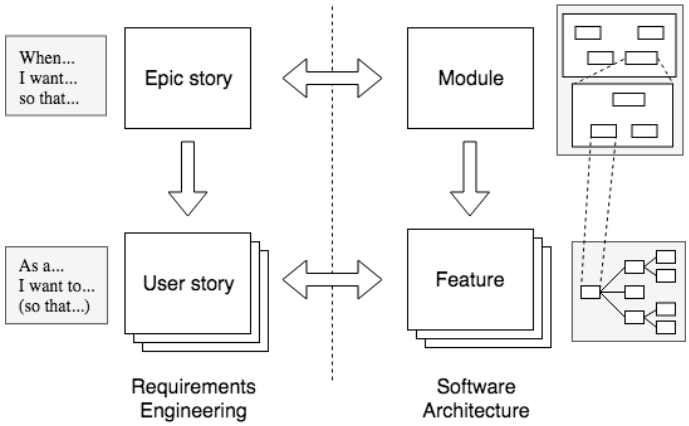

Figure 1: The Requirements Engineering for Software Architecture (RE4SA) model, including the four concepts.

On the left side RE documentation, such as Epic Stories (Epics) and User Stories (USs), describe modules and features in the SA on the right side. As was stated previously, the relationship between RE and SA is known, but the exact nature remains underexposed. However, this could provide several benefits, such as: software traceability, decrease in ambiguity and facilitation of communication between requirements engineers and architect (among other stakeholders). In this research, the connection between requirements documentation and SA is analyzed by means of a case study using real-world documentation.

1.1 Problem Statement

Requirements specify the needs of stakeholders for a software product or system, which makes them influential when designing product architectures. However, these requirements are not fixed. New requirements can be introduced and existing ones can be modified or removed. The SA should reflect these changes. Moreover, software products are known for their continuously changing requirements and thus a continuously changing architecture (Lucassen, Dalpiaz, van der Werf, & Brinkkemper, 2015a). So-called change impact analyses demand that software architects are aware of the changing requirements and can predict and track their impact on the SA (Khan, Greenwood, Garcia, & Rashid, 2008).

This relationship between requirements and architecture is important, as explained by the 'evil circle principle', which states that if there is something wrong with the requirements, the resulting architecture will be equally wrong, ultimately leading to a continuously bad project (Gilb & Finzi, 1988). Furthermore, communication flaws within the project team are listed as one of the most important issues in RE, sometimes even considered the main cause for project failure by organizations (Fernández et al., 2017). Other information systems failure contributors are (among others): inadequate requirements and related scope creep, poor communications within the project or even among the majority or all stakeholders and key internal stakeholders leaving the project (Hughes, Dwivedi, Rana, & Simintiras, 2016).

Traceability is able to tackle one of these issues, since it can support change impact analysis by relating software artifacts by means of trace links (Cleland-Huang, Gotel, Huffman Hayes, Mäder, & Zisman, 2014). An example of how such trace links can be established is by using ontologies. This ontological traceability approach, however, does not take linguistics into account, apart from extracting linguistic terms (Martens, Brinkkemper, & Dalpiaz, 2018). Ambiguous linguistic terms can either introduce trace links that should not exist or fail to identify a trace link that should be established. By considering linguistic links, ambiguity can also be play a role in determining relationships between artifacts. Moreover, if the hierarchical structure of linguistic links is examined as well, it might be possible to map the dependencies between and within software artifacts more clearly and accurately.

The RE4SA model (Brinkkemper, 2018), as was briefly mentioned earlier, links the RE process of an application to its SA: the relationship between Epics and USs on one side and the corresponding modules and features on the other. However, it is not certain that the two sides are in linguistic agreement. For example, if a US specifies a delivery as urgent, does the feature utilize the same terminology or could it be called a priority delivery? This slight alteration could influence future changes. Newly added or changes in user requirements, adjusted business goals and a changing environment all might introduce changes to the requirements and, by extension, SA of an application. Hosseini, Breaux and Niu, devised a method for creating a shared meaning of certain terms and concepts for different stakeholders, however, the meaning of the word and the interpretation of their proposed function are not the same (Hosseini, Breaux, & Niu, 2018). Thus, by checking the relationship between requirements documentation and SA documentation, the authors of requirements can see whether their descriptions were interpreted correctly. Therefore, in change management, it is important to know whether the SA and requirements documentation are in agreement. After all, the success of a system is determined by "*the degree to which it meets the purpose for which it was intended*" (Nuseibeh & Easterbrook, 2000).

Moreover, the Twin Peaks model (Nuseibeh, 2001) and, especially, the Reciprocal Twin Peaks model (Lucassen et al., 2015a) describe the importance of the synergy between requirements and architecture, of which the latter focuses specifically on agile development of software products. The challenge in product software is the nature of its development process, which has to manage, on the one hand, a continuous flow of requirements and on the other hand a continuously changing architecture. The consistency between the two is important in order to avoid misunderstandings. However, realizing this consistency should not burden the involved stakeholders with more work. The prevention of misunderstandings is of importance, since this can lead to incorrect implementation and thus potential rework, wasting time and resources. Taking the aforementioned factors into account, tools are needed to ensure the consistency between artifacts and to establish traceability of requirements, saving the stakeholders time and effort.

1.2 Research Objective

The objective of this research is twofold: validation of the RE4SA model and new insights and/or confirmation of previous findings. Firstly, the proposed research aims to verify the RE4SA model by applying it to real life cases. Secondly, it is expected to deliver new insights into and improved understanding of the relationship between RE and SA. Therefore, the main artifact produced by this research consists of the findings from the various case studies. The results can be used in future research and to analyze existing RE and SA documentations. Ideally, it proves to be a basis for the automatic generation of concept names within artifacts or even partial artifacts, such as modules or features in SA documentation and/or modeling. This can be achieved by identifying frequently used PoS tags and positioning of shared words. These can serve as a starting point from which to generate artifact names. Furthermore, they can also be used when formulating artifacts such as Epics and USs manually.

Similarly, a future goal is to automatically update SA documentation based on RE or source code and also vice versa in case of the latter. Supporting the need for tools that realize consistency and traceability in the software product development process (as was described in Chapter 1.1). Additionally, it could support naming conventions/suggestions for concepts within artifacts. The linguistic links can be utilized for planning and modeling extensions of an implemented system as well. For one, by analyzing linguistic links it is possible to determine whether some functionality needed for the extension already exists somewhere in the current system. Furthermore, the links support positioning of the new architectural components. Finally, in terms of literature, it aims

to more precisely define SA concepts, to better understand the relationship of these concepts with RE.

The remainder of this thesis is structured as follows. In Chapter 2, the research approach, including the research questions and method, is described. Subsequently, Chapter 3 contains a literature study to support the concepts and domains included in this research. The theoretical background consists of four main topics, namely: RE, SA, RE4SA and linguistics. Then, in Chapter 4, the preparation and selection of the case studies are described. This also includes an overview of the steps to be performed for each case and a summary of the case study study execution. In Chapter 4.1.3, the cases are analyzed according to the case study steps and theoretical background as described in the literature study. Chapter 5 discusses the analysis per case, consisting of a dependency analysis, a comparison and the links between Epics and modules, as well as USs and features, identified semantic frames and observed synonyms and homonyms. In addition, the meaning of the term functionality in the two domains in practice is examined. The last part of the analysis is concerned with the quality of the Epics and USs. After which, in Chapter 6, the findings are extracted from the analyses and used to answer the MRQ and sub-RQs. The findings per case are presented, as well as the results of the research as a whole. Finally, the strengths and limitations of the research are discussed, as well as future research directions and the overall conclusion, in Chapters 7 and 8 respectively.

2 Research Approach

The main goal of this research, as mentioned in Chapter 1.2, is to validate a theoretical model by means of a case study. The theoretical model that is considered is the RE4SA model, as depicted in figure 2. To improve legibility, the story templates and architecture examples have been omitted in this version of the model.

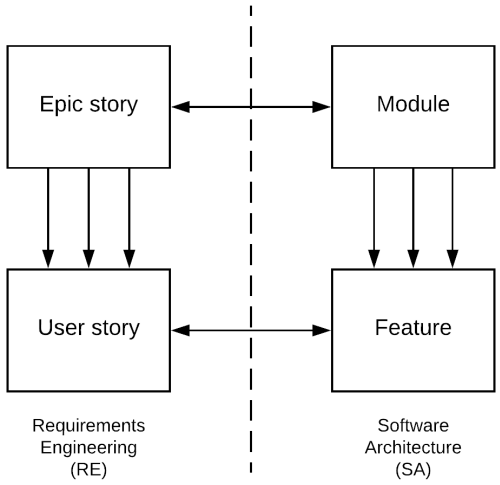

Figure 2: The Requirements Engineering for Software Architecture (RE4SA) model.

The RE4SA model states that RE and SA are related through their elements. Epics can be connected to modules. Meaning that modules implement Epics and that Epics describe modules. On a lower level, the same is true for USs and features. Furthermore, while Epics can be refined into USs, modules contain multiple features, which implies that the hierarchical structures in both domains are similar in granularity. By reconstructing an existing application based on its linguistics, new insights into the relationship between requirements and SA can be obtained. This reconstruction entails the visualization of the SA (using any and all architecture documentation available) and comparing it to the requirements documentation. This comparison mainly concerns the linguistic terms used in the artifacts at hand.

2.1 Research Questions

The following main research question (MRQ) was formulated based on the aforementioned goals and problem statement:

Can linguistic links be identified between the two domains of RE4SA using application reconstruction?

In this study application reconstruction refers to the reconstruction and recovery of artifacts and artifact documentation. While USs have a high adoption, the other artifacts are less prominent. For Epics, the term 'reconstruction' is used, since they are formulated based on RE documentation. For functional architecture artifacts (modules and features), the term recovery is used, since they are recovered from an implemented system.

In addition, five sub-Research Questions (RQs) are stated to help answer the MRQ and further guide the research:

1. What is understood as a feature in literature and in practice?
2. What is understood as functionality in literature and in practice?
3. Is there a linguistic relationship between the names and descriptions of Epics and USs and modules and features?

4. Are the dependencies between Epics and USs the same as their corresponding modules and features?
5. Is there a one-to-one relationship between USs and features?

Since all RQs are based on the same theoretical model, figure 3 illustrates where the sub-RQs are located in the RE4SA model. The main research question is concerned with the model as a whole.

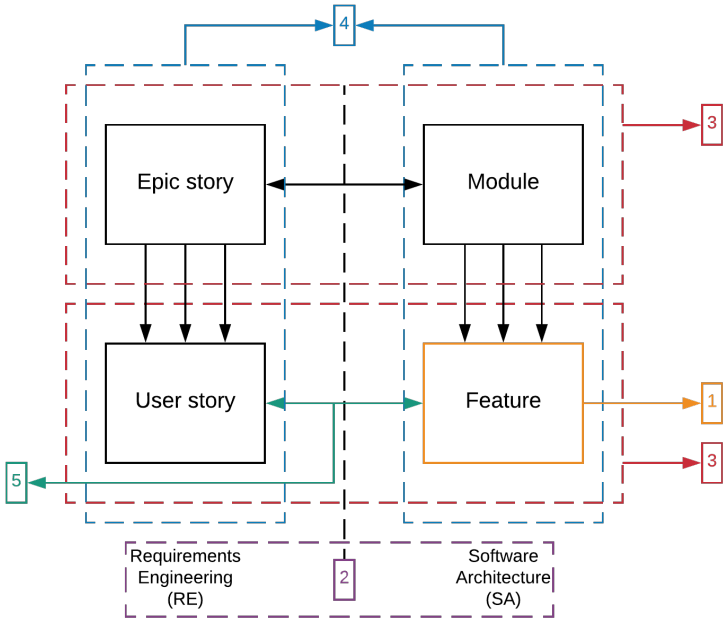

Figure 3: Visualization of the RQs in relation to the RE4SA model.

As the formulation may suggest, sub-RQs one and two can be partly answered using literature research. In an attempt to verify the results of this research, the questions will also be considered in practice, during the case study. Sub-RQ 1 is solely concerned with what is considered to be a feature in the SA and whether these are usually of the same size within the architecture. The results of the literature study and the case study are compared to see whether they are in agreement. Sub-RQ 2 uses the same approach, although it also considers RE and not solely SA. Thirdly, as shown in figure 3, the relationships between Epics and modules and USs and features is investigated in terms of linguistics. This sub-RQ takes a 'horizontal' approach, meaning it pertains to the relationship between RE and SA concepts. Contrarily, sub-RQ 4 regards the model 'vertically', being involved with the concepts within RE and SA. However, apart from these dependencies within the domains, this sub-RQ also requires a comparison of the dependencies in the two domains. Finally, sub-RQ 5 specifically examines the relationships, or rather the cardinalities of these relationships, between USs and features.

In light of the theoretical model, hypotheses can be formulated for sub-RQs three to five. According to the RE4SA model and given that SA is based on RE and vice versa, it is likely that there it a linguistic relationship between names and descriptions of Epics and USs and modules and features. If the theory proves valid, the answer to sub-RQ three should be yes. Similarly, the dependencies between between Epics and USs should be the same as the dependencies between their corresponding modules and features. Given the previous expectations, sub-RQ five should also be true, meaning that there is a one-to-one relationship. If, for any of the three sub-RQs, the hypotheses need to be rejected, it is important to investigate whether this results from the theoretical model being invalid, or potential errors and/or flaws in either the RE or SA domain. Since the objects of study are man-made, it is possible the grounds for rejection were introduced by human errors.

As was stated previously, the first two sub-RQs are partially answered in a literature study. Sub-RQ 1 is addressed in a systematic literature review (SLR) that examines the various definitions of the concept 'features' used in research. Moreover, the concept is researched in practice, by

studying what constitutes as a feature in SA. This is done by means of SA models, explained in detail in Chapter 3.2. The same approach is utilized for sub-RQ 2, although the concept 'functionality' is studied in practice by looking into the RE domain, as opposed to the SA domain. Sub-RQ 3 is studied by extracting linguistic terms from RE and SA documentation and comparing them. Similarly, for sub-RQ 4, the linguistic links are analyzed as well as the dependencies among them, again comparing the artifacts of both domains. Finally, sub-RQ 5 specifically investigates the relationships between USs and features. Since the case study-related sub-RQs are based on linguistics and man-made artifacts, they can be subject to interpretation. Therefore, the extracted linguistic links are evaluated to determine how likely or unlikely they are to be correct. Given the results of all sub-RQs, the MRQ can be answered.

Before going into more detail about the research and its theoretical background, the proposed term 'linguistic link' should be defined. Based on the definition of software traceability by Cleland-Huang, Gotel and Zisman (which can be found in Chapter 3.7) (Cleland-Huang et al., 2014), the following definition can be formulated for 'linguistic traceability': "*Linguistic software traceability is the ability to semantically interrelate any uniquely identifiable software engineering artifact to any other*". Such links can be referred to as linguistic links and in short can be described as a linguistic relationship between software artifacts. To be more specific, linguistic links consist of linguistic terms that can be matched. As for the former, linguistic terms refer to 'semantically meaningful' words, oftentimes nouns, verbs (modular verbs excluded) and adjectives, as most other types of words are used to formulate a coherent sentence, but are not essential to its meaning or purpose. Along those lines, words that are part of the template are omitted. Secondly, matches can be defined as the ability to identify a linguistic term in two artifacts (on the same level of abstraction) or a linguistic term and its synonym. If the match is unique, as in the only match to be made including those terms on the level of abstraction, it can be considered a linguistic link.

2.2 Research Method

The case study requires existing artifacts to support the different domains in the RE4SA model. However, if certain artifacts are not (readily) available, they are reconstructed or recovered using existing documentation and artifacts. An example of a recovery is the creation of feature diagrams using the source code and Graphical User Interface (GUI) as input.

The RE4SA model is thought to be applicable and relevant in most, if not all, cases. Therefore, a multiple-case design is used to test the theory. By analyzing multiple cases and having multiple sources from which to draw results, an attempt to verify and validate the RE4SA model can be made. If the model is correct, the results should be replicable across the different cases in the case study. To be more specific, a sequential, direct replication multiple-case design is used. This approach stipulates that a case must be completed prior to starting the next one. Due to this sequential order, it is possible to slightly adapt the case study approach and adjust or reconsider previous findings. Each case should follow more or less the same steps as the others in the case study (R. K. Yin, 1981). Case studies are most often exploratory and make use of qualitative data (Runeson & Höst, 2009). Runeson and Höst define five main steps for a conducting a case study:

1. Case study design
2. Preparation for data collection
3. Collecting evidence
4. Analysis of collected data
5. Reporting

The case study design, as was mentioned previously, is a multiple-case design, performed sequentially. The different phases and activities are illustrated in a Process-Deliverable Diagram (PDD) in figure 4. While the main steps remain mostly the same for each case, the most variation is expected in the preparation for data collection step. The research requires multiple software artifacts as input, the most important ones being Epics, USs, a functional architecture and feature model. Additional artifacts, such as requirements documentation and test cases, can also be of use, but are not crucial to the collection of evidence step. It is possible to recover such missing artifacts if necessary. For example, if the functional architecture is missing, it can be reverse engineered using the source code and GUI. Similarly, a feature diagram can be extracted from both the functional architecture (or source code) and the GUI. USs can also be reconstructed, however, this should be avoided, as there is a high risk of subjectivity involved. Epics, however, are a different matter

entirely. Epics as they are used in relation to the RE4SA model (as described in Chapter 3.4.3) are rarely, if ever, used in practice or research. Approaches similar to that of Epics are more likely to exist (such as themes or USs groups). These can be used to formulate proper Epics, since using the SA documents would reduce validity. Finally, the SA artifacts should not be recovered based on the RE artifacts, as this would negatively affect the validity of the research as well.

When all artifacts are gathered and/or reconstructed/recovered, the activities are identical for all cases. First, the SA artifacts are analyzed per module, which also include the corresponding features. Similarly, the RE artifacts are analyzed per Epic, which include the corresponding USs. Based on these analyses, linguistic links can be extracted. These linguistic links will then be compared on four different aspects. Firstly, how similar the linguistic terms are, which is also referred to as the evaluation of linguistic links. This similarity considers whether the terms are identical or synonyms, among others. Secondly, the relationships are analyzed, to see which concepts and artifacts can be related using the linguistic links. Thirdly, after these activities have been completed, the dependencies among the various artifacts and concepts will be investigated. Fourthly and finally, these dependencies among concepts will be compared to what has been defined in theory, so in this case the RE4SA model. The case study and results analyses phases are executed three to four times, as this can be considered sufficient according to Yin (R. K. Yin, 1981). The analysis is not solely focused on theory testing, but also on theory building. The former is done, in short, by testing hypotheses the RE4SA model is based on. The latter is achieved by further defining the relationship between RE and SA from a linguistic perspective, which is likely to lead to new hypotheses.

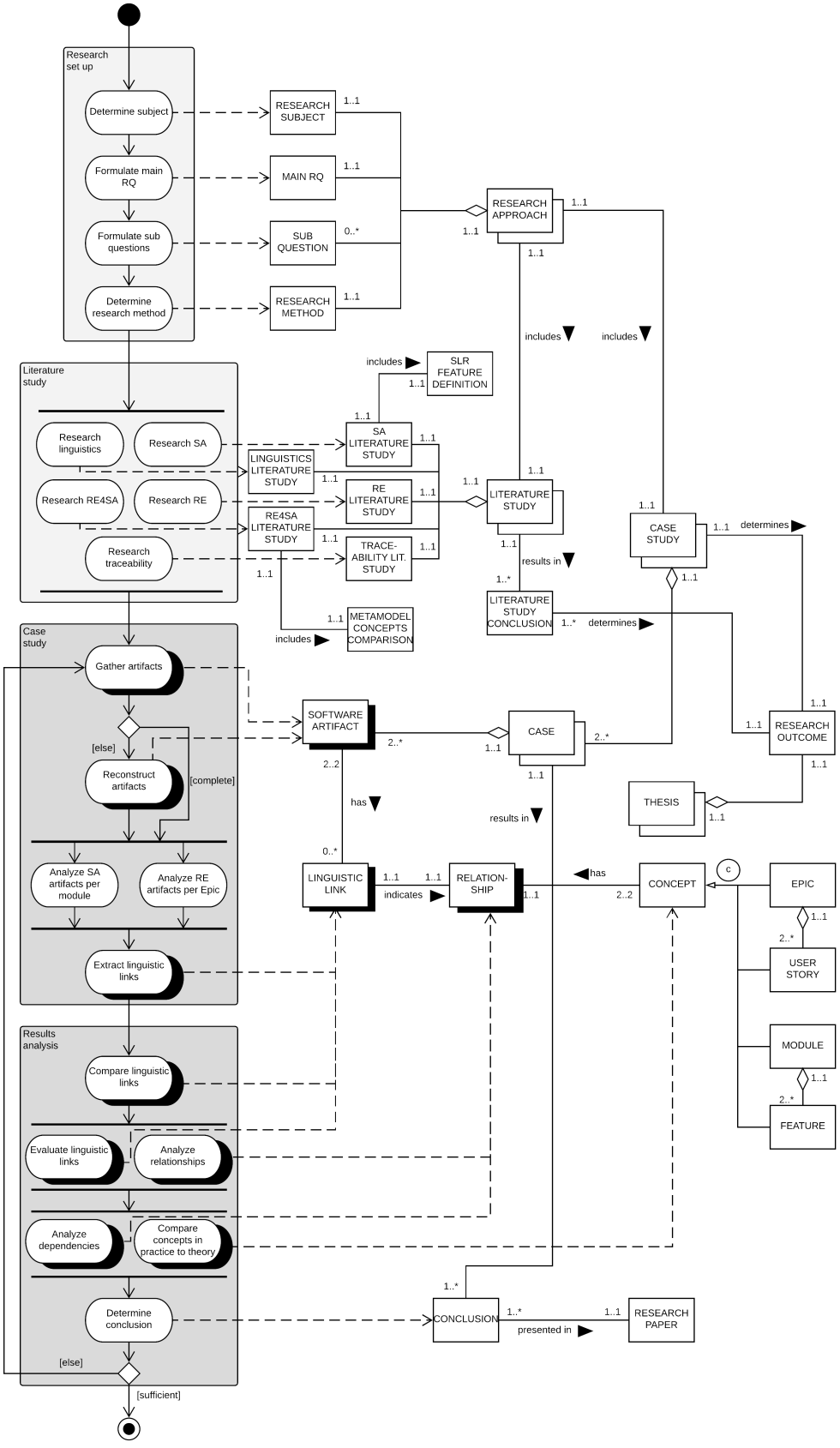

Figure 4: Visualization of the research approach using a PDD.

2.3 Literature Study Approach

In the literature the main subjects used in and relevant to the research are discussed. Since the sole purpose of this study is to support and guide the research, only the required literature is included (it is by no means a systematic literature review). Sources were gathered by searching (using various search terms related to the topics) on Google Scholar and by snowballing (forward and backward searching) the results. The results of this search are presented and discussed in Chapter 3. In addition, to answer sub-RQ 1, a systematic literature review (SLR) was conducted. The results of this SLR are presented and discussed in Chapter 3.3.

2.4 Case Study Approach

As was mentioned earlier, a multi-case study is used. The steps performed in the case study are repeated for every case (refer to Chapters 4.1.2 and 4.1.3. An estimated three to four cases are used to not only gather data, but also validate results obtained from the other cases. Cases are selected based on several selection criteria, as presented in Chapter 4.1.1. To be able to learn from previous cases (for example to prevent any issues or challenges), each case is fully analyzed, prior to the next one being started.

3 Literature Study

The literature study presented in this chapter focuses on the main subjects related to this research, these being RE, SA and linguistics. For each subject, the main topics and concepts of importance to or utilized throughout this thesis are discussed. Firstly, in Chapter 3.1 the subject of RE is examined, followed by a chapter on SA in 3.2. The latter also discusses modeling techniques and the concept of features (in relation to RQ1). The RE4SA model and its concepts are explained in-depth in Chapter 3.4. This chapter describes various templates for RE concepts. To improve readability these are written in verbatim. In Chapter 3.5 functionality is examined in the context of RQ2. Subsequently, in Chapter 3.6, various topics related to linguistics are discussed, such as ambiguity, parse trees, ontologies and linguistic structures. Finally, the relationship between RE and SA and its use is explained in terms of traceability in Chapter 3.7.

3.1 Requirements Engineering

While this chapter discusses RE in general, the RE related concepts presented in the RE4SA model are clarified in Chapter 3.4.

RE is concerned with the analysis, elicitation, validation and management of requirements, among other activities. RE can be defined as follows: "*Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families*" (Zave, 1997). In short RE handles requirements, in which a requirement can be described as "*a property that must be exhibited by something in order to solve some problem in the real world*" (Bourque & Fairley, 2014). The elicitation of requirements has little to do with architecture, as it is mostly concerned with identifying stakeholders, gathering information and setting goals for the system. The next step of the RE process, modeling and analyzing requirements is not of much use either. The main architectural design challenge is in communicating and evolving requirements (Nuseibeh & Easterbrook, 2000). Both activities focus heavily on change management. Change management should not be considered an afterthought, software development is dynamic, meaning that requirements can change and evolve while development is not yet finished (Nurmuliani, Zowghi, & Powell, 2004).

Changing requirements are also referred to as volatile requirements. Such requirements volatility is generally perceived as unwanted and is considered a major problem in the software industry (Curtis, Krasner, & Iscoe, 1988). Reportedly, volatile requirements negatively a software project, resulting in projects not being delivered on time and exceeding budgets (Zowghi & Nurmuliani, 2002). Requirements volatility can be defined as: "*the tendency of requirements to change over time in response to the evolving needs of customers, stakeholders, organisation, and work environment*". In RE, two types of requirements can be distinguished, namely functional and nonfunctional requirements. The former refer to certain functions a system should offer, these are often called features or capabilities. The latter refer to requirements that express constraints for the solution. Therefore, they are sometimes referred to as constraints or quality requirements (Bourque & Fairley, 2014). Nonfunctional requirements are often not located or addressed in a single part of the architecture, but rather in multiple parts or even the product or system as a whole. Since these requirements cannot be directly related to concepts in the SA, only functional requirements are considered throughout the case studies. The concepts of Epics and USs are described in Chapter 3.4.

3.2 Software Architecture

In this chapter, the SA dimension of the RE4SA model is described in more detail. First, SA is examined in general, after which the SA related concepts in the RE4SA model are discussed (modules and features).

According to Rozanski and Woods, "*the architecture of a system is the set of fundamental concepts or properties of the system in its environment, embodied in its elements, relationships, and the principles of its design and evolution*". Since this research is concerned with the implementation and realization of requirements, only the functional viewpoint of a SA is examined. This viewpoint handles the runtime functional elements of a system, their responsibilities, primary interactions and interfaces. Essentially, it describes how a system performs its necessary functions (Rozanski & Woods, 2011). As has been explained previously, software architects are required to understand

evolving requirements and their impact on the architectural design of a system. Khan, Greenwood, Garcia and Rashid distinguish six types of dependencies in architectures (Khan et al., 2008):

1. Goal: related to requirements specifying development and quality of service.
2. Service: related to requirements describing characteristics of the system that correspond to operations and functions in the architecture.
3. Conditional: related to requirements that describe events that trigger services, processes and tasks, which are realized in the architecture.
4. Temporal: related to requirements that are considered time-sensitive in the architecture.
5. Task: related to the connections between artifacts which require user input.
6. Infrastructure: related to resources, infrastructures, technical standards and compatibility realized in the architecture.

Even though architectural models can describe a system's structure, it is not in full agreement with the source code. This discrepancy is also referred to as the 'model-code gap' by Fairbanks. He states that the architecture is an abstraction and therefore includes elements that are not present in the code, design decisions and constraints for example. He continues by saying you can either avoid or manage it. The former, however, prevents you from sufficiently dealing with scalability and complexity, so the latter option might be more appropriate. In shot managing the gap means managing the consistency between source code and architecture. Fairbanks takes a risk-driven approach to architecture. This means that he proposes doing "just enough" architecture, since it can consume a lot of time and effort. Similarly, he describes a 'constant sync' of architecture and source code as 'expensive and uncommon' (Fairbanks, 2010). Therefore, it could be valuable to devise an approach to automatically updating the SA documentations using source code or vice versa. In addition, RE documentation could also be used as a means for updating the SA documentation and, by extension, the source code.

3.2.1 Functional Architecture Modeling

In the RE4SA model, the SA domain contains two concepts, namely modules and features. While features can be modeled by means of feature diagrams, modules are often not included in such visualizations. The utrecht Architecture Description Language (uADL), however, does provide an approach for modeling modules (as well as the related features). The functional viewpoint in the uADL, based on the work by Rozanski and Woods, utilizes Functional Architecture Models (FAMs), feature diagrams and scenario overlays (Jansen & van Rhijn, 2018). The FAM as used in the uADL was developed by Brinkkemper and Pachidi, for the purpose of modeling functionality in relation to the satisfaction of requirements (Brinkkemper & Pachidi, 2010). Given this relationship between FAMs and requirements, the modeling technique suits the aim of this research. In figure 5 a nonspecific example of such a FAM is provided.

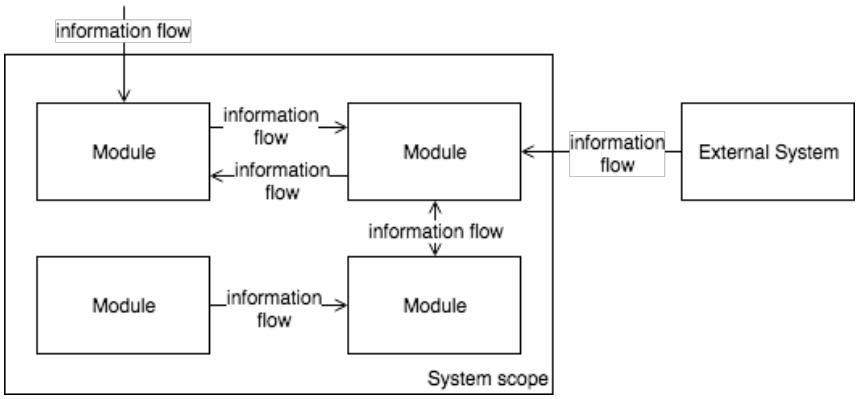

Figure 5: Example of a Functional Architecture Model (FAM).

In this model, the scope can be defined by the product or system that is being described. The system consists of modules, information flows and, optionally, external systems. Brinkkemper and

Pachidi distinguish two types of modules, namely modules that contain sub-modules and stand-alone modules. Modules can consist of sub-modules (which are also modeled in the functional architecture) or features, which are modeled using feature diagram, as described in Chapter 3.2.2. In addition, there are systems outside of the scope that the system to be modeled interacts with. Interactions, or rather communication, between modules and external systems are visualized using arrows called information flows. Information flows can either be one-way or bidirectional. Brinkkemper and Pachidi also propose the use of Functional Architecture Diagrams (FADs). FADs are used to separate different layers of the functional architecture. A nonspecific example of a FAD is presented in figure 6. FADs can be created for all modules that contain sub-modules.

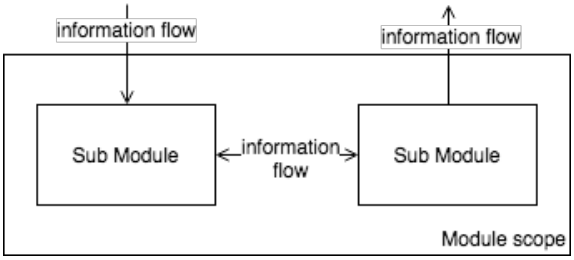

Figure 6: Example of a Functional Architecture Diagram (FAD).

The main difference with FAMs is that sub-modules are included in these diagrams. Furthermore, the inbound and outbound origins and destinations of information flows of the module are not included, since they are not part of the module scope. Other than that, the types of flows that can be used are the same.

To better illustrate how FAMs can be used, consider the context-specific example (created for an imaginary maps application, akin to Google Maps in functionality) shown in figure 7.

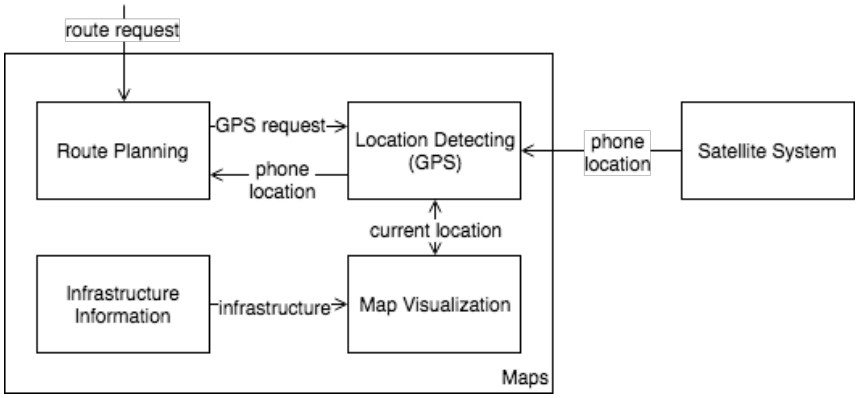

Figure 7: Context-specific example of a FAM of a maps application.

The maps application, aptly called 'Maps' (as shown in the scope in the bottom right), consists of four modules, one of which contains sub-modules ('Route Planning'). For this module it is possible to create a FAD. The application also interacts with an external system. Finally, scenarios can be written to describe the use of the application. These scenarios can be illustrated by using a scenario overlay on the FAM, which essentially means showing which modules are used in which order by putting arrows on top of the information flows in the FAM.

To provide an indication of what a FAD could look like, the scope of the 'Route Planning' module from the previous figure is illustrated in figure 8.

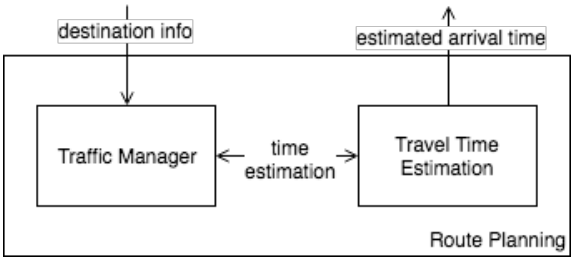

Figure 8: Context-specific example of a FAD of a maps application.

An example of which sub-modules can be contained in the 'Route Planning' module are 'Traffic Manager' and 'Travel Time Estimation', the purposes of which are quite self-explanatory. Naturally, the 'Route Planning' module could (and probably should) contain more sub-modules, but they are not included in this non-exhaustive example.

3.2.2 Feature Diagrams

In addition to FAMs and FADs, feature diagrams are also used to illustrate the SA, an example of a feature diagram is presented in figure 9.

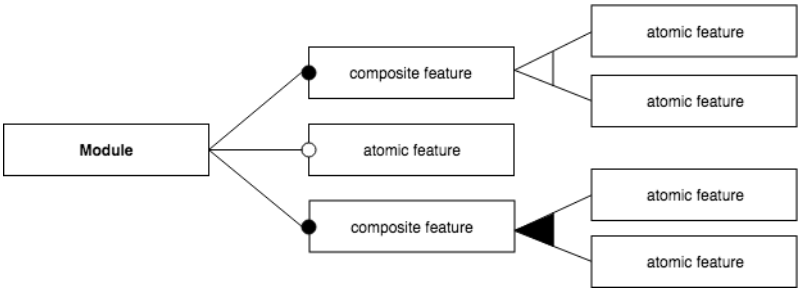

Figure 9: Example of a feature diagram.

A feature diagram, in this format also referred to as a feature tree, since it shows which features are contained within a module and which features are part of other features. A filled in circle means that the feature is mandatory, while a white circle indicates an optional feature. In addition, some features can consist of other features, in which a black decomposition indicates an 'OR' and the white decomposition an alternative (Cechticky, Pasetti, Rohlik, & Schaufelberger, 2004). As figure 9 has shown, composite features also exist. The distinction between modules, sub-modules, features and sub-features may seem arbitrary, given that their formulation and description can be slightly altered to turn move them up or down the hierarchy in terms of abstraction. This granularity issue also exists in RE and is discussed further in Chapter 3.4.4.

Using the maps application as an example once again, figure 10 below illustrates what a feature diagram for the 'Route Planning' module could look like. For completeness, all other modules and their features should also be included in a feature diagram, but they have been omitted in this example.

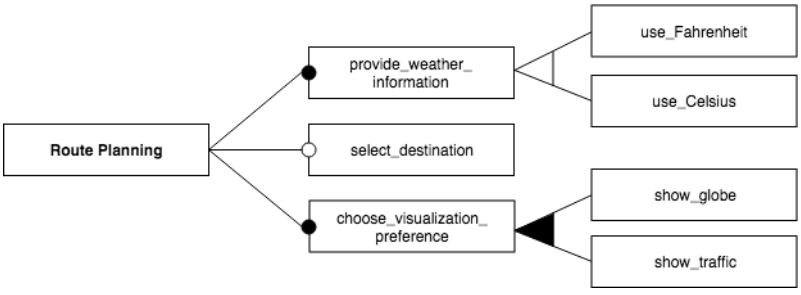

Figure 10: Context-specific example of a feature diagram of a maps application.

The black circles indicate mandatory features, while providing information about the weather is considered optional. In addition, two of the three features are composite and can be specified further. For the visualization preference, it is possible to select a globe view and to show traffic, one of the two or neither. In regards to the weather information you can see the temperature in Celsius or, alternatively, in Fahrenheit, but they essentially provide the same functionality, namely showing the temperature.

3.2.3 Software Architecture Recovery

Creating a model from an existing system is often referred to as reverse engineering. Chikofksy and Cross define the term as: "*the process of analyzing a subject system to identify the system's components and their inter-relationships and create representations of the system in another form or at a higher level of abstraction*" (Chikofsky & Cross, 1990). The purpose of reverse engineering is therefore to examine a system as-is and not necessarily to create a perfect or complete model (Canfora, Di Penta, & Cerulo, 2011). Architecture Recovery (AR) is defined as "*when the implemented architecture is extracted from the implemented system*" (Ali, Baker, O'Crowley, Herold, & Buckley, 2018). AR can be divided into two approaches, namely top-down and bottom-up, called 'discovery' and 'recovery' respectively. Discovery using high-level knowledge as input, such as requirements. However, since in this research the goal is to map the architecture to the requirements based on granularity and linguistics, this is an undesirable approach. The recovered architecture would be heavily influenced by the RE artifacts, negatively affecting the validity of the results. Therefore, the recovery approach is used instead. Recovery utilizes source code and models that represent it to create a high-level visualization of the architecture (Ali et al., 2018). In contrast with discovery, recovery solely requires the use of SA artifacts, meaning that it will not impact the validity due to (linguistic) bias.

The potential need for SA artifact recovery was already alluded to in Chapter 2, this, however, should be done carefully. As described earlier, Fairbanks has identified a gap between architectural designs and code by stating that the former is abstract and high-level and the latter concrete and low-level (Fairbanks, 2010). This leads to the conclusion that it is perhaps not possible (or maybe desirable) to map them one-on-one. Software reflexion models were developed to bridge this gap. In this approach, high-level architectural models (depicting modules and calls) is mapped to the source code. Then, a software reflexion model is generated by a tool that illustrates the similarities and conflicts between the high-level model and the software reflexion model based on the code (Murphy, Notkin, & Sullivan, 1995). This approach, however, seems to focus on the calls between modules, which can be compared to the information flows between modules in a FAM or FAD (refer to Chapter 3.2.1). Information flows are not explicitly included in the RE4SA model and therefore not necessarily of importance when analyzing linguistic structures, making the model-code gap less of an issue.

3.3 Features

The concept 'feature' in relation to software has been defined in various ways over the last three decades. Arguably one of the first definitions was presented in the early 90s in a technical report on feature-oriented domain analysis (Kang, Cohen, Hess, Novak, & Peterson, 1990). This definition seems to be adapted from the American Heritage dictionary entry for feature. The report, in terms of citations, was quite popular. However, ever since, feature definitions have started to deviate. In an attempt to properly define the concept and categorize existing definitions, a small-scale SLR was conducted, following the guide by Okoli and Schabram (Okoli & Schabram, 2010). This guide specifies the following eight steps for executing an SLR:

1. Purpose of the literature review
2. Protocol and training
3. Searching for the literature
4. Practical screen
5. Quality appraisal
6. Data extraction
7. Synthesis of studies
8. Writing the review

Since the goal of this SLR is to provide definitions from various perspectives and for multiple purposes, it is not the intent to create an exhaustive list of all feature definitions. instead, the aim is to collect definitions from different perspectives and analyze them. This analysis, which is a categorization of the definitions, is the main contribution. Furthermore, it should be noted that the second step is superfluous in this case, since the search, screening and analysis were all performed by the same person, so the risk of alignment issues is practically non-existent.

Results were found by using operator searching on Google Scholar. Since all definitions should be specifically for the term 'feature', this word is always included in the search query. Since that term in combination with the word 'definition' often lead to results not related to information and/or computing science, more specific queries were used instead. The second term in the search query is based on other topics relevant to this thesis, starting with RE and SA. The objective of this SLR is, after all, to provide definitions for various purposes, which is why multiple research fields are included in the search. Table 1 provides an overview of the search queries used and the results that were included for each query.

Search query	**Included results**
"feature" AND "requirements engineering"	Classen et al., 2008; Kang et al., 1990
"feature" AND "software architecture"	Apel & Kästner 2009; Kang et al., 1990
"feature" AND "product lines"	Apel et al., 2013
"feature" AND "software system"	Apel et al., 2013; Apel & Kästner 2009
"feature" AND "feature-oriented specification"	Guerra et al., 1996; Apel & Kästner, 2009
"feature" AND "source code"	Dit et al., 2013

Table 1: Overview of utilized search queries (on Google Scholar) and their included results.

The results are presented in order of the search results (so relevance in relation to the search query). It should be noted that the work by Apel et al. published in 2013, is stated as a work from 2016 by Google Scholar. However, the book itself includes a copyright text from 2013 and the foreword was also dated 2013. More results were presented, but in the table only results included as per the selection criteria (described later) are shown.

In addition to search queries, the snowballing technique was also utilized. In this case this consisted of backwards searching. Two articles were selected for this technique, since these two works explicitly cited various definitions of the term 'feature'. Table 2 summarizes which works have been found in each article.

Source	**References**
Classen et al., 2008	Kang et al., 1990; Kang et al., 1998; Bosch, 2000; Czarnecki & Eisenecker, 2000; Batory, 2004; Batory et al., 2004; Pohl et al., 2005; Batory et al., 2006; Apel et al., 2007
Apel & Kästner, 2009	Kang et al., 1990; Kang et al., 1998; Bosch, 2000; Czarnecki & Eisenecker, 2000; Zave, 2003; Batory et al., 2004; Chen et al., 2005; Czarnecki et al., 2005; Pohl et al., 2005; Batory et al., 2006; Apel et al., 2007; Classen et al., 2008; Kästner et al., 2008

Table 2: Works found through the use of the snowballing technique.

The references are shown in chronological order and, if two or more works were published in the same year, alphabetical order. Overlapping references between the two works are included for both for completeness. Based on correspondence with Sven Apel, three works co-written by Thorsten Berger were included as well (S. Apel, personal communication, February 12, 2019). More than one definition written by Apel has been included and the article providing an overview of feature-oriented development was used not only as a starting point for searching for more definitions, but also inspired the synthesis in part. Therefore, the recommendation was gladly accepted. Moreover, these three works were published more recently than most of the other included works, providing a nice overview of the term 'feature' over the past thirty years.

To scale down the number of results and to assure quality of the results, some results had to be excluded. The definition and the works in they were provided should meet the following selection criteria:

1. Works must be written in English.
2. Works must have been published in journals, workshops, conferences, books and the like.

3. Articles and books only, other media, such as blogs, videos and slides, are not included.
4. Works must present a unique definition of the concept feature.

At first, a minimal number of citations requirement was used. However, since the works used in the SLR range from 1990 to 2018, this as deemed an 'unfair' criterion, since older works have had more time to get cited. The fourth and final criterion refers to the fact that works that cite another definition are not included here. Moreover, the aim is to provide an overview of existing definition, so 'unpopular' definitions should be featured as well for completeness. After applying the selection criteria, 22 works remained from the original set of 23. Table 3 shows the feature definitions in chronological order. The references in the table are written using short citations, due to the size and thus legibility of the table.

Authors	**Year of publication**	**Definition**
Kang, Cohen, Hess, Novak & Peterson (Kang et al., 1990)	1990	*"a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems"*
Guerra, Ryan & Sernadas (Guerra et al., 1996)	1996	*"is a part or aspect of a specification which a user perceives as having a self-contained functional role"*
Kang, Kim, Lee, Kim, Shin & Huh (Kang et al., 1998)	1998	*"distinctively identifiable functional abstractions that must be implemented, tested, delivered, and maintained"*
Bosch (Bosch, 2000)	2000	*"a logical unit of behaviour specified by a set of functional and non-functional requirements"*
Czarnecki & Eisenecker (Czarnecki & Eisenecker, 2000)	2000	*"a distinguishable characteristic of a concept (e.g., system, component, and so on) that is relevant to some stakeholder of the concept"*
Zave (Zave, 2003)	2003	*"an optional or incremental unit of functionality"*
Batory (Batory, 2004)	2004	*"the primary units of software modularity"*
Batory, Sarvela & Rauschmayer (Batory et al., 2004)	2004	*"a product characteristic that is used in distinguishing programs within a family of related programs"*
Chen, Zhang, Zhao & Mei (K. Chen et al., 2005)	2005	*"a product characteristic from user or customer views, which essentially consists of a cohesive set of individual requirements"*
Czarnecki, Helsen & Eisenecker (Czarnecki et al., 2005)	2005	*"a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate among systems in a family"*
Pohl, Böckle & van der Linden (Pohl et al., 2005)	2005	*"an end-user visible characteristic of a system"*
Batory, Benavides & Ruiz-Cortes (Batory et al., 2006)	2006	*"an increment in product functionality"*
Apel, Lengauer, Batory, Möller & Kästner (Apel et al., 2007)	2007	*"a structure that extends and modifies the structure of a given program in order to satisfy a stakeholder's requirement, to implement and encapsulate a design decision, and to offer a configuration option."*
Classen, Heymans & Schobbens (Classen et al., 2008)	2008	*"a triplet, f = (R,W,S), where R represents the requirements the feature satisfies, W the assumptions the feature takes about its environment and S its specification"*
Kästner, Apel & Kuhlemann (Kästner et al., 2008)	2008	*"represents an increment in functionality relevant to stakeholders"*
Apel & Kästner (Apel & Kästner, 2009)	2009	*"is a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option"*
Apel, Batory, Kästner & Saake (Apel et al., 2013)	2013	*"is a characteristic or end-user-visible behavior of a software system"*
Dit, Revelle, Gethers & Poshyvanyk (Dit et al., 2013)	2013	*"represents a functionality that is defined by requirements and accessible to developers and users"*
Berger, Lettner, Rubin, Grünbacher, Silva, Becker, Chechik & Czarnecki (Berger et al., 2015)	2015	*"describe the functional and non-functional characteristics of a system"*
Andam, Burger, Berger & Chaudron (Andam et al., 2017)	2017	*"are high-level, domain-specific abstractions over implementation artifacts"*
Krüger, Gu, Shen, Mukelabai, Hebig & Berger (Krüger et al., 2018)	2018	*"used to specify, manage, and communicate the behavior of software systems and to support developers in comprehending, reusing, or changing these systems"*
Rodríguez, Mendes & Turhan (Rodriíguez et al., 2018)	2018	*"represent needs that are gathered via meetings with customers or other stakeholders, which, once selected, are refined during a requirements elicitation process"*

Table 3: Overview of 22 definitions of the concept 'feature', presented in chronological order.

As is evident from the list in the table, many definitions of the term 'feature' exist. To complicate matters further, there is no one true definition and, at first glance, one definition is not necessarily better than another. Therefore, the goal is not to identify the 'perfect' definition, but to provide multiple options given various context. Therefore, five different categorizations of the 22 definitions are provided in this chapter:

1. Categorization by research topic
2. Categorization by number of citations on Google Scholar
3. Categorization by h-index of authors

4. Categorization by level of abstraction and viewpoint
5. Categorization by relationship based on cites and cited by

Firstly, it is possible to identify various research fields/topics. This division based on research topic is provided in table 4.

Research topic	**Related works**
Feature-oriented software	Kang et al., 1990, 1998; Batory, 2004; Apel et al., 2007; Apel & Kästner, 2009; Dit et al., 2013
Feature-oriented specifications	Guerra et al., 1996; Zave, 2003
Generative programming	Czarnecki & Eisenecker, 2000
Software product lines	Bosch, 2000; Batory et al., 2004; Pohl et al., 2005; Kästner et al., 2008; Apel et al., 2013; Berger et al., 2015; Andam et al., 2017; Krüger et al., 2018
Feature modeling	Chen et al., 2005; Czarnecki et al., 2005; Batory et al., 2006
Requirements engineering	Classen et al., 2008
Release planning	Rodríguez et al., 2018

Table 4: Feature definitions categorized by research topic.

The research topics were determined based on which topics or fields were mentioned in the abstract, keywords or introduction. In addition, the research fields or topics related to the journal or conference proceedings the work was published in were also taken into account. Some overlap between the topics is possible, since some works included a more specific topic or field than others. For example, in table 4, feature-oriented software may also be interpreted as feature-oriented programming in some cases, but to keep it more generic, the former topic description has been used instead. In addition, it is possible that a definition could fit more than one research topic, in such cases the most important or prominent one was selected. For instance, the definition by Rodríquez et al., could also fit the RE topic, but was categorized as release planning, since this was the main topic of the work. Analyses of the evolution of a definition have not been considered viable. While some authors have published multiple definitions on the same research topic, they were in all cases written by a different research team. Therefore, the influence of the new/different researchers could be great. Analyses of evolutions would thus be heavily based on assumptions.

Another approach could be to use the definitions presented in the most frequently cited works. In that case, figure 11 could be used to select a definition. However, this does not take research topics into account. Moreover, more recently published works have had less time to get cited, resulting in a slightly skewed view.

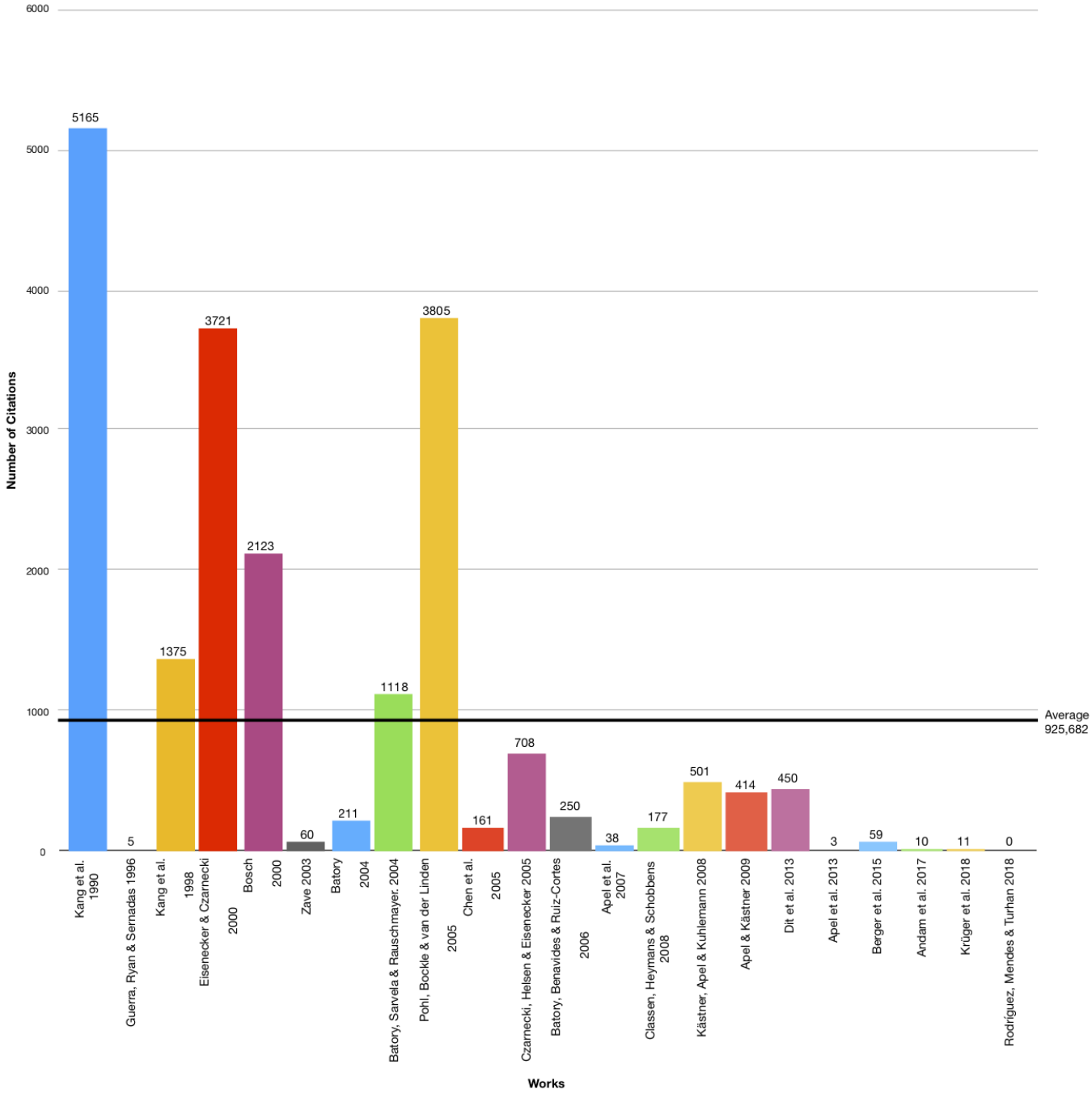

Figure 11: Overview of number of citations on Google Scholar of works in which definitions are provided (as of November 2018).

One approach for selecting a feature definition could be to consider all works that score above average on the number of citations and use that one, or use the definition by Kang et al. from 1990, since it has the most citations.

In addition to the number of citations on Google Scholar, the h-index of authors as presented by Google Scholar can also be considered. Table 5 provides an overview of the h-index scores for all the authors that have been included in the feature definition overview (table 3), according to Google Scholar (scores as of January 2019). These scores can give an indication of the prestige or renown of an author, arguing their trustworthiness.

Work	**Author(s)**	**h-index**
Kang et al., 1990	Kang, *Cohen*, *Hess*, *Novak*, *Peterson*	29, *9*, *7*, *8*, *38*
Guerra et al., 1996	*Guerra*, Ryan, *Sernadas*	*10*, 39, *34*
Kang et al., 1998	Kang, Kim, Lee, Kim, *Shin*, *Huh*	29, 21, 21, 125, *1*, *1**
Czarnecki & Eisenecker, 2000	Czarnecki, *Eisenecker*	56, *10*
Bosch, 2000	Bosch	56
Zave, 2003	Zave	32
Batory, 2004	Batory	60
Batory et al., 2004	Batory, *Sarvela*, *Rauschmayer*	60, *4*, *10*
Pohl et al., 2005	Pohl, *Böckle*, *van der Linden*	49, *10*, *22*
Chen et al., 2005	*Chen*, *Zhang*, Zhao, *Mei*	*51*, *51*, 15, *51*
Czarnecki et al., 2005	Czarnecki, *Helsen*, Eisenecker	56, *12*, 10
Batory et al., 2006	Batory, Benavides, Ruiz Cortés	60, 30, 39
Apel et al., 2007	Apel, Lengauer, Batory, *Möller*, Kästner	54, 34, 60, *29*, 53
Classen et al., 2008	Classen, Heymans, Schobbens	23, 43, 32
Kästner et al., 2008	Kästner, Apel, *Kuhlemann*	53, 54, *18*
Apel & Kästner, 2009	Apel, Kästner	54, 53
Dit et al., 2013	Dit, *Revelle*, *Gethers*, Poshyvanyk	15, *11*, *21*, 54
Apel et al., 2013	Apel, Batory, Kästner, Saake	54, 60, 53, 46
Berger et al., 2015	Berger, Lettner, Rubin, Grünbacher, *Silva*, *Becker*, Chechik, Czarnecki	24, 11, 18, 40, *5*, *51*, 40, 56
Andam et al., 2017	*Andam*, Burger, Berger, Chaudron	*1*, 5, 24, 31
Krüger et al., 2018	Krüger, *Gu*, *Shen*, Mukelabai, Hebig, Berger	7, *1*, *51*, 2, 12, 24
Rodríguez et al., 2018	Rodríguez, Mendes, Turhan	12, 45, 27

Table 5: Overview of h-index scores of authors (as of January 2019) included in the feature definition table.

The short citations of the works have been included to improve consistency between the various tables and figures. The h-index scores, if multiple are stated, are presented in the same order as the authors are named in the 'Author(s)' column. For some authors their h-index was not available on Google Scholar. Instead, a Google Chrome extension (the Scholar H-Index Calculator for Google Chrome™, accessible here: `https://chrome.google.com/webstore/detail/scholar-h-index-calculato/cdpobfbhbdlpbloccjokjgekjnmifbng`) was used to calculate the h-index for missing authors (included in italics in table 5). The h-index without any normalization was used, since on Google Scholar the overall h-index was used. However, for some authors, their h-index could not be precisely determined. Whenever the h-index exceeded fifty, only "> 50" was shown and since the exact h-index cannot be accurately determined, for all authors exceeding fifty, 51 was included in the table. Similarly, for one author the h-index was "> 1" including normalized calculations. This score is indicated by an asterisk.

Using these data, figure 12 shows the average h-index of all authors of which the h-index was available.

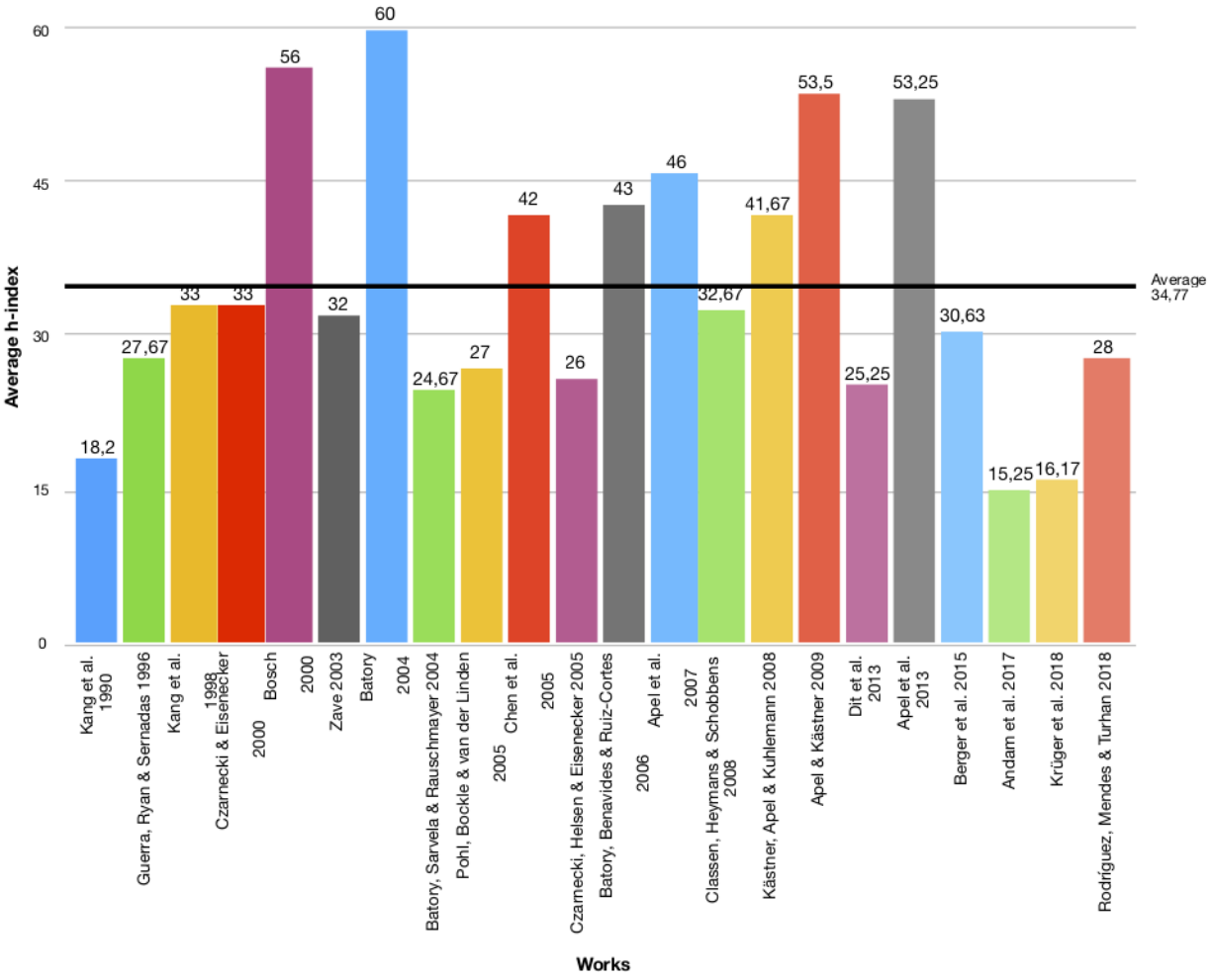

Figure 12: Average h-index of authors per work.

This figure represents the scientific value of the authors, so to speak, so this overview can also be used to select a definition. Again, by selecting one from the authors that score above average or by choosing from the four with the highest score.

To provide a slightly different perspective, figure 13 depicts the h-index scores of the first author.

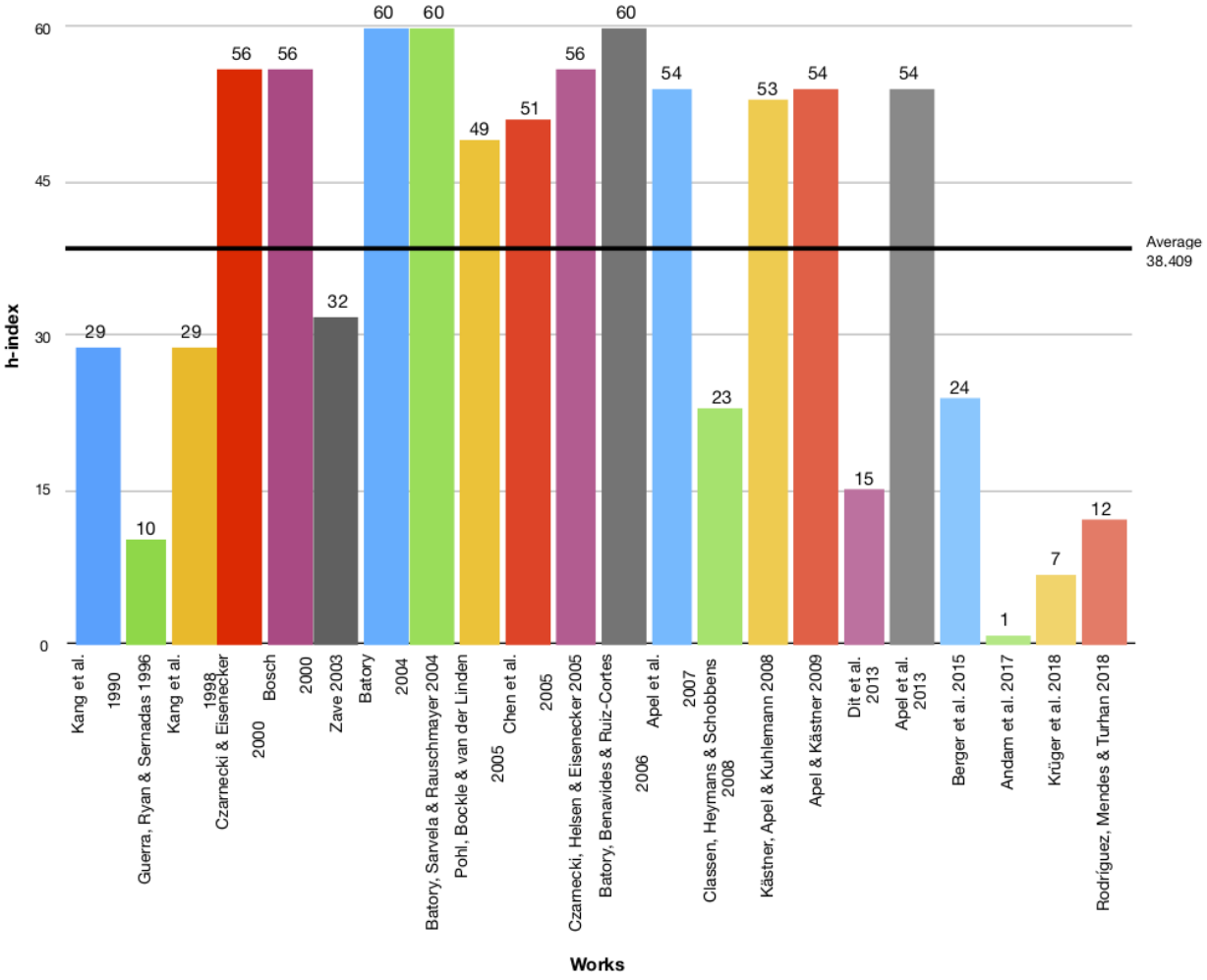

Figure 13: h-index of first author with available data.

As is apparent, this figure provides a slightly different impression of the authors. Similarly to the previous categorizations, it could be advisable to select a definition by an author or authors that score above average on the h-index. The top three, however, is dominated by Batory, so choosing between these three could result in some bias.

Based on the categorization presented earlier, it is possible to reduce the number of relevant definitions for a certain research topic. For research topics that are not included in the table and topics that have multiple definitions to choose from, the issue is still not resolved. However, some definitions might be more suitable than others given a certain context or intention. In all definitions two aspects can be distinguished, namely: level of abstraction and viewpoint. The former aspect was inspired by the differentiation between abstract and technical feature definitions as proposed by Apel and Kästner (Apel & Kästner, 2009). They also recognize that features have more than one use and describe the differentiation as follows:

1. Abstract: "*features are abstract concepts of the target domain, used to specify and distinguish software systems*" (problem space)
2. Technical: "*features must be implemented in order to satisfy requirements*" (solution space)

Czarnecki and Eisenecker have separated the problem space from the solution space, in which the former focuses on domain-specific abstractions and the latter in implementation-oriented abstractions (Czarnecki & Eisenecker, 2000). Apel and Kästner use this distinction to further define abstract and technical, relating abstract definitions to the problem space and technical definitions to the solution space (Apel & Kästner, 2009). In addition to this distinction between abstract and technical definitions, they have provided a list of ten definitions (all of which are also included

in table 3) and ordered them from abstract to technical. However, they have not described how they decided on which definition is more technical than another. Moreover, they identified seven abstract definitions and only three technical ones. In short, while the line between abstract and technical is clear, the gap between the two is not and the reasoning behind the order within both distinctions is vague at best. To clarify the interpretations of abstract and technical, prof. dr. Apel was asked to comment on the paper. He stated that the first seven definitions "take a user-centric/problem-space-centric perspective", while the eighth definitions is only formal from an RE perspective. The last two definitions focus on the implementation and are thus solution-space-specific. He continues by saying that, within these categorizations, the definitions are more or less sorted by date (S. Apel, personal communication, February 12, 2019). To conclude, this approach was quite informal and therefore difficult to replicate. Moreover, it still does not solve the mystery of which definition is more abstract or technical than another. A more formal categorization is needed to tackle these challenges.

In an attempt to recreate such an order based on level of abstraction (from abstract to technical), nine characteristics were extracted from the collection of 19 definitions (the other three were added later). Four of these characteristics were labelled abstract (characteristic, distinct (or variations thereof), aspect and abstraction) and five were considered technical (specification, functionality (or variations thereof), requirements, behavior and unit). Present abstract characteristics receive a 1, present technical characteristics receive a 0, divide this by the number of characteristics and the resulting scores are somewhere between 0 and 1 (0 and 1 included). In this case, 1 is the most abstract and 0 the most technical (or least abstract). A test comparing the order based on these nine characteristics and resulting score and the order of seven definitions as presented by Apel and Kästner resulted in the following findings:

1. Six out of seven definitions received a different position in the order.
2. Two definitions had a shift of three spaces.
3. If the line between abstract and technical is placed at 0.5, one definition shifts from abstract to technical and one is shifted the other way around.

In other words, this approach is not satisfactory. A second attempt, taking a different approach yielded better results. After analyzing the different characteristics of abstract and technical as stated by Czarnecki and Eisenecker and Apel and Kästner, as discussed earlier in this chapter, the following eight characteristics were selected:

1. Abstract: problem space, description of requirements, description of intended behavior and characteristic/abstract/abstraction
2. Technical: solution space, satisfaction of requirements, implementation of intended behavior and functionality

Using this approach, with the same method for calculating a score, the 19 definitions and an additional three (due to new definition suggestions) were ordered once again. So, 22 definitions were ordered, with the following results:

1. If the line between abstract and technical is placed at 0.5, none of the definitions shift from abstract to technical or vice versa.
2. The three technical definitions are in the same order.
3. Out of the seven abstract definitions only two are out of order (and the order among those two is the same as in the order by Apel and Kästner).

To summarize, out of the ten definitions, only two were out of order (and disregarding the other definitions, those two were in the correct order). Another advantage of this approach is that it is not based on terms/characteristics extracted from the definition, but on theoretical resources by Czarnecki and Eisenecker and Apel and Kästner. Furthermore, it should not be forgotten that it is unclear whether the original order as devised by Apel and Kästner is on an ordinal scale. It is reasonable to assume so, since the definitions are numbered. However, the reasoning behind this specific order is not thoroughly explained, apart from the descriptions of abstract and technical as stated previously. The one fully unambiguous aspect is the distinction between the abstract and technical definitions, since this was explicitly mentioned.

In addition to the level of abstraction, five viewpoints were also extracted from the 22 definitions:

1. System

2. Product
3. Developer (stakeholder)
4. User (stakeholder)
5. Customer (stakeholder)

Firstly, system and product are considered separate viewpoints, since a system can be contained within a product, but a product can indicate more than just a system. Secondly, three stakeholders were identified and only human beings are considered a stakeholder. The developer was included, not because it was explicitly mentioned in any of the definitions, but sometimes the word stakeholder also refers to the development viewpoint. Thirdly, the user viewpoint also includes end-users and differs from the developer viewpoint, since developer do not necessarily use the product or system, but other employees of the product's or system's company might. Fourthly, customers are separated from user, since they are more specific than just any (end-) user. Finally, whenever no specific viewpoint is mentioned or can be reasonably assumed given a definition, the system is considered the viewpoint, due to features being part of the SA, which describes a system. Figure 14 shows the categorization of the definitions based on the level of abstraction score (as described previously) and the identified viewpoints.

Viewpoint

Level of Abstraction: Abstract ↑ Technical

Definition	System	Product	Developer (stakeholder)	User (stakeholder)	Customer (stakeholder)
Kang et al. (1990)	✓			✓	
Bosch (2000)	✓				
Pohl et al. (2005)	✓			✓	
Chen et al. (2005)		✓		✓	✓
Guerra et al. (1996)				✓	
Kang et al. (1998)	✓				
Czarnecki & Eisenecker (2000)	✓		✓	✓	✓
Batory et al. (2004)		✓			
Czarnecki et al., 2005			✓	✓	✓
Apel et al. (2013)	✓			✓	
Rodríguez et al. (2018)			✓	✓	✓
Classen, et al. (2008)	✓				
Zave (2003)	✓				
Batory (2004)	✓				
Batory et al. (2006)		✓			
Kästner et al. (2008)			✓	✓	✓
Dit et al. (2013)			✓	✓	
Apel et al. (2007)	✓		✓	✓	✓
Apel & Kästner (2009)	✓				

abstract - technical split

Figure 14: Categorization of nineteen feature definitions, based on level of abstraction and viewpoint.

The nineteen definitions and the scoring system were also presented to peers and information science researchers. Both groups expressed a difficulty in understanding what the term 'technical' was supposed to mean in this context. Especially given their background, since they automatically assumed technical characteristics to be related to development aspects or implementations (such as code). Moreover, throughout this thesis, the level of abstraction is often seen as the level of granularity, while in this categorization that is not the case. To make the categorization easier to read and understand, a different name and more specific minimal and maximum values would be desirable. Changing 'technical' to 'detailed' might solve the issue of misinterpreting technical characteristics, but would be an inaccurate description. The definitions do not necessarily refer to a certain level of detail and abstract definitions can still provide a detailed description of the term feature. Garlan, a computer scientist, explains the role of SA in RE. In his explanation, he refers to RE as being concerned with the 'shape of the problem space', while SA focuses on the 'shape of the solution space' (Shekaran et al., 1994). The distinction between problem and solution space is already present in the categorization, given the fact that the description of the problem space is considered an abstract characteristic and, on the other hand, the solution space is considered a technical characteristic (Apel & Kästner, 2009). To strengthen this reasoning, the QUS framework is in agreement stating that a US should be problem-oriented, meaning that "*a user story only specifies the problem, not the solution to it*" (Lucassen, Dalpiaz, van der Werf, & Brinkkemper, 2016a). Moreover, Hofmeister et al. mention that architecture solutions help move the design from the problem space (in which architecturally significant requirements (ASRs) are formulated) to the solution space (Hofmeister et al., 2007). Splitting the definitions into two main categories can make selecting a definition easier, depending on the purpose for and context in which it is used. However, problem-space definitions (RE) can arguably be considered of higher quality or more useful, based on research by Berger et al. They state that "*good features need to precisely describe customer-relevant functionality*" (Berger et al., 2015). Moreover, this would mean that definitions which include the customer viewpoints are more suitable than those that do not.

The previous categorization or analysis of the definitions heavily relied on interpretation. To provide a different perspective, the definitions were also analyzed objectively by determining the term frequency. Only semantically interesting terms were examined, meaning nouns, verbs and adjectives. Terms like determiners, adverbs and modular verbs are mostly used to construct a legible sentence. The PoS tags were assigned using the CoreNLP tool. Table 6 shows how often a term could be found in the definitions. Note that this does not equal the total number of times the term is present in all definitions.

NN/NNS	Frequency	VX	Frequency	JJ	Frequency
system*	8	represent	5	functional	3
characteristic	7	implement	2	visible	3
requirement*	7	specify	2	relevant	3
software	5			non-functional	2
functionality	5				
stakeholder	4				
unit	4				
user*	3				
behavior	3				
product	3				
specification	2				
aspect	2				
abstraction	2				
set	2				
program*	2				
family	2				
design	2				
decision	2				
configuration	2				
option	2				
increment	2				
developers	2				
customer*	2				

Table 6: Observed term frequency in all definitions.

Words followed by an asterisk indicate that the singular and plural forms have been included in the frequency, if no asterisk is present, the word was observed exactly as it is written in the table. For the verbs, any kind of verb was included (hence the 'VX'), except for modular verbs. In addition, verbs that are not context-specific were excluded, e.g. 'is' and 'have' among others. 'Visible' is included by itself, but can sometimes be found in conjunction with other words as a more specific adjective (for example 'user-visible'). Only terms that could be observed more than once are shown in the table. Furthermore, both 'design decision' and 'configuration option' can be considered compound nouns, as these terms could only be observed in combination. The term 'user' was observed three times, but if the observation included the adjectives as well (e.g. 'end-user-visible'), it could be found six times.

Furthermore, it would be interesting to see how certain definitions were developed and whether they were inspired by any of the other definitions included in the SLR. The hermeneutics framework as proposed by Cole and Avison visualizes how one goes from understanding a phenomenon to explaining it and finally to interpreting it. The final step in this framework, part of interpretation, is 'fusion of horizons', meaning that someone has gathered shared meanings and has created new concepts (or in this case variations thereof) that differ from the original concepts and their meanings (Cole & Avison, 2007). In an attempt to discern how the definitions were formulated, the works that were cited have been looked into. Figure 15 shows who was cited by whom, possibly indicating inspiration for the definitions.

	Kang et al., 1990	Guerra et al., 1996	Kang et al., 1998	Czarnecki & Eisenecker, 2000	Bosch, 2000	Zave, 2003	Batory, 2004	Batory et al, 2004	Pohl et al, 2005	Chen et al, 2005	Czarnecki et al, 2005	Batory et al, 2006	Apel et al, 2007	Classen et al, 2008	Kästner et al, 2008	Apel & Kästner, 2009	Dit et al., 2013	Apel et al., 2013	Berger et al., 2015	Andam et al., 2017	Krüger et al., 2018	Rodríguez et al., 2018
Kang et al., 1990	X		✓		✓				✓	✓	✓	✓	✓	✓		✓		✓	✓			
Guerra et al., 1996		X																				
Kang et al., 1998	✓		X							✓				✓		✓		✓	✓			
Czarnecki & Eisenecker, 2000				X					✓	✓	✓		✓	✓	✓	✓		✓	✓			
Bosch, 2000	✓				X				✓		✓			✓		✓		✓	✓			
Zave, 2003						X		✓								✓		✓				
Batory, 2004							X							✓								
Batory et al., 2004						✓		X			✓		✓		✓	✓		✓				
Pohl et al., 2005	✓			✓	✓				X			✓		✓		✓		✓	✓			
Chen et al., 2005	✓		✓	✓						X				✓		✓		✓				
Czarnecki et al., 2005	✓			✓	✓			✓			X					✓		✓				
Batory et al., 2006	✓								✓			X		✓		✓						
Apel et al., 2007	✓			✓				✓					X	✓	✓	✓						
Classen et al., 2008	✓		✓	✓	✓		✓		✓	✓		✓	✓	X		✓		✓	✓		✓	
Kästner et al., 2008				✓				✓					✓		X	✓		✓				
Apel & Kästner, 2009	✓		✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	X		✓	✓	✓		
Dit et al., 2013																	X				✓	
Apel et al., 2013	✓		✓	✓	✓	✓		✓	✓	✓	✓			✓	✓	✓		X		✓	✓	
Berger et al., 2015	✓		✓	✓	✓				✓					✓		✓			X	✓	✓	
Andam et al., 2017																✓		✓	✓	X		
Krüger et al., 2018														✓			✓	✓	✓		X	
Rodríguez et al., 2018																						X

Figure 15: Overview of included works referenced by other works included in the selection.

The figure depicts which works have been referenced by other works included in table 3. It is important that these relationships show potential influence, rather than direct input. Only the articles published by Classen et al. in 2008 and Apel and Kästner in 2009 have cited exact definitions. Other than that, it can only be assumed that a definition was inspired by another based on a reference to the work in which the latter was published. It is apparent that the first paper selected in the SLR, by Kang et al. in 1990, has been used in many other works. The same is true for the works written by Czarnecki and Eisenecker in 2000, Bosch in 2000 and Batory et al. in 2004. Another remarkable observation is that three of the included works were neither referenced by nor referenced any of the other works. Interestingly, all research fields either referenced or were referenced by another field, implying that they have overlap. Unfortunately, it is not possible to determine whether authors refined or improved their definitions, if multiple have been included. For example, Batory, Apel, Czarnecki, Kästner and Berger have been included more than once. However, they were accompanied by different researchers every time, so it is uncertain whether they changed their mind about their previous or original definition, or whether the influence of the other authors caused a change in the definition.

In conclusion, there is no one 'true' or 'correct' definition. Many definitions exist and one is not necessarily better or worse than the next. Instead, only guidelines or support on how to select a definition for a specific goal can be provided. To summarize, these guidelines include: related research topic, popularity based on number of citations on Google Scholar, reputation of authors based on h-index, categorization based on abstraction and viewpoint, included terms and finally which works the work that contained the definition is referenced by and which it references.

3.4 RE4SA

This chapter discusses the theoretical background of the RE4SA model in more detail, by first describing the need for this relationship between RE and SA and then discussing all the RE related concepts included in the model.

In 2001 Nuseibeh already recognized that requirements specification and design cannot be fully separated due to their dependencies. In the Twin Peaks model, requirements and architecture specifications are defined in concurrence, while still considered as separate structures and specifications. However, the model also states that the two domains are dependent on each other, as is illustrated in figure 16.

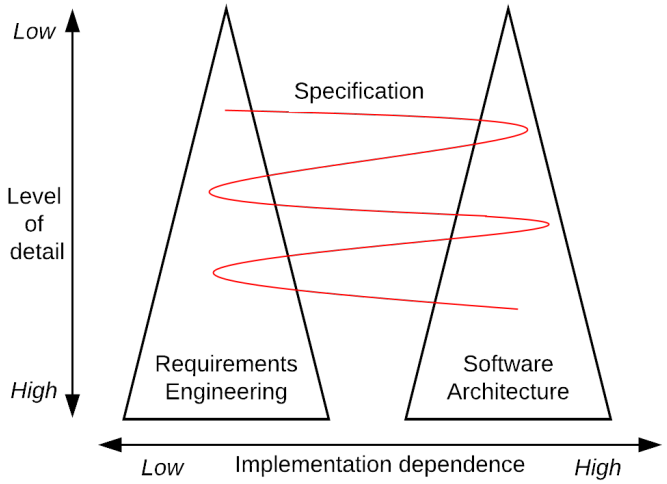

Figure 16: The Twin Peaks model by Nuseibeh.

The red line represents a software development process. According to the figure, the focus on either SA or RE shifts back and forth as development progresses, indicating an iterative and concurrent process. The strength of this model is that requirements are still able to guide architectural designs and yet architectural constraints can be taken into account when formulating requirements (Nuseibeh, 2001). Requirements can be written using various techniques, however, in the RE4SA model Epics and USs are used.

3.4.1 User Stories

USs are used to represent customer requirements (Jeffries, 2001). The most commonly used method for writing USs is the Connextra template, which was popularized by Cohn (Cohn, 2004). This template consists of three elements: who, what and why, of which the latter is optional. The format describes these three elements respectively:

```
As a <role>, I want <action>(, so that <benefit>).
```

In an industry research conducted by Lucassen, Dalpiaz, Van Der Werf and Brinkkemper it became apparent that nearly all Scrum practitioners also make use of USs, stating a use of 99% (Lucassen, Dalpiaz, van der Werf, & Brinkkemper, 2016b). An example of a context-specific US, again using the maps application as running example, is:

```
As a motorcyclist, I want to avoid densely populated areas while on a
leisurely drive, so that I do not disturb anyone with the noise of my bike.
```

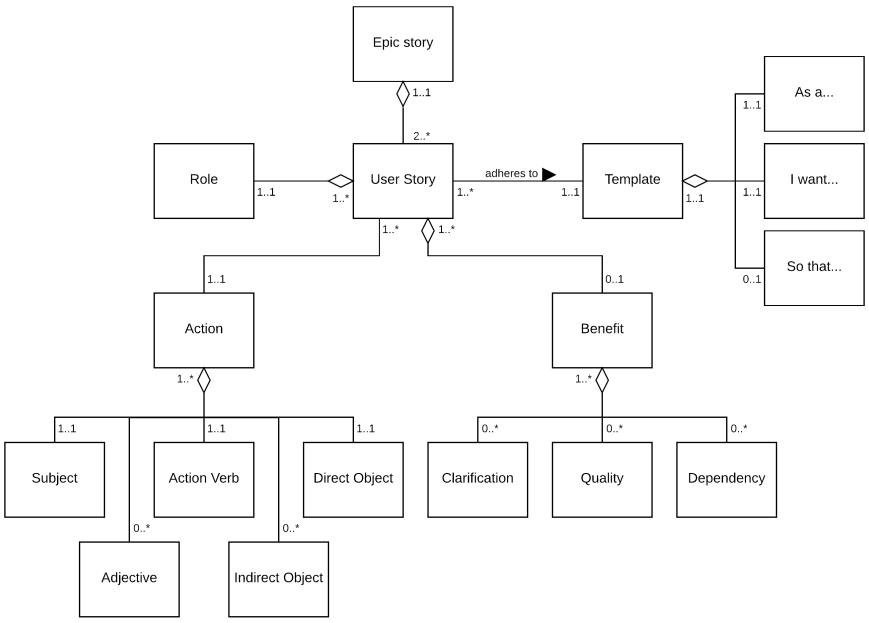

Figure 17: Linguistic model of a US.

The linguistic US model in figure 17 was adapted from a conceptual model (Lucassen, Dalpiaz, van der Werf, & Brinkkemper, 2015b). First and foremost, the "means" and "end" are formulated as "action" and "benefit" respectively in this model, to more accurately reflect the US template. In addition, the cardinality of the relationship between an Epic and a US has been changed from one-to-many to two-to-many. If an Epic were to contain solely one US, one could argue the Epic is supposed to be a US instead. Therefore, an Epic now requires at least two USs in order to be considered an Epic. Furthermore, the relationship between "role" and "role" from the original model has been omitted, since it is not of importance here. Instead of a format the template was added as a concept. Since USs do not contain the template (or formerly the format), the aggregation relationship was changed to 'adheres to'. For completeness and clarity the components of the template are included. Additionally, missing cardinalities have been added. Finally, it should be noted that this linguistic model can only accurately describe correct USs.

Lucassen et al. describe the relationship between USs and features as follows: "*the user story is the most granular representation of a requirement that developers use to build new features*" (Lucassen et al., 2016b). This description, however, does not comment on the cardinalities involved in this relationship, which is precisely the subject of RQ5. The same research team, both a month later and a year earlier, discuss the same relationship in more depth in the context of the Quality User Story (QUS) framework. This framework states that "*a user story expresses a requirement for exactly one feature*" (Lucassen et al., 2015b, 2016a). Nevertheless, two questions regarding the relationship remain unanswered. Firstly, this expresses solely one cardinality, while relationships should have two, meaning that it is uncertain whether a feature is described by exactly one US or more than one, this is visualized in figure 24. Secondly, the QUS framework exclusively includes 'correct' USs, while not all USs in practice may adhere to the rules stated in the framework.

3.4.2 Jobs, Jobs-to-be-Done and Job Stories

Despite their widespread usage and adoption, not everyone agrees with the effectiveness of USs. In fact, Klement has called for the replacement of USs with Job stories, as based on the Job-to-be-Done (JTBD) theory by Christensen. Intercom has created a template (provided below) for writing JTBDs. They describe an event or situation that serves as the trigger, the motivation and goal and the intended outcome (Adams, n.d.). According to Klement, USs contain too many assumptions and do not focus enough on the 'why' part. Instead, he proposes the use of the Job story format. He diverts slightly from the approach taken by Intercom by only describing a situation, motivation and expected outcome:

```
When <situation>, I want (to) <motivation>, so that (I can) <expected
```

`outcome>.`

This template focuses more on motivation rather than implementation (Klement, 2013b). Additionally, Klement also takes a different approach to describing the situation. While Christensen states that the triggering event or situation can be either a problem or an opportunity, Klement only considers the former, saying that: "*your Job story needs a struggling moment*" (Klement, 2016). Confusingly, Klement released a book ("When Coffee & Kale Compete") that solely mentions JTBD and Customer Jobs, never mentioning A Job story. To make matters more puzzling, the template he used for the Job story is nowhere to be found (Klement, 2018). Disregarding the complications introduced by the aforementioned book, the following conceptual model can be deducted from Klement's work as shown in figure 18.

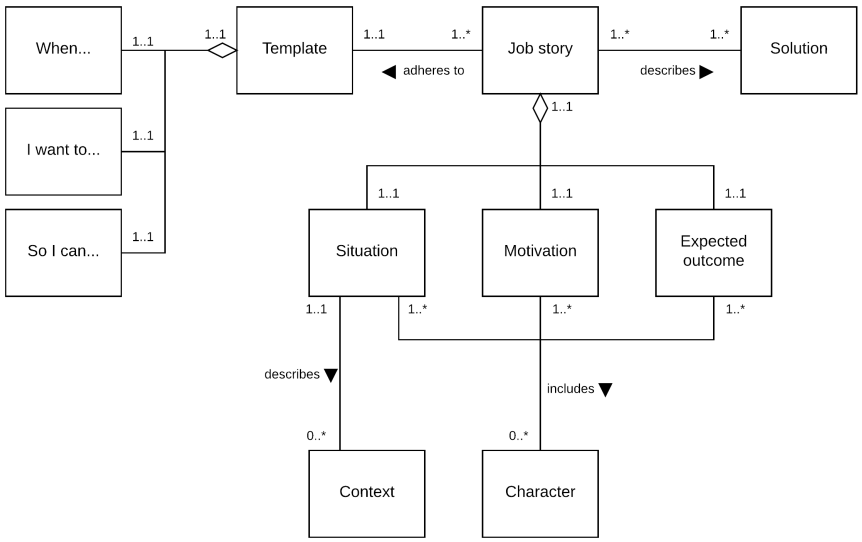

Figure 18: Conceptual model of the Job story as proposed by Klement.

The template and components of the Job story have already been described and follow Klement's original proposal of the concept (Klement, 2013b). In addition, he extends his original description by sharing five writing tips. In these tips he mentions the possibility of adding a context and character to the Job story and also includes a solution that should fit the story (Klement, 2013a).

As was briefly mentioned earlier, Klement's Job story is based on the JTBD theory as proposed by Christensen. Similarly, JTBDs are also concerned with the motivations of customers.

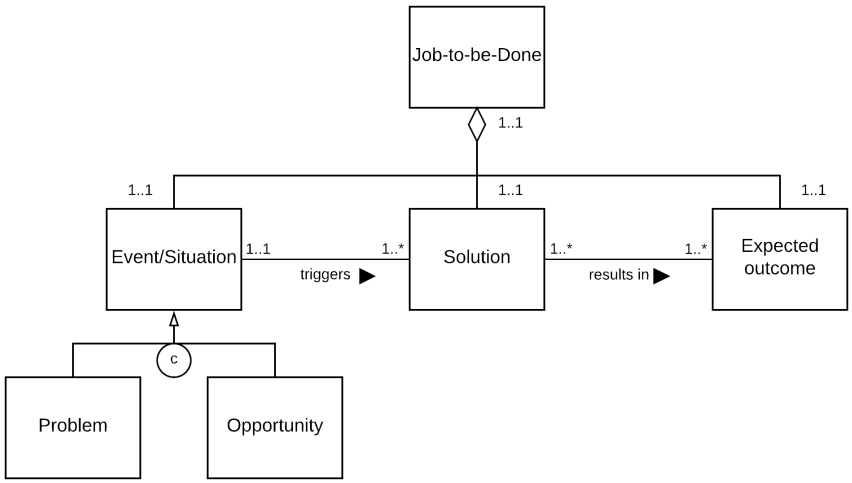

Figure 19: Conceptual model of a Job-to-be-Done as described by Christensen.

The model is explained using the example situation Christensen describes. In this example he states that people 'hire' milkshakes (Christensen, Anthony, Berstell, & Nitterhouse, 2007). Firstly, one needs to understand why they do this. For instance, because they have a long commute ahead. Christensen describes this as the situation that triggers the need for a solution. It is also possible that something suddenly occurs that warrants a solution, this is considered an event. Moreover, he distinguishes two types of triggers, namely problems and opportunities. Opportunities are more difficult to describe, since they can often be formulated as a problem. An example of an opportunity can be a desire for something, for instance, someone wants to taste a milkshake, which is not really a problem that needs solving. The aforementioned solution(s) result or should result in an expected outcome, since this is the reason people hired the product; they expect it to solve their problem. It is possible that solutions have multiple expected outcomes, for example, a milkshake can keep you from getting bored during your drive, while simultaneously ensuring you do not get hungry. It is also possible that one situation or event has multiple solution, for instance, if you want to quench your thirst you could drink water or drink a milkshake. However, a JTBD should include one event or situation, one solution and one expected outcome (Christensen, Hall, Dillon, & Duncan, 2016).

To further complicate the concepts of Job stories and JTBDs, Ulwick has taken another approach (Bettencourt was also involved, but did not seem continue to work on the Jobs). While he agrees with the idea of people hiring products, he proposes job mapping. Job mapping essentially means describing a job using eight different steps. At every step one should aim to look for opportunities for helping the customer. The conceptual model in figure 20 combines the original description of Jobs and any extensions described by Ulwick. Note that Ulwick uses the terms Job and JTBD alternately without clear reasoning, so both terms are used throughout the model and its description.

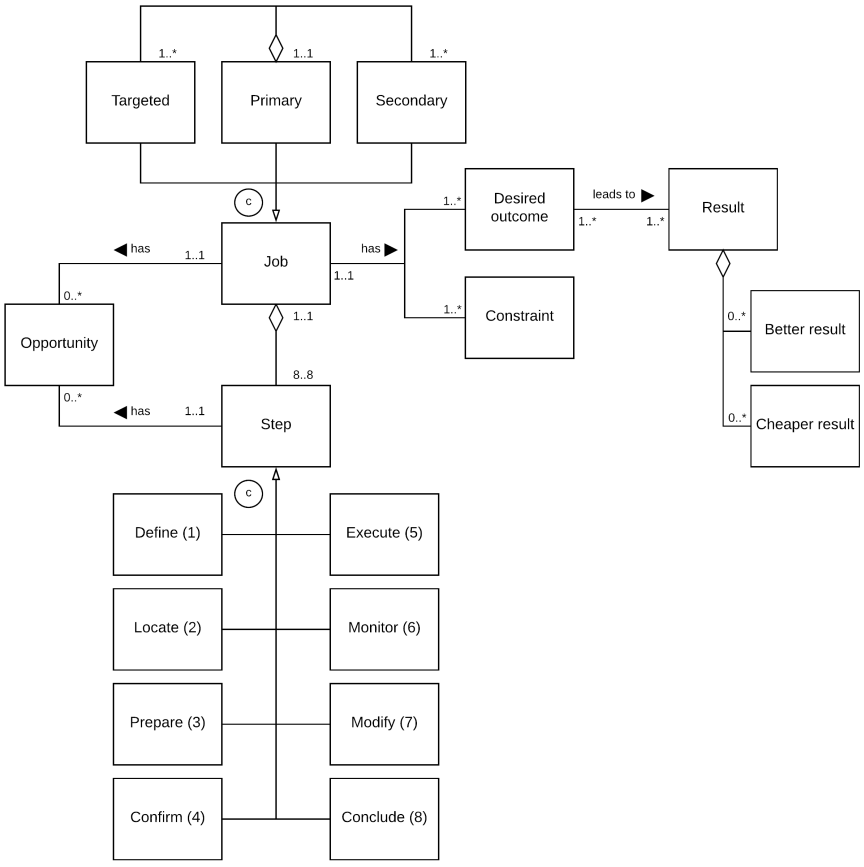

Figure 20: Conceptual model of Jobs-To-Be-Done as stated by Ulwick.

As was stated previously, according to Ulwick, a Job should be divided into different steps, eight to be precise. In the figure the numbers of the steps have been added for clarity. Every Job and every step should be analyzed to see whether there are any opportunities for improving the Job

or one or more of the individual steps (Bettencourt & Ulwick, 2008). In addition, three different types of Jobs can be described, namely: primary, secondary and targeted. Primary Jobs contain targeted and secondary Jobs. Furthermore, Jobs should have one or more desired outcomes and constraints, specified by the customer (Ulwick, 2003). The desired outcome should lead to a better and/or cheaper result (Ulwick & Hamilton, 2016). As an aside, Ulwick has also commented on the confusion surrounding the entire JTBD/Jobs theory (Ulwick, 2018). However, in this thesis no fingers will be pointed and the different theories are treated as separate entities, apart from their origins.

Lucassen, van de Keuken, Dalpiaz, Brinkkemper, Sloof and Schlingmann, however, have established a different approach. Instead of using either Job stories or JTBD theory, they have combined the two, which resulted in the Integrated Job Story method. This method considers the JTBD theory (by Christensen) as the highest level formulation of a requirement. They have designed a template for the formulation of Jobs, the template simply states (Lucassen et al., 2018):

```
Help me <verb> <noun phrase>.
```

This is in line with Christensen's idea that customers buy a product to help them do a job (Christensen et al., 2007, 2016). Then, they use the Job story (by Klement) to further define requirements for a given JTBD. Finally, at the lowest level, USs specify requirements for designing a solution to a problem described in a Job story (Lucassen et al., 2018). To avoid confusion, the Job story was later renamed to Epic story (see Chapter 3.4.3 for further details) and the JTBDs to simply 'Jobs'. In an attempt to avoid any misunderstandings, the Job as used in conjunction with the RE4SA model is illustrated in figure 21.

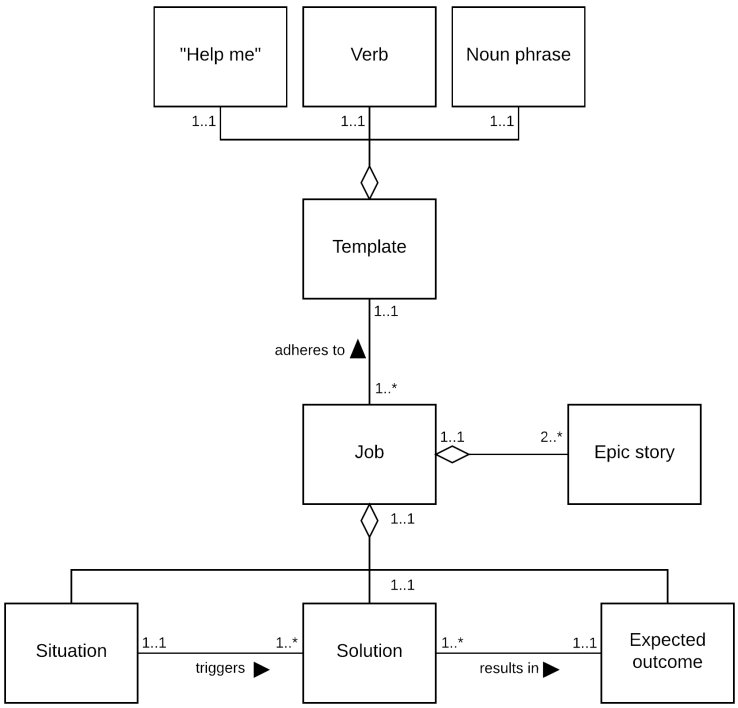

Figure 21: Conceptual model of a Job as described by van de Keuken et al.

This model differs from figure 19, since it does not specify two types of situations, instead only problematic situations are used in Jobs. Apart from that modification, only the Job template (as stated above) was added. Furthermore, the aggregation relationship with Epics is included.

3.4.3 Epic Stories

Epics are seen as large USs, since they describe a larger number of functionalities (Lin, Yu, Shen, & Miao, 2014). Van de Lucassen et al. have not merely copied the Job story by Klement and

called it an Epic. Instead, they have extended it slightly, as shown in figure 22 (Lucassen et al., 2018).

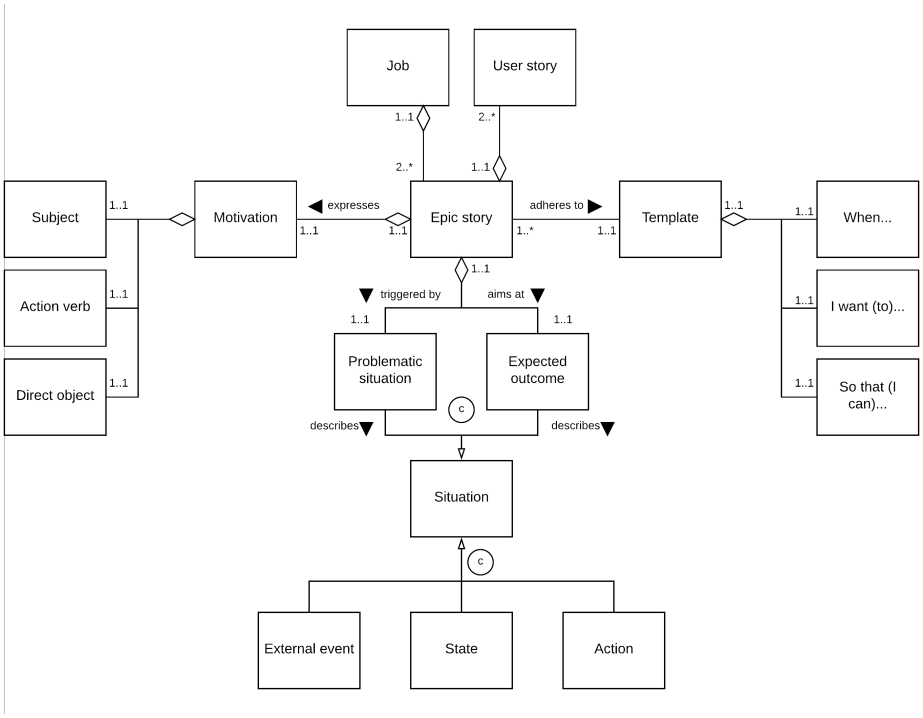

Figure 22: Conceptual model of an Epic as described by Lucassen et al.

The original model has been modified and expanded upon slightly. Firstly, three components of the template have been added for completeness. The aggregation relationships between 'Epic story' and 'template' has been changed to a 'regular' relationship, since an Epic does not contain the template, but rather adheres to it. To indicate how Epics fit into the hierarchy of the RE4SA model, relationships with the concepts Job and US have been introduced. A context-specific example of an Epic for the imaginary maps application (following the template as discussed in 3.4.2 could be:

> `When I am navigating to my destination, I want to select a scenic route, so that I can enjoy my daily commute.`

3.4.4 The Barista Problem

When considering the RE4SA model, the main granularity issue exists in the middle layer, so in Epics and modules (given that Jobs and Products can be placed at the top, refer to figure 23 and figure 24). Epics can be formulated broadly, turning them into Jobs or in detail, making them USs. Blessinga refers to this granularity challenge in his thesis as 'The Barista Problem'. This problem states that even a job with simple activities can be difficult to model (Blessinga, 2018). For instance, one can formulate the following Epic for a barista:

> `When a customer places an order, I want to make their coffee, so that I can satisfy their request.`

However, this Epic can also be formulated as a US:

> `As a barista, I want to make coffee, so that I can satisfy customers' requests.`

To make matters worse, a Job can also be written:

> `Help me satisfy customers' requests.`

The formulation and thus the level of abstraction is dependent on the level of detail that is required to properly state a requirement using Jobs, Epics or USs. To solve the granularity issue, Blessinga

has adopted the approach taken by the JTBD theory. This states that high and low level Jobs can be formulated, depending on how much abstraction is needed. Using this method, USs are at the lowest level of abstraction, followed by Epics and then Job, high or low depending on necessity. Basically, as opposed to defining multiple levels of Epics, the issue is moved up to the Job level (Blessinga, 2018). Taking pages out of the books of other research does not solve the issue, unfortunately. Epics as they are presented in Chapter 3.4.3 could not be found in literature (apart from the article cited in said chapter). Instead, other interpretations of Epics are used. For example, Wautelet et al. group USs around themes and call these themes Epics (Wautelet, Heng, Kolp, Mirbel, & Poelmans, 2016). Another approach is to use 'large' USs (also referred to as Epics), which are basically regular USs written to be very broad (Lin et al., 2014). However, making USs less specific and detailed is in conflict with the quality requirements as described in the QUS framework, which states that USs should be atomic (Lucassen et al., 2016a). Finally, one more level of granularity was identified in literature, namely tasks. Tasks are positioned below USs and further refine requirements (Patton, 2005). They are, however, mostly used in Agile development, during sprints, which is why they are not investigated any further in this research.

3.4.5 Research Scope

Taking these theories and templates into account, the structure is at follows: Jobs at the highest level, then Epics followed by USs. Taking this hierarchy of Jobs, Epics and USs into consideration, the RE4SA model can be extended to better suit this research, as illustrated in figure 23.

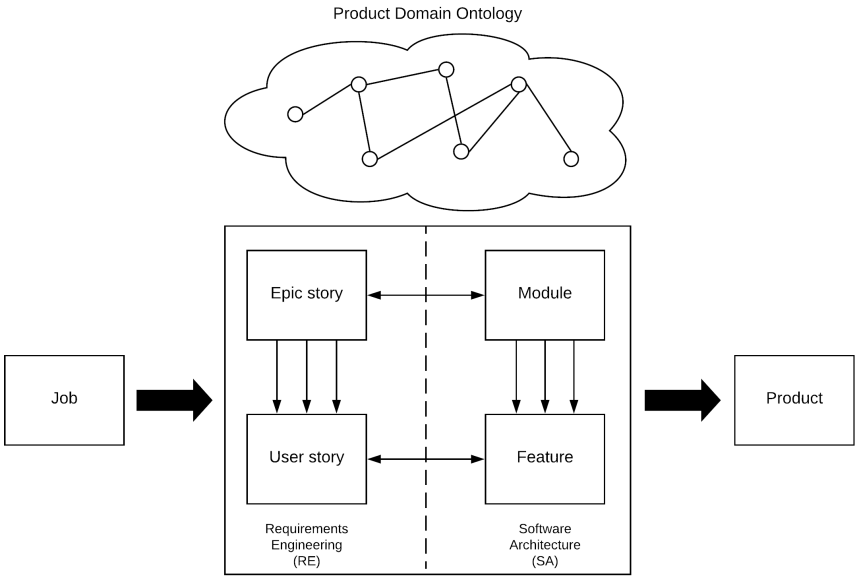

Figure 23: The RE4SA model in the scope of this research.

On the left side, Jobs are presented as input for the RE4SA model, as they can be further refined into Epics and subsequently USs. On the right side a product can be defined, which is the combination of all modules and features, resulting in a system, application or piece of software, which is referred to as a product. Finally, the Product Domain Ontology (PDO) (also refer to Chapter 3.7), which is a domain ontology that describes a product, encompasses both domains (Martens et al., 2018). All elements (Epics, USs, modules and features) may contain concepts and the relationships among them, which can be included in the PDO. The scope of this research is restricted to the original RE4SA model (between the two arrows), with the addition of linguistics in the form of the PDO. In the context of meta-modeling, the RE4SA model can be visualized as shown in figure 24 (cardinalities in italics are (reasonable) assumptions).

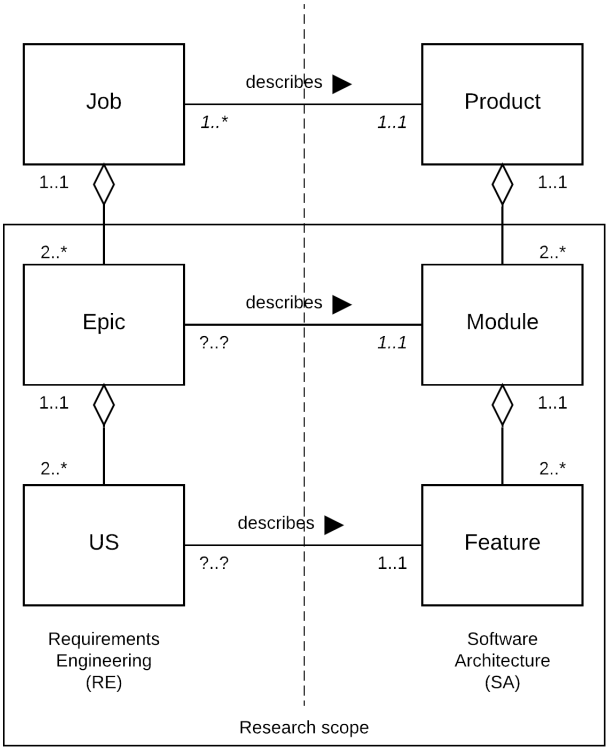

Figure 24: The meta-model of the concepts included in the RE4SA model.

An Epic contains at least two USs, since otherwise the one US that is contained within the Epic might as well be an Epic itself or the Epic it is related to is actually a US. The same is said for modules and features in this model. According to Lucassen et al., USs describe the requirement for exactly one feature, hence the one-to-one cardinality (Lucassen et al., 2015b). This is assumed to be the case for Epics and modules as well, following the same reasoning, but has not been verified. The most interesting cardinalities here are the missing ones, which can, hopefully, be added at the end of this research. For completeness, Jobs and Products have been included, even though they are outside of the scope of this research.

Theoretically speaking the relationship between features and USs should have a one-to-many cardinality on the US side. Considering that a feature can be linked to more than one US, if two different roles (stakeholders), desire the same action. However, does the module to which the feature belongs change the situation and can the benefit in the US impact the implementation? Furthermore, would the involved USs and feature still adhere to the template rules and quality criteria? Additionally, one can question the need for two USs with the exact same action and different roles, since it might be possible to group these roles and reduce the number of near-duplicates. However, consider the role structure as illustrated in figure 25.

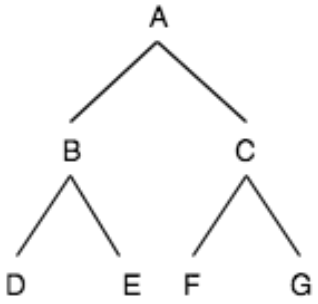

Figure 25: Example of a tree structure of different roles.

Assume that there are four different roles, which can be grouped in two different roles and, at the top, be described by one role (for example 'stakeholder' or 'user'). If a feature concerns roles D and E, they can simply be grouped by stating role B. However, if roles D and G desire the same feature, are they then grouped in a new role H to ensure that a feature is described by only one US? For a case with few different roles this can be a feasible approach, but for larger, complex cases this could lead to a big and confusing tree structure. Finally, in feature diagrams, it is syntactically correct to model a feature with more than one parent-feature or parent-module. Lee, Kang and Lee provide an example of such a case. They state that some complex relationships can be simplified by looking at the closest common parent of two or more features that are related to the same child-feature. This does not always solve the issue though, which means one is still left with two parent-features being related to the same child-feature (Lee, Kang, & Lee, 2002). So, returning to the original statement, the cardinality should theoretically be one-to-many, but three questions remain, namely: (1) does this happen in practice, (2) is this correct and (3) is this desirable?

3.5 Functionality in RE and SA

As part of answering RQ2, functionality as defined in literature is investigated. Since the RE4SA model consists of two main dimensions, functionality is considered in the context of both perspectives, starting with RE.

In terms of RE defining 'functionality' might seem easy at first glance, since '*functional* requirements' can be formulated. According to SWEBOK: "*Functional requirements describe the functions that the software is to execute; for example, formatting some text or modulating a signal*". They can also be characterized as requirements for which test steps can be written to validate whether they function correctly and do what they are supposed to do. This definition, however, does not imply the granularity of functionality. In the SWEBOK guide, USs are said to address (required) functionality by describing it in customer terms. The reasoning for this is that it helps developers estimate the time needed to implement the US or required functionality and that acceptance tests can be written to validate the behavior (Bourque & Fairley, 2014). This confirmed by Gilb and Finzi, who state that in principle a "functional requirement specification lists essential things which the product must do, and which must be delivered at specified times". In addition they mention that function is either present or not and cannot be semi-implemented, for instance (Gilb & Finzi, 1988). This notion of functionality being behavior the system must exhibit or express is also agreed upon by Nuseibeh and Easterbrook. While they do not explicitly define functionality, they touch upon it while describing behavioral modeling. They explain that such models can be analyzed "to determine essential functionality" (Nuseibeh & Easterbrook, 2000).

Given the former description of functional requirements, specifying functionality should be at the lowest level of abstraction, since, according to the RE4SA model, USs are at the lowest level. The following can be said about functionality in the context of RE (given the previous information):

1. Functionality is described by stating functions using functional requirements;
2. Functionality can be described in USs;
3. Functionality is something the system must do;
4. Functionality must be delivered at specified times;
5. Functionality is either implemented or not;
6. Functionality must be tested to validate the behavior.

While the SWEBOK guide states that USs describe functionality in customer terms, this sentiment is not shared. USs can indeed be used to describe customer wishes, but the roles specified in USs are not limited to customers of the product or system. Other stakeholders can be included and they are allowed and able to express desires for a system's functionality as well.

Following the RE4SA model and the previous reasoning, functionality in SA should be located at feature-level, since this is the lowest level of architecture and also related to USs. According to Bosch, functionality in SA is something that is demanded from and imposed on architectural components. He also adds that design decisions can "*add requirements on the expected behavior of components*", which is in line with the behavioral aspect in the RE context. Unfortunately, he does not properly define 'components'. They are described as key concepts in SA, along with connectors (Bosch, 2004). Regrettably, this does not establish whether modules, features or both are

considered components. Brinkkemper and Pachidi utilize a modular decomposition for functionality of a product or system. The FAMs illustrate the primary functionality of a product through modeling its modules. Each module should implement a part of the product's functionality. Since modules consist of features, this would mean that the functionality of a product is also present in its features (feature diagrams). In addition, when they describe modeling flows that represent the main functionality, these flows can include both modules and features (Brinkkemper & Pachidi, 2010). Given the former explanation of functionality being present in the architectural components and the latter of functionality in modules and by extension in features, it becomes apparent that functionality is specified on all levels of abstraction in the context of SA.

3.6 Linguistics

As visualized in figure 23, the third overarching subject in this research is linguistics. Many different aspects can be included, but only the most applicable and relevant ones are discussed in this chapter, these being: ambiguity, relationships in terminology, parse trees, similarity, ontologies and linguistic structures.

Language errors, such as vagueness and ambiguity, can cause misunderstandings between stakeholders and can ultimately cause the system to be insufficient given the needs of stakeholders (Dalpiaz, van der Schalk, & Lucassen, 2018), which is why semantics are of importance in RE and SA. Shaw and Gaines describe four types of relationships between concepts, dependent on chosen terminology by their users (Shaw & Gaines, 1989):

1. Consensus: same terminology, same distinction: "experts use the same concepts in the same way"
2. Correspondence: different terminology, same distinction: "experts use different terminology for the same concepts"
3. Conflict: same terminology, different distinction: "experts use the same terminology for different concepts"
4. Contrast: different terminology, different distinction: "experts differ in terminology and concepts"

Essentially, in case of the former, people use the same word with the same intended meaning and in case of the latter people use different words with different intended meanings. These two types of relationships are not of interest in this research. The other two types, correspondence and conflict, or synonyms and homonyms respectively, are of significance. When two people are in correspondence, they will use different words that have the same meaning, so synonyms. When two people are in conflict, they use the same word that has two different intended meanings. For example, one person might use the word 'pen' to denote something you can write with, while another person might use it to describe an enclosure.

Cimiano et al. have proposed an approach to determining synonyms, namely by taking the context in which a term appears into account. They calculate the similarity of concepts by applying the Jaccard coefficient to terms extracted from the corpus. In short, this coefficient considers the nouns from a parsed text, along with the verbs they are related to. Based on their individual relationships with these verbs, a score from 0 to 1 can be calculated, in which 1 is a perfect match (the exact same term) and 0 means no similarity (Cimiano, Mädche, Staab, & Völker, 2009). In this case, the 'corpus' refers to "*any collection of written or spoken texts*" and, in modern linguistics, also denotes a finite size, machine-readable form and a standard reference for a language among others (Lüdeling & Kytö, 2008). Cimiano et al. also discuss the use of lexico-syntactic patterns. However, since these patterns as the researchers describe them are not expected to occur in either RE or SA, they are not discussed here. Another analysis technique that may be of use is agglomerative clustering. This technique clusters different concepts based on their similarity, using single linkage. Using this technique, a cluster can be created that contains all extracted concepts or until a predetermined stopping point is reached (Cimiano et al., 2009).

Hosseini, Breaux and Niu are more specific in their linguistic approach, as they define five terms in relation to describing ontology fragments (Hosseini et al., 2018):

1. Hypernym: "a noun phrase, also called a superordinate term, that is more generic than another noun phrase, called the hyponym or subordinate term"
2. Meronym: "a noun phrase that represents a part of a whole, which is also a noun phrase and called a holonym"

3. Synonym: "a noun phrase that has a similar meaning to another noun phrase"
4. Lexicon: "a collection of phrases or concept names that may be used in an ontology"
5. Ontology: "a collection of concept names and logical relations between these concepts, including hypernymy, meronymy and synonymy among others"

The aforementioned terms can be used to describe terms extracted from the software artifacts. In addition, these denotations are also used to compare two terms and determine how likely or strong the relationship between them is. Lexical relationships are only used to determine synonymy or hyperonymy, since lexicons are linguistic objects and ontologies are not. It is not, however, within the scope of this research to develop a lexicon (vocabulary) for the RE and SA concepts stating precise definitions (Hirst, 2009).

To be more specific given the context, an ontology can be defined as follows: "an explicit specification of a conceptualization. An ontology specifies the concepts, relationships, and other distinctions that are relevant for modelling a domain" (Grüber, 1995). Guarino distinguishes four different types of ontologies: top-level, domain, task and application. In this context, only domain ontologies are considered, which describe "*the vocabulary related to a generic domain*" (Guarino, 1997). In short, an ontology, or to be more specific a 'domain ontology', can be seen as a 'shared language' within a certain domain. More specifically, the ontology contains domain concepts, the relationships among them and groups them based on similarity. An example of this can be found in Chapter 3.7.

Once ontologies are created, they can be used to write textual descriptions. Androutsopoulos, Lampouras and Galanis presented NaturalOWL, which is a system that can automatically generate a natural language description of concepts or classes within an ontology (Androutsopoulos, Lampouras, & Galanis, 2013). While generating a description of SA artifacts using such an approach can be interesting for development, it is not within the scope of this research. However, the usage of parse trees within this system can be of use still. These parse trees show the linguistic structure of a certain sentence(Lampouras & Androutsopoulos, 2018). Parse trees and linguistic structures can be especially useful in the light of artifact generation and naming conventions.

Despite the existence of domain ontologies, it is still possible that stakeholders misinterpret or misunderstand each other. Natural language oftentimes introduces ambiguity. Berry, Kamsties and Krieger omit the uncertainty aspect of ambiguity and instead use the following definition: "*the capability of being understood in two or more possible senses or ways*". In addition, they identify four different types of ambiguity, namely (Berry, Kamsties, & Krieger, 2003):

1. Lexical: happens when a word has more than one meaning
2. Syntactic: when phrase can have multiple grammatical structures and thus multiple meanings
3. Semantic: when the meaning of a phrase depends on how it is read within the same context
4. Pragmatic: when the meaning of a phrase can differ given different contexts

The aforementioned types of ambiguity can be split into even more specific types of ambiguity, but these are not included here. Ambiguity is only expected in RE documents, since this is where natural language is most prevalent. Chances are that these ambiguities will not have a large impact on the names and descriptions of features and models, since these can be quite generic. Rather such misunderstandings introduced by ambiguity should be visible in the source code, where the exact implementation and interpretation of requirements becomes apparent. Unfortunately, source code is not within the scope of this research, unless SA artifacts need to be recovered. It is, however, important to keep ambiguity in mind when identifying and validating potential linguistic links (parse trees are discussed in more detail in Chapter 3.6.1.

Linguistic links will not solely be established by a yes or no answer to the question is there a linguistic relationship. The links will also be analyzed based on their linguistic structure, which was briefly touched upon earlier. Müter et al. have analyzed tasks that resulted from USs on their linguistic structure. This analysis lead to the identification of the ten most frequently observed linguistic structures of task labels and the most frequently occurring action verbs in task labels. They describe a linguistic structure using the types of words used, for example nouns, verbs and adjectives among others (Müter, Deoskar, Mathijssen, Brinkkemper, & Dalpiaz, 2018). The linguistic relationships can be analyzed in order to identify the most frequently occurring linguistic structure, but also whether the structure of a name of a module or feature can be inferred from the Epic or USs it is related to.

In addition, it might be valuable to attempt to identify semantic frames. The concept can be defined as follows: "*frame semantics involves the specific structures of encyclopedic knowledge that it invokes. Basically, these 'frames' are things happening and together in reality*" (Fillmore, 1982). To put it simply, semantic frames describe which words are often found in combination with other words or which words you can expect given some other word. For example, words such as 'banish', 'extradite', 'evacuate' and 'deport' are all used to describe removing a person from a certain location or situation (Goldberg, 2010). Therefore, one can expect a human being that is removed and a place or circumstance from which they were removed, without needing knowledge of the exact sentence. Two examples of repositories containing such frames are 'FrameNet' and 'Levin's English Verb Classes' (Baker & Ruppenhofer, 2002). As the name may suggest, the latter only includes verbs. Since software development does not solely include verbs, FrameNet was examined. Using the word 'building' (the verb, not the noun), the following information is presented: a definition, the semantic type, the core related words and the non-core related words. In this the core contains an agent (someone who builds), components (building materials) and created entity (what is being built) ("FrameNet Data", n.d.). However, the FrameNet index might not contain all terms that come up in the cases, so the English Verb Classes can be used as well (Levin, 1993).

Fillmore and Baker have provided an example of how text can be broken down into the individual words and their dependencies, using Part-of-Speech (PoS) tagging. To summarize, the following steps are performed (Fillmore & Baker, 2001):

1. Divide text into sentences.
2. Write down all the words in a specific sentence.
3. Index the words in said sentence by numbering them.
4. Determine the PoS tag for each word.
5. Identify the dependent elements for each word (it should be noted that not every word needs to have a dependent).

This technique will be used to determine which kinds of words are commonly used in the various artifacts. Since they mainly form the basis of identifying the linguistic relationship between artifact instances, the dependencies of the elements are not included, as the focus is not on the linguistic structure of individual phrases.

3.6.1 Linguistic Structures in RE4SA Concepts

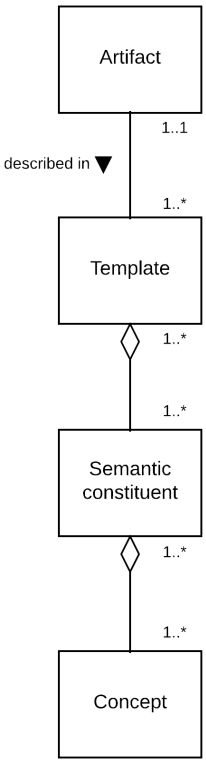

Figure 26: Linguistic decomposition of artifacts (meta-meta model).

In the context of this research, artifacts (both RE and SA) can be described in one or multiple templates. These templates can be divided into narrative and diagrammatic templates. The former type can be found in RE artifacts, especially Epics and USs (templates are provided in 3.4). The latter are found in SA artifacts, namely FAMs, FADs and feature diagrams (refer to 3.2. Both types of templates consists of two parts: semantic and syntactic constituents. In short, these can be described as the filled in texts and fixed texts respectively. In case of narrative templates distinguishing between semantic and syntactic constituent is quite self-explanatory, since the templates itself contains fixed texts. Consider the following US:

```
As a (user), I want to (book a ticket), so that (I can attend the event).
```

The phrases in parentheses are the filled in texts (semantic constituents) and the remaining phrases are part of the fixed texts in the template (syntactic constituents). In diagrammatic templates identifying semantic and syntactic constituents is less straightforward. Diagrammatic templates, such as FAMs, can include fixed texts, or rather non-semantic constituents. In this case, that means that the constituents are not required to understand and read the model or diagram. For example, in FAMs, databases can be modeled. Databases, however, are indicated by a cylinder shape that are used solely for this purpose. Even so, these shapes often have a name that ends with 'DB'. Similarly it is possible that module names end with 'module'. Such constituents can be considered syntactic.

From these semantic constituents, concepts (or linguistic terms) can be extracted, as one would when constructing an ontology. Considering the aforementioned US as an example, the concepts per semantic constituent are: 'user', 'book' and 'ticket', 'attend' and 'event'.

Based on the aforementioned literature and figures, some linguistic structures present in the RE4SA concepts can be extracted, which are presented in table 7.

Concept	Mandatory	Optional
Job	Verb, noun phrase	N/A
Epic	Subject, action verb, direct object	N/A
US	Subject, action verb, direct object	Adjective, indirect object
Module	Noun (substantivized)	Noun, verb
Feature	Verb, noun phrase	N/A

Table 7: Linguistic structures found in RE4SA concepts.

According to the figures shown in Chapter 3.4, Jobs and Epics exclusively consist of mandatory structures, namely verbs, noun phrases and subjects, action verbs and direct objects respectively. Since USs have an optional element (the 'why' element), subjects, action verbs and direct objects are mandatory to include, while adjectives and indirect objects are optional. Note that the structures related to the template are omitted. According to Brinkkemper and Pachidi, module names should always start with a noun, a substantivized noun if applicable (for example, 'planning' as opposed to 'plan'). However, the example FAMs and FADs provided also show modules that contain additional nouns and verbs (such as 'bookkeeping application' or 'order processing') (Brinkkemper & Pachidi, 2010). For features, Brinkkemper specifies that they should start with a verb followed by a noun phrase, for instance 'save file'. He also states that features visualized in feature diagrams do not often conform to these naming conventions (Brinkkemper, 2018). However, they can be reformulated quite easily. For example, if a feature in a feature diagram is simply named 'window' it can be changed to 'open window' in order to comply with the naming conventions, without changing the feature's purpose.

Given that subjects are nouns (or perhaps noun phrases), it may be possible to identify patterns between the formulation of Epics and the names of modules. Similarly, the action verb and the potential verb(s) in a module may correspond. In the case of USs and features, the subject and noun phrase can be compared, as well as the action verb and the verb.

As was briefly mentioned earlier, parse trees can be used to model sentence structures (constituency-based parse trees to be precise). Consider the following sentence as an example: "The girl read a book at home." The parse tree of this sentence is illustrated in figure 27.

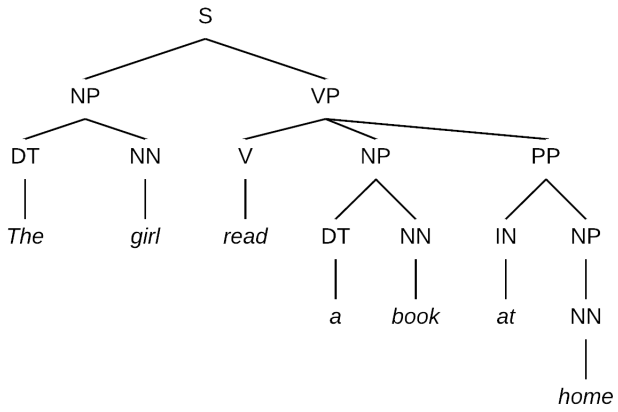

Figure 27: Constituency-based parse tree example.

The meaning of the abbreviations (tags) can be found in the list of the abbreviations. These tags are largely based on the Penn Treebank tagset (M. Marcus, Marcinkiewicz, & Santorini, 1993). The only exception is that verbs are not distinguished based on tense and person, but are simply tagged as a verb only. The tags will be used to create links between RE and SA documents and their concepts, so the specific types of verbs are not expected to be of any importance. When two words are the same and receive the same tag (so distinguishing verbs and nouns that can be spelled exactly the same), this will be called a linguistic match. It is important to note that two words also match regardless of tense in case of verbs and plural or singular in case of nouns, with the exception of this difference changing the meaning or purpose of the artifact. Then, if a

match is unique, which implies that there are no other matches with those words, the match can be considered a linguistic link. In addition, the types of phrases are also identified: noun, verb or prepositional phrases. Finally, the words that appeared in the input sentence are put in italics to improve legibility.

3.7 Traceability

Traceability refers to the linking various software artifacts to each other and thus also plays a role in the relationship between RE and SA.

Software development results in various software artifacts. A software artifact "*is a tangible piece of information obtained through the software development process*" (Schach & Tomer, 2000). Every artifact in the production process serves a purpose and the relationships between them can be informative and useful (Bouillon, Mäder, & Philippow, 2013). However, the software artifacts are not connected initially, meaning that the relationships between them are either implicit or unclear (Cleland-Huang, Gotel, & Zisman, 2012). Software traceability is a means for establishing such connections: "*software traceability is the ability to interrelate any uniquely identifiable software engineering artifact to any other, maintain required links over time, and use the resulting network to answer questions of both the software product and its development process*" (Cleland-Huang et al., 2014). Traceability can help to expand the ontology of a product, by continuously adding new concepts and relationships. A growth in the ontology can also result in an increase in the semantic understanding of the concepts included in said ontology.

Links such as those described in the previous definition are often referred to as trace links, these can be used for change management. The activity of predicting how changing requirements affect other domains in software development is also referred to as impact forecasting. Figure 28 illustrates how changes in different domains can impact the other domains by means of the relationships between them.

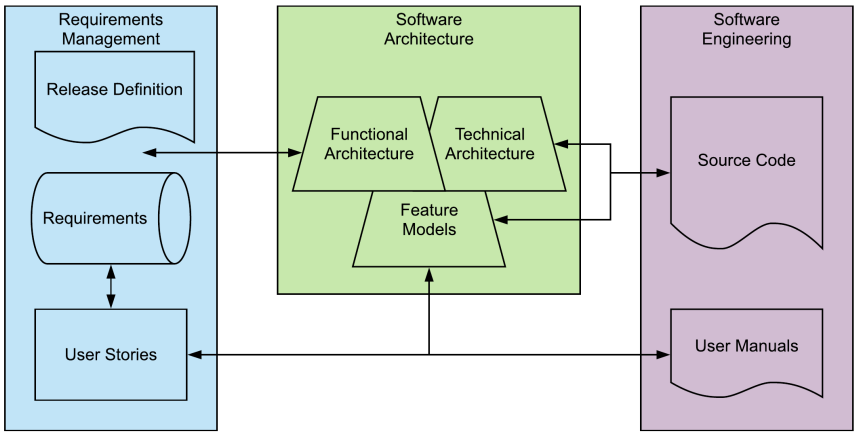

Figure 28: Visualization of impact forecasting on three development domains (Brinkkemper, n.d.).

When on the topic of changing requirements, generally three different types of changes are considered throughout this research: adding, modifying or deleting requirements (Brinkkemper, n.d.).

Traceability can also be used to transform existing artifacts into new ones, or rather, generating new artifacts using information contained in existing ones. Lucassen et al. present the Behavior-Driven Traceability (BDT) method, which uses automated acceptance tests to link requirements to source code. The BDT method depends on two characteristics of Behavior-Driven Development (BDD), namely its description of end-user interaction in steps and the run-time execution of these steps rather than source code, basically it simulates usage of the system as opposed to just running part of the code in isolation (North, 2006). The BDT method utilizes the BDD tests by generating a trace that links a US to a piece of source code. Then, it saves these traces in a matrix, which allows developers to see all source code related to the realization of a particular US. This method is expected to be beneficial for change impact activities and bug fixing (Lucassen, Dalpiaz, van der Werf, Brinkkemper, & Zowghi, 2017). Moreover, it might be possible to generate new acceptance

tests by using USs as input, for example by extracting the phrases not included in the template (Brinkkemper, 2018).

Trace links can be established manually or by means of tools. In case of the latter it is also possible to identify trace links by using ontologies, meaning that if two artifacts share a domain concept within an ontology, a trace link is established. Basically this approach, called the PDO traceability method extracts terms, uses these to create a sub-ontology and subsequently compares the two sub-ontologies by calculating a similarity score (Martens et al., 2018). The process of this method is visualized in figure 29 (steps should be read from bottom to top).

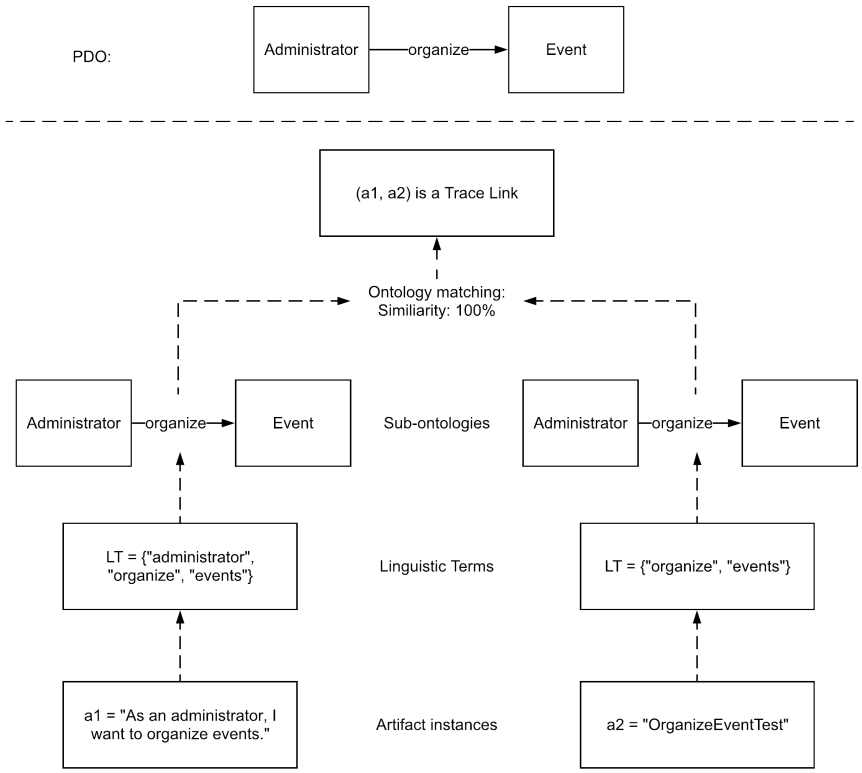

Figure 29: Example of the PDO traceability method process.

The PDO method can also be used to suggest names within artifacts. For example, if there is a requirement that describes booking tickets and a piece of code describes booking a coupon, the method could suggest to change the word 'coupon' in the code to 'ticket'. These naming suggestions are based on the sub-ontologies extracted from the artifacts. Unfortunately, these sub-ontologies are not always reliable. Whenever a sub-ontology is fully contained within another sub-ontology, this received similarity score of 100% and is thus considered a perfect match. However, if the sub-ontology that is contained consists of only two terms, while the second consists of twenty, this is also considered a perfect match even though just two out of twenty terms correspond. In addition, this ontological traceability approach does not take semantics into account, as only identical terms are taken into consideration and synonyms are ignored. Therefore, software artifacts will be connected based on 'linguistic links' throughout this research, as they are expected to be insightful still.

Latent Semantic Indexing (LSI), does, unsurprisingly, take semantics into account. LSI captures the meanings of words and phrases, also including the context. Based on this, constraints can be determined with which to deduct the similarity of the meaning of sets of words. However, again, this approach does not consider all aspects of semantics, there is still a chance that synonyms are ignored or that homonyms are included when they should not have been. In addition, this approach has solely been used to trace documents to source code, not taking SA artifacts into account (A. Marcus & Maletic, 2003).

3.8 Naming Conventions for Models

Given previous chapters, a little detour is required to explain the different kinds of models included in this research. While modeling itself can be difficult, finding the right name for that model is also challenging. For instance, figure 17. The main cause for this issue is the fact that there are various levels of abstraction in these models. Originally, this was done to include as many details as possible, while still generalizable and thus applicable to most if not all cases.

Firstly, when discussing conceptual models, a distinction should be made between the model of a concept and a model that is conceptual. In case of the latter, there should be a real-world instance of what is being modeled. In case of the former, however, there is no need for a real-world instance of the concept. Modeling concepts is especially useful since it can be used to visualize concepts that have no physical representation. According to Gregory, this leads to a dilemma, because if a conceptual model does not necessarily need a physical instance, then there is no certainty that there ever will be a conceptual model that corresponds to a physical instance. An approach to solving this is to consider logico-linguistic conceptual models, which are based on stakeholders creating and agreeing on a language for describing the problem. Subsequently, when all stakeholders agree, the models are provided with a syntactical structure (Gregory, 1993). However, this approach moves conceptual models into the context of predicate logic, which is not the desirable method for description here.

Kung and Sölvberg argue that conceptual models serve four roles (Kung & Sölvberg, 1986):

1. Used as a reference framework to facilitate communication with future users of a system
2. Used to model reality, to improve understanding of the application domains and user needs
3. Basis for design and implementation of a database
4. Part of the documentation, used during maintenance and evolution phases

According to this explanation, however, conceptual models are aimed towards the representation and design of systems, while, in the previous chapter, other concepts are modeled, such as Epics. Entity Relationship Diagrams (ERDs) also have a prominent place in the context of information systems modeling. Since these are focused on database modeling (P. Chen, 1976), the name ERD would not be desirable either, due to possible confusion and misunderstandings. To emphasize the overlap between software development and linguistics, the term ontology could be used here. On the other hand, when looking at the following definition of an ontology: "*an explicit specification of a conceptualization*" (Grüber, 1995), the term does not seem appropriate, since the models still represent concepts. Another type of model to consider is a linguistic model. Unfortunately, these are often concerned with logic as well, making them mostly rule-based (Gacto, Alcalá, & Herrera, 2011). Finally, a meta-modeling approach can be considered. One such specification is Meta-Object Facility (MOF), which describes four levels of models: meta-meta, meta, model and data/information. This does not solve the issue of naming the model, since there are no specific types mentioned in the model layer (M1) (OMG Group, 2002). Kühne presents another meta-modeling approach. He uses token models, which do not include cardinalities, type models and token & type models, the latter two do specify cardinalities. To be able to go into more detail using this meta-modeling approach, he allows for ontological and linguistic instantiations of the models. The former referring to concepts (such as 'novel') and the latter referring to specific instances (such as 'Moby Dick'). He calls such linguistic instances of ontological instances of a meta-model linguistic meta-models, even though there are not technically meta-models according to the MOF specification (Kühne, 2006).

To summarize, the models cannot (accurately) be considered meta-models, since they are not models of models, but rather a model or visualization of a concept and its template. Secondly, it is not technically a conceptual model in the context of information science, since these models tend to describe systems. Thirdly, linguistic models are often rule-based, while these models are not. So, while all of these names are somewhat feasible options, they are not 100% accurate and can lead to confusion, especially across fields of research. However, since these models are used for research in the information science community, the importance of accuracy in the field of linguistics is decreased. Therefore, the term 'linguistic conceptual model' is proposed.

4 Case Study

In this chapter the case study is described in further detail. To start, the required preparations are discussed, which include the case selection criteria and process, as well as the steps that are executed. As was stated previously, the cases are performed sequentially, so a case must be fully completed prior to the start of the next one. All cases are investigated following the steps presented in the remainder of this chapter according to the replication design as stated by Yin (R. Yin, 2017), unless specified otherwise in Chapters 4.2 and 4.1.3 (due to potential additional steps).

4.1 Case Study Preparation

Before analyzing each case, materials and background information are required, as described later in this chapter. In addition, since multiple cases are under investigation, selection criteria are used to ensure they are somewhat similar and comparable.

4.1.1 Case Study Selection

The cases included in this research must adhere to a set of selection criteria. The first being that (extensive) requirements documentation, specifically USs, must be available. As was mentioned earlier and is explained later, architecture documentation can be recovered if need be. Likewise, Epics can be formulated based on the USs, although groupings, themes and large USs could prove to be useful, since they already provide some direction as to which USs describe similar functionality. Secondly, the selected cases must not be or include legacy software, but be a stand-alone piece of software. This coincidentally also serves as the case boundary. Since legacy software is built in several phases that depend on each other and keep extending the software, the architecture runs a risk of becoming too complex and large. This could (over)complicate the identification of (linguistic) links and dependencies between RE and SA artifacts. Moreover, it could lead to establishing links that should not exist, introducing false positives. Thirdly, the source code or the access to the system itself should be made available, in case any SA documentation needs to be recovered. In addition it would be preferable if someone that was involved in the project was available for questions, should there be any. Finally, supplementary documentation is desirable, but not required.

4.1.2 Case Study Preparation and Data Gathering Approach

For each case, the gathering and preparation steps as shown in figure 30 are executed. It should be noted that this approach is devised prior to the case study, any deviations from the original steps are discussed in Chapter 4.2.

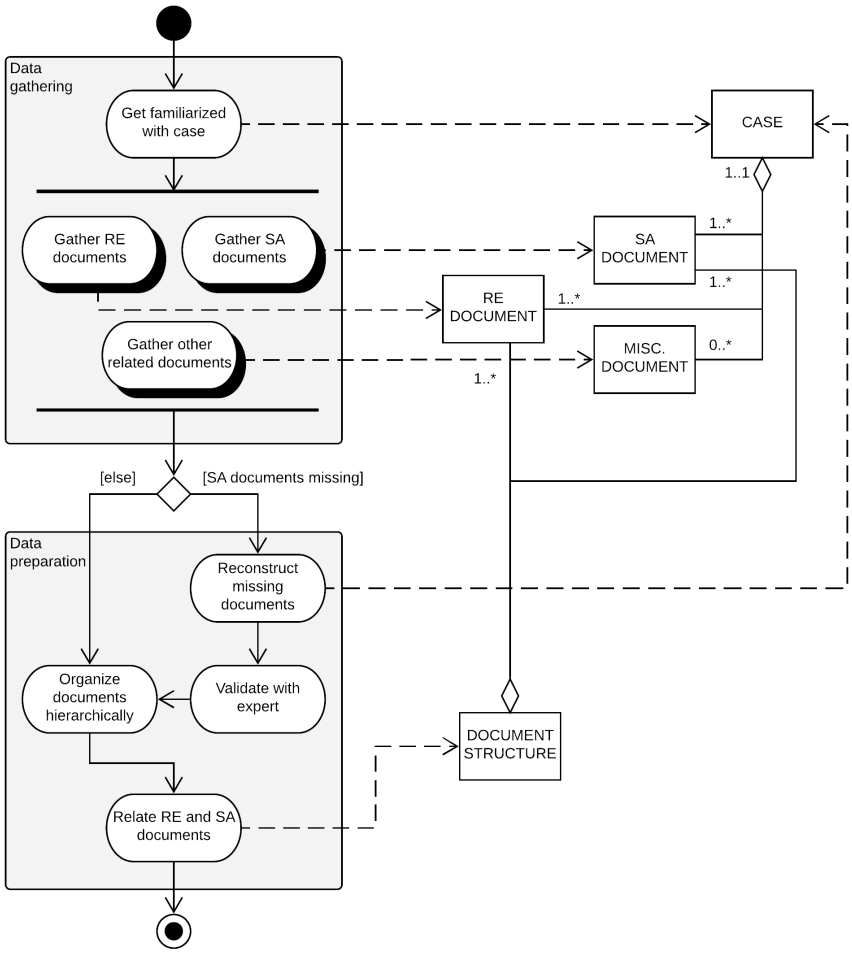

Figure 30: PDD depicting the data gathering and preparation process.

To start off, it is important to get familiarized with the case, since the linguistic terms can be context-specific. Familiarizing oneself, in this research, refers to reading the documentation and taking a look at the software system itself. Subsequently, the required documents are gathered, which include RE documents (such as Epics, USs, etc.), SA documents (FAMs, FADs and feature diagrams, among other architectural viewpoints and models) and additional documentation. The latter are used during analysis, should there be difficult to define and/or interpret linguistic terms.

It is possible to recover missing SA documents, such as FAMs, FADs and feature diagrams. It is, however, unacceptable to recreate all RE documents, since they are too dependent on subjectivity and interpretation as was mentioned earlier. Then again, it is unlikely that Epics as described in Chapter 3.4.3 are available. Instead themes or other kinds of US groupings are considered. If these are also unavailable, Epics are written using the USs as the only input. The SA models should be recovered using source code, the GUI and other architectural documentation. On the occasion that RE documents are used as input, the risk of introducing linguistic links is too high, given that RE concepts are utilized. To ensure the recovered SA documents are of sufficient quality, they will be validated by an expert.

Then, the acquired documents are structured hierarchically, meaning that the dependencies within the artifacts are taken into account. The hierarchical structure is determined starting from the lowest level of abstraction, so the USs and feature diagrams. Finally, the dependencies, or rather relationships, between artifacts are considered.

4.1.3 Case Study Execution and Analysis Approach

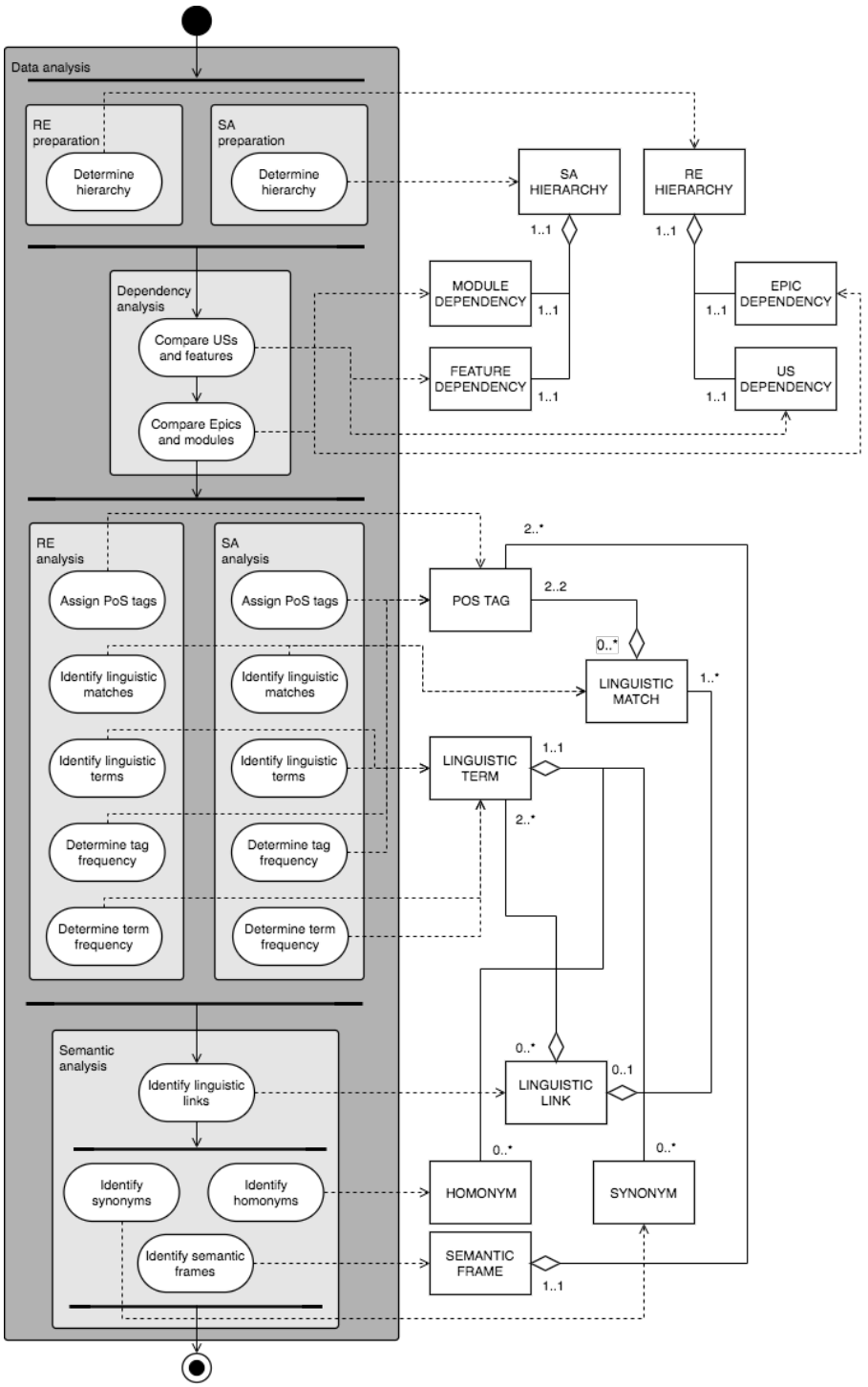

Figure 31: PDD presenting the steps in the analysis process.

During the preparation phase, the documents were hierarchically ordered. However, in this phase, the hierarchies are specified in more detail by organizing the individual Epics, USs, modules and features hierarchically. Subsequently, the USs and features and the Epics and modules are compared. This comparison mainly functions as a way to determine whether there are as many USs as features and as many Epics as modules. However, after the linguistic analysis is performed, these numbers may have to be re-adjusted.

Then, the artifacts of both domains are analyzed in more detail. All words in the artifacts will be assigned a PoS tag using the CoreNLP tool (Manning et al., 2014). To ensure context-

specific concepts and improperly formulated sentences are tagged correctly, all tags will be checked manually and changed if need be. Subsequently, linguistic terms are extracted by filtering out all the tags that do not indicate a verb or a noun (in whatever form). The PoS tags are also used to determine the frequency of each tag (excluding the tags that are specific to the various templates). In the final phase of the analysis, the extracted linguistic terms are compared to see if linguistic links can be identified between artifacts, as well as the positions of the matched words. In addition, potential synonyms, homonyms and semantic frames are identified. Not every word that is a potential homonym is considered, there are two exceptions. Firstly, potential homonyms that have an obvious difference in meaning based on their PoS tag, e.g.: 'store' as a noun and 'store' as a verb. Secondly, words that have more than one meaning, but are only used in one way, e.g.: 'fans' as in rotating blades for ventilation and 'fans' as in supporters, but only the former meaning of the word is ever used in the context.

The analyses per case are presented in individual case reports in Chapters 5.1 to 5.3. Subsequently, cross-case conclusions are drawn from the individual analyses (R. Yin, 2017). To determine whether two semantically similar words can actually be considered synonyms, WordNet is used. WordNet is a lexical database that groups nouns, verbs, adverbs and adjectives by synonyms (Miller, 1998). However, for terms that are not present in this database and to verify the accuracy, the Merriam-Webster dictionary is also used to identify synonyms. Examples of potentially missing terms are abbreviations, context-specific concepts and compound nouns. As stated by Yin, it is possible (and acceptable) to refine and/or change the steps in the process between cases.

All artifacts	**Epics & modules**	**USs & features**
Term frequency	Term frequency	Term frequency
Tag frequency	Tag frequency	Tag frequency
Semantic frames	Semantic frames	Semantic frames
Synonyms	Linguistic structure	Linguistic structure
Homonyms	Added terms	Added terms
	Modified terms	Modified terms
	Deleted terms	Deleted terms
	Aggregation structure	Cardinality

Table 8: Planned observations related to PoS tags and linguistic terms.

Note that these planned observations are not final and may be modified or adjusted during the case study and subsequent analysis phase.

The different concepts, their respective analyses and their purposes are as follows:

1. **Dependency of artifacts:** used to test the cardinalities between the two levels of abstraction, involves comparing the categorization of the USs in Epics to that of features in modules.
2. **PoS tags:** used to determine whether there is a linguistic relationship between RE and SA and, if so, what this relationship entails and how it can be used to benefit software development (arguably words can be used as well, but when using tags it is possible to generalize the results and identify (potential) patterns).
3. **Linguistic matches:** words two mapped artifact instances have in common, also determines how strong the match is, in case of a unique match a linguistic link can be established.
4. **Match positioning:** might give insight into the linguistic structure of artifacts, which can be used to support PoS tag patterns if there are any and useful for educational purposes (e.g. how to formulate the different artifacts).
5. **Linguistic terms:** used to identify synonyms, homonyms and semantic frames, of which the former two are used to determine whether the words can be considered linguistic matches.
6. **Semantic frames:** support the analysis of linguistic structures and can be used for educational purposes, such as what requirements the use of certain words put on the remainder of the US, can also provide additional, implicit information that might be useful in software development.

4.2 Case Study Execution

This chapter contains descriptions of the cases that are analyzed in Chapter 5. In addition, the process of executing the case study is described in terms of which documents were gathered, whether there were any difficulties in PoS tagging the different artifact and if and how artifacts were recovered and/or reconstructed. Three different systems were analyzed, an overview of their descriptions is provided in table 9. Due to the use of sensitive materials in case 3, no details may be disclosed and are therefore omitted in the remainder of this chapter.

ID	Description	Domain	Organization	Role interviewee
Case 1	Automated greenhouse software	Agriculture	Start-up	Product owner
Case 2	Research data DMS with web application	Research	University department	Developer
Case 3	Small web application	Undisclosed	Undisclosed	Product manager

Table 9: Overview of the three cases included in the case study.

4.2.1 Case Descriptions

The first case that is selected is that of greenhouse software. The software of this start-up was designed to fully automate greenhouses, while collaborating with existing processes, software and hardware systems. It should be noted that the artifacts of this case were created keeping an earlier version of the RE4SA model in mind. While the naming conventions for the different artifacts were adhered to, this case did not take into account linguistic links in any way. The case documents consist of:

1. Jobs (8)
2. Epics (31)
3. USs (96)
4. FAM (1)
5. Context diagram (1)
6. Scenario overlays (3)
7. Feature diagram (partial, 1 module)

The case includes an extensive list of Jobs, Epics and USs. Moreover, a FAM that represents the whole system is included. Unfortunately, only one feature diagram is available, meaning that the features for one module are modeled. In an attempt to discover more features, the owner of the documents was asked whether more feature names, descriptions or diagrams can still be provided, which was unfortunately not the case. Despite the lack of feature diagrams, this case was considered valuable, since it includes an extensive set of Epics, formulated according to Lucassen's work. Epics that adhere to the template are few and far between, so this case provides a rare opportunity to analyze them. Besides, the USs can still be compared to the one feature diagram that is available and can be analyzed individually.

The second case is an archiving system for research data for a university, YODA. Not only can data be stored in an immutable way for replication purposes, it can also be used to share data within a research team. In short, it is a data management system as well as an archive. The case documents that were delivered are:

1. Epics (32)
2. USs (300+)
3. Software design & documentation (available at: `https://utrechtuniversity.github.io/yoda-docs/`)
4. Code (mostly) available on GitHub: `https://github.com/UtrechtUniversity/`

Due to the size and complexity of the system, it was advised by one of the developers to focus on a part of the system. Especially the data storage, which includes communication with various remote and distributed databases and servers, is a complex and difficult to comprehend part of the system. Instead, a subset of the system will be used as a means to scope the case. More specifically, the client/user side of the system will be investigated. This includes part of the data management functionality, as well as interface considerations and authorization among others. This scope provides one main benefit, namely, being able to interact with the system to clear up any

uncertainties and to answer possible questions. In addition, the web application can be utilized during architecture recovery. Documentation is provided in English, so there are no translation concerns.

Among the USs, other types of stories can be found. One type is aptly called 'ideas', which is also available as an Epic/theme. More notable are the so-called 'enabler stories'. These stories are almost exclusively written from the perspective of developers and/or testers. According to Lo and Chen enabler stories are "*technical stories enabling the system to fulfill business user stories or achieve system attributes*" (Lo & Chen, 2017). In short, they can describe constraints and quality requirements. Interestingly, this is the only result for the "enabler story requirements engineering" search query on Google Scholar from 2015 to now. Given the fact that this research is focused on USs, enabler stories are excluded from analysis.

Details of the third case may not be disclosed. Based on the recommendations of the product manager, only the parts of the system (web application) that were covered in the USs were included in the scope. The functional architecture, so both FAM and feature diagram had to be recovered.

4.2.2 Case Execution Process

In the artifacts for all cases, minor spelling errors or typos were corrected to prevent erroneous PoS-tagging by the CoreNLP tool. For example, in some stories 'I' was spelled in lowercase, which lead the tool to tag this word as a foreign word (tag 'FW'), while it is in fact a personal pronoun (tag 'PRP'). Similarly, to avoid confusion and incorrect tags, all contractions have been removed and written in full instead (for example 'don't') for case 1. In addition, some words received the wrong tag. 'Changes' was considered a plural noun, while it was in fact a verb (for instance 'When [a noun] changes (...)'). The word 'set' was tagged as a verb, but since the word before it was 'data', it was actually a noun (as in 'the set'). Likewise, the word 'light' was tagged as an adjective (tag 'JJ'), while in combination with the word 'manager', it is also a noun. Finally, a minor difficulty was the word 'data', which was often tagged as a plural noun, but in some cases where it was preceded by 'a' as a singular noun. Like in the previous two issues, the determiner 'a' was actually related to the noun following 'data' (for instance 'a data manager'), so the plural noun tag was used instead.

In the USs of the second case, extra information was sometimes added in parentheses. This extra information is included in the analysis if it provides an additional explanation, for instance when examples are stated. However, in one case (YDA-1605) a word was defined by adding another word in parentheses, since this did not make sense grammatically, this word was not taken into account during analysis. For the sake of consistency between cases, the word 'data' is considered plural (this also includes the word 'metadata'). In multiple USs, a forward slash is present. At first these were tagged as a symbol, but to be able to extract some meaning from them, they were later considered 'CC' tags, since they can indicate 'and', 'or' or both.

During USs selection of the second case, it became apparent that perhaps not all USs can be linked to features. However, the question is not solely how to solve this problem, but more importantly what its source is. In feature diagrams, as discussed in Chapter 3.2.2, features are usually considered interactive functionality between the system and the user, e.g. buttons that can be clicked. In the USs of the second case, there are requirements for features that are not, at first glance, functional requirements, nor are they quality requirements or constraints. They either describe some extended functionality for an existing feature or functionality that is not directly related to user interaction. The following US is an example of this:

"*As a data manager, I want that revisions of files are removed from Yoda according to the retention policy, so that I can manage the storage costs.*"

Arguably, it can be considered a quality requirement, but it does not address any of the quality perspectives of SA, as described by Rozanski and Woods (Rozanski & Woods, 2011), and the focus is not on the quality requirement aspect, since it is included in the benefit constituent. Secondly, it does not define a constraint, nor a functional requirement in the sense that it describes user interaction. Essentially, it more accurately illustrates something that 'just happens', or rather system behavior or a system response. One can argue that this is a bad US, since it does not describe a requirement from a user perspective. However, how else would functionality like this be captured in requirements or otherwise included in the system? Or should this be considered 'developmental freedom'? In addition, perhaps the purpose or intent of a feature can be twofold. In

that case, a feature with more than one functionality could be called a complex feature. Unfortunately, this brings us back to an earlier question, should this be considered extending functionality of another feature (and thus another US) or should this be considered functionality an sich and be described by its very own US? If the former approach is selected, that means that, technically, a US does not describe a requirement for one feature anymore. Were the USs to be extended, then the purpose of USs being a technique to describe piecemeal requirements is defeated. To make matters even more complicated, if extended or complex functionality is introduced, how can this be represented in feature diagrams? The RE4SA model, although not proven, assumes that there is a one-to-one relationship between the formulation of USs and the design or description of features. Does it benefit software development to enforce these cardinalities and if so, how can they be enforced or prescribed? On the other hand, the question whether there exists (or should exist) a one-to-one relationship between features and functionality remains.

In an attempt to mitigate these issues, the concern-based requirements taxonomy as presented by Glinz was applied to the USs. This taxonomy distinguishes four types of system requirements:

1. Functionality and behavior: functions, data, stimuli, reactions and behavior.
2. Time and space bounds: timing, speed, volume and throughput.
3. "-ilities": reliability, usability, security, availability, portability and maintainability (non-exhaustive list).
4. Constraints: physical, legal, cultural, environmental, design & implementation and interface (non-exhaustive list).

The first type is also referred to as functional requirements, followed by performance requirements, specific quality requirements and finally constraints (Glinz, 2007). Applying the taxonomy resulted in the removal of six quality requirements and one constraint. However, it can be reasonably assumed that not all functional requirements can be translated into a functional architecture element. The functional view is concerned with modeling the internal structure of a system and its communication with external systems (Rozanski & Woods, 2011). The information and concurrency views, on the other hand, might be more suitable to model interactions and behavior. USs that describe behavioral requirements will be included to examine if and how they can be mapped to the functional architecture.

Since the Epics were more akin to themes, they had to be formulated into Epics according to the template and then validated with an expert, as specified in figure 30. A selection of Epics, not adhering to template (ones that involved the client-side of the system), were transformed into Epics, adhering to the template, and discussed with a key stakeholder (developer to be specific) of the system. They were formulated based on RE documentation only, so the existing Epic descriptions and their corresponding USs.

First and foremost, the developer advised to exclude an additional Epic (EPOS-MSL), since technically it was not part of the system, but rather part of the IT infrastructure of the university as a whole. Furthermore, four motivation constituents had to be changed slightly to increase accuracy and completeness. He also explained some of the Epics and their goals in more detail and while they did not require rewriting, the information is included here:

1. YDA-321 Manage vault: while considered a separate Epic, there is some overlap with the Vault Epic (YDA-96).
2. YDA-2208 Metadataschema: mainly concerned with metadata settings for communities within the system (such as default metadata forms), in the future it will also be changed for transforming existing schemas into other schemas, so different versions of the metadata.
3. YDA-2694 Export: not meant for migrating data, but rather for archiving data when you do not want to store it in the system any longer.

Another important remark is that data is considered just data or files or folders when stored inside the research module and is called a datapackage as soon as it enters the vault.

In similar fashion, a FAM was recovered. Only architecture documentation, the system itself and source code was used to model the functional architecture. A first version was created and discussed with the same developer. Some sub-modules were changed into modules and vice versa and some missing information flows were added. The names of the modules and flows were also discussed, but no modifications were required. There was a necessity to model two external systems for completeness. While not part of the internal structure of the system in question, these external

systems were deemed important for providing an accurate representation of the system as well as explaining its functionalities. After changes were made, the second (improved) version was sent back for a follow up validation.

The feature diagrams were recovered using the architecture documentation, the GUI of the application and the source code. For the majority of the features, the GUI was used. Each interactive element (buttons, sliders, textboxes, etc.) is considered a feature. Then, if an interaction leads to more interactions (options), it is modeled as a composite feature. If no other interactions succeed the interaction, this is considered an atomic feature. In some feature diagrams the features are not only grouped based on composite features and modules, but also by headers in the GUI. This was only done in situations in which there was an exceptionally large number of features related to one module or composite feature. These headers do not have any effect on the architecture, but are utilized to improve legibility. An example of how feature diagrams are recovered utilizing the GUI as input is shown in figure 32.

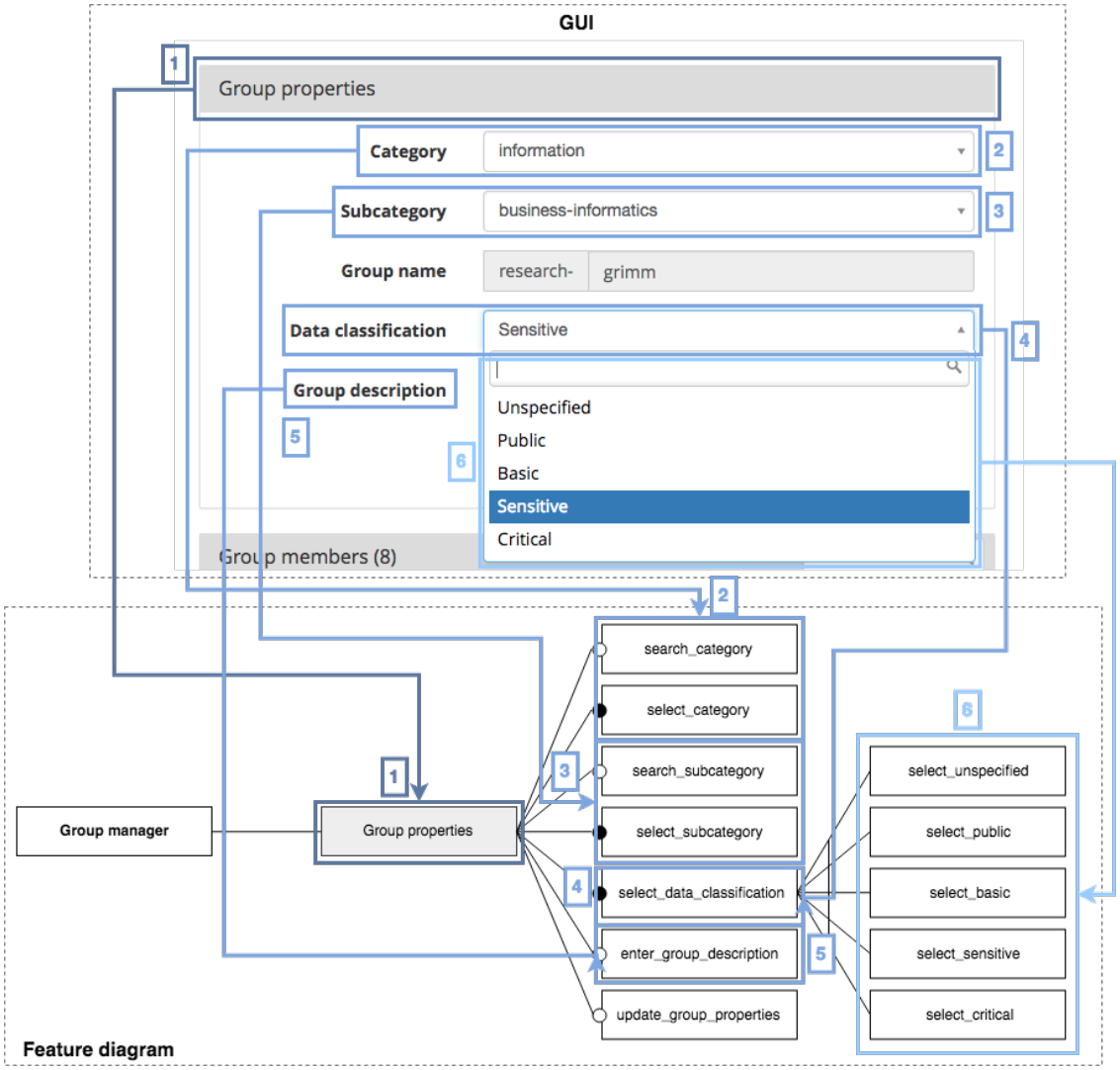

Figure 32: Example of how the GUI elements are utilized to recover feature diagrams.

A screenshot of part of the Group Manager page is shown at the top and the corresponding part of the Group Manager module feature diagram is included at the bottom. The three layer structure in the GUI is represented as a three layer depth in the feature diagram and the following architectural elements were extracted from the GUI:

1. Group Manager: the starting point of the feature diagram is the module in which the feature are positioned. In this case this is the 'Group manager' (shown in bold), since these features

are accessible through the 'Group Manager' page.

2. Group Properties: as was mentioned previously, headers from the GUI are included to improve legibility of the feature diagrams (or in other cases allow for an alternative decomposition). As they do not provide any functionality, they are not considered features. Headers are shown as gray boxes.
3. Category & Subcategory: the category element in the GUI consists of searching for a category and selecting a category and is accessed by clicking on the arrow.
4. Data Classification: composite feature that is refined into five options (atomic features).
5. Group Description: text box, currently invisible due to the unfolded data classification menu.
6. Classification Selection: the composite feature 'data classification' is refined into five options (atomic features): unspecified, public, basic, sensitive and critical.

Note that the update group properties feature is not visible in the GUI, since it is hidden behind the data classification menu. Features are considered mandatory if they are required to carry out a certain task. In the example, when a user creates a group, they must select a category, subcategory and data classification. Other examples of required and thus mandatory features are when entering personal details, in such a situation the required elements are indicated by an asterisk.

In addition, the features were grouped based on the modules as specified in the FAM. The quality of the feature diagrams in terms of completeness is sufficient, since all GUI features are included and the diagrams were validated by a Yoda developer. However, the accuracy in terms of presentation is questionable. The feature diagram syntax does not allow for 'conditional' features. Meaning that features are only available when certain other conditions are met. For instance, in terms of service agreements, oftentimes a box needs to be checked prior to accepting the terms. The lack of conditionality expressions mainly has consequences for the depth of branches in feature diagrams. By including the order of activities in the feature diagram, some features are placed at a far greater depth than they are given the GUI. The depth of a feature diagram refers to how many steps or actions need to be performed in order to make use of a feature or carry out a specific task. For instance, in the Yoda case, to get the option to submit a folder to the vault, there first needs to be a folder, which needs to be selected and subsequently locked. While the vault submission feature is placed at the same depth as the locking of the folder (so after a folder is selected). Since the structural accuracy is not of importance here, the issue is ignored here, but included as part of future research directions. For the sake of brevity, the feature diagrams and their descriptions are included in Appendix B.

5 Analysis

In this chapter, the performed steps in the analysis process (as described in Chapter 4.1.3) are discussed in more detail. Throughout this chapter, qualitative findings included in the results chapter (Chapter 6) are indicated by an ID and presented in bold. Table 10 provides an overview of the Penn Treebank PoS tagset (M. Marcus et al., 1993) that are used throughout the linguistic analysis of the artifacts.

Tag	Description	Tag	Description
CC	Coordinating conjuction	TO	*to*
CD	Cardinal number	UH	Interjection
DT	Determiner	VB	Verb, base form
EX	Existential *there*	VBD	Verb, past tense
FW	Foreign word	VBG	Verb, gerund/present participle
IN	Preposition/subord. conjuction	VBN	Verb, past participle
JJ	Adjective	VBP	Verb, non-3rd ps. sing. present
JJR	Adjective, comparative	VBZ	Verb, 3rd ps. sing. present
JJS	Adjective, superlative	WDT	*wh*-determiner
LS	List item marker	WP	*wh*-pronoun
MD	Modal	WP$	Possessive *wh*-pronoun
NN	Noun, singular or mass	WRB	*wh*-adverb
NNS	Noun, plural	#	Pound sign
NNP	Proper noun, singular	$	Dollar sign
NNPS	Proper noun, plural	.	Sentence-final punctuation
PDT	Predeterminer	,	Comma
POS	Possessive ending	:	Colon, semi-colon
PRP	Personal pronoun	(	Left bracket character
PP$	Possessive pronoun	)	Right bracket character
RB	Adverb	"	Straight double quote
RBR	Adverb, comparative	‘	Left open single quote
RBS	Adverb, superlative	“	Left open double quote
RP	Particle	’	Right close single quote
SYM	Symbol (mathematical or scientific)	”	Right close double quote

Table 10: Overview of the Penn Treebank PoS tagset used for linguistic analysis.

In some cases, while PoS tagging, a word could be interpreted in more than one way, resulting in more than one appropriate tag. For example, when considering verbs in the past participle preceding a noun, they should actually be tagged as adjectives (JJ). Not all such cases were tagged correctly and had to be fixed manually. As another example, the word ‘much’ can either be an adjective (JJ) or an adverb (RB), depending on the sentence structure. However, these were tagged correctly and few occurrences of the word could be observed. Finally, when looking at the matches and subsequent links between RE artifacts and SA artifacts, mostly verbs and nouns are taken into account. In case of the former, modals (MD) were not regarded as verbs. The reason for this is twofold: firstly, verb tags exist and they are purposely not included in those tags. Secondly, the sentences or phrases that include modals can be rephrased in such a way that they can be omitted without changing the meaning. Earlier, it was stated that only nouns and verbs would be considered during the linguistic analyses. However, it became apparent that additional words (and thus tags) can be necessary in order to establish a linguistic link. Therefore, all tags are considered during the linguistic matching and linking activities.

Linguistic matches are analyzed to determine how accurate or precise they are. Generally, four types of matches are distinguished:

1. Precise: the match consists of two words that are exactly the same. E.g.: ‘temperature’ (NN) and ‘temperature’ (NN).
2. Synonym: the match consists of two synonyms. E.g.: ‘moisture’ (NN) and ‘humidity’ (NN).
3. Near-precise: the match is nearly identical (e.g. one compound noun and two nouns or singular and plural). E.g.: ‘sprays’ (NNS) and ‘spray’ (NN).

4. Partial: difference in PoS tag (such as a verb and a noun), but the root of the word is the same. E.g.: 'pump' (VB) and 'pump' (NN).

After analyzing the first case, it became apparent that determining the frequency of terms is not of any use to the objectives of this research. The same is true for added, modified and/or deleted terms, since these are not of any influence on the current linguistic structure. On the other hand, an observation that was missing is the position of the words that can be matched. This is especially relevant to determine how artifact names can be generated and how matches could possibly established automatically.

5.1 Case 1

As was stated in the case description, both Epics and USs were provided in this case, as well as a functional architecture. Moreover, the Epics and USs were already hierarchically ordered, stating which USs are related to which Epic. The same is true for the functional architecture, the modules were already divided over the different systems and the parent module of the feature diagram was included. Considering the dependencies, it soon became apparent that there exist as many Epics as modules. Unfortunately, since only one feature diagram was included, little can be said about the dependency between USs and features. In case of the former, it is not completely self-evident which Epics are related to which modules. The provided figures can be found in Appendix A. In the references a link to the thesis can be found, which includes the stories. An interesting note is that the roles of the USs in this case are actually system components. So, rather than using specific stakeholders in roles, different parts of the system that communicate and interact with each other are used instead, which makes sense since the requirements were written for an autonomous system.

5.1.1 Dependency Analysis

Fortunately, the Epics, modules, USs and features are well-documented, making it easy to see which USs are part of which Epic and which features are contained in which module. From the 31 Epics, all have been modeled as a module. However, only one feature diagram is included, as was mentioned previously, so only a subset of the features and USs can be analyzed in terms of dependency. The feature diagram that was included (depicted in Appendix A, figure 38), consists of ten features. Arguably, it contains six features if the one-to-one composite-atomic features are not taken into account and five if only the lowest level features are included. In any case, the Epic that is related to the module of which the feature diagram is created contains only four USs, so **there is no one-to-one relationship (1.1)**. Based on the linguistic matches, the USs can be linked to the ten features. Oddly, two USs resulted in four features, one US in two features and the final US in no features. These discrepancies can be explained by three possible issues based on the literature study:

1. Poorly written USs
2. Poorly modeled features
3. Same action is desired by more than one role

Keep in mind that these possibilities are not mutually exclusive. Starting with US 2-1:1, this one is related to four features. One of which, 'verify begin state' cannot be matched to any of the words in the US, but the relationship is inferred from a composite feature ('process light instructions'). Other than that, it seems as if the US contains two actions rather than one, resulting in more than one feature. Subsequently, one of those features required additional functionality, resulting in another feature. According to the QUS framework, the US should describe a requirement for exactly one feature, meaning that the US should have been split into two.

US 2-1:2 is related to two features, one of which can be related through a linguistic match, the other is the child feature to that parent feature. In this case it seems that, again, a feature required an additional feature to work. So, this could mean that a US describing this feature was missing.

US 2-1:3 is related to four features, which can all be matched linguistically. It seems that the US describes two features again, one for monitoring screens and one for monitoring lamps, which could have been split into two separate USs. They can even share the same benefit. Like before,

additional features were added that are not described by a US. Both 'assigning a screen state' and 'assigning a light state' could have been formulated in USs. So it seems as if there are USs missing.

Finally, US 2-1:4 cannot be related to any of the features. So either it describes a feature that is present in another module or it cannot be contained in a feature. When reading the US, its description of a potential feature is quite vague and seems to describe functionality on a higher-level, almost as if it could have been an Epic. However, due to the lack of other feature diagrams it is difficult to determine what the purpose of this US was.

Coming back to the possible issues, it can be assumed that the one-to-many relationship between three USs and their features can be explained by poorly written USs, referring to the fact that they contain a requirement for more than one feature, and missing USs.

To gain more insight into the relationship between USs and features, the USs that are not related to a feature are analyzed as well. It is possible, based on their formulation, to infer whether they would refer (or seem to refer) to one or more features. An indication of whether a US describes a requirement for more than one feature is the word 'and'. In extension, the word 'or' is considered as well, meaning that USs that contain any 'CC' tags are examined. The 'CC' tag, according to table 15 was observed 24 times. The benefit constituent is not mandatory and therefore does not provide essential requirement information for a feature, which means fourteen USs are left. The features that can be extracted from USs are, however, subject to interpretation. Consider the following US as an example:

"*As a temperature manager, I want to open or close the overhead windows, so that hot air can exit the greenhouse.*"

Based on this US, it is possible to identify two features, namely opening and closing the overhead windows. On the other hand, the US can be rewritten as follows:

"*As a temperature manager, I want to determine the state of the overhead windows, so that hot air can exit the greenhouse.*"

In this case, the 'state' can refer to either opening or closing the windows. This results in another question, should this state be refined into opening or closing, or left as-is? Another approach would be to split the former US into one that describes opening the windows and the other that describes closing the windows, again resulting into two features. To make matters more complicated, developers and architects can make the decision to combine opening and closing into the state of the windows, as described in the second example, leading to one feature. So there are four possibilities:

1. One US resulting in one feature
2. One US resulting in two features
3. Two USs resulting in one feature
4. Two USs resulting in two features

Of the fourteen 'CC' tags, three out of ten times 'and' could cause confusion when translating the described requirement into one or more features. For 'or' this was the case three out of four times. The non-problematic cases were USs in which 'and' or 'or' were used to describe a combination of resources.

5.1.2 Epics & Modules

PoS-tagging the module names shows that they solely consist of nouns, with the exception of two modules that also contained an adjective. Seventeen modules consisted of two words, while the remaining fourteen consisted of three. The terms that are part of the module name template are the last noun or the last compound noun. Table 11 provides an overview of the PoS tags that could be identified and their frequency of occurrence per Epic and module constituent.

PoS tag	Situation	Motivation	Expected outcome	Total	Module name	Module template	Total
CC	1	4	1	6	-	-	-
DT	16	24	51	91	-	-	-
EX	1	-	-	1	-	-	-
IN	2	5	27	34	-	-	-
JJ	8	10	21	39	2	-	2
JJR	2	-	-	2	-	-	-
MD	-	-	4	4	-	-	-
NN	49	33	45	127	38	35	73
NNS	3	14	23	40	1	-	1
NNP	-	-	1	1	-	-	-
PRP	3	3	1	7	-	-	-
PP$	-	-	1	1	-	-	-
RB	-	4	2	6	-	-	-
RP	-	13	2	15	-	-	-
TO	3	4	2	9	-	-	-
VB	3	35	12	50	-	-	-
VBG	10	-	1	11	-	-	-
VBN	6	2	6	14	-	-	-
VBP	4	0	3	7	-	-	-
VBZ	17	-	17	34	-	-	-
WRB	-	3	1	4	-	-	-
,	-	2	-	2	-	-	-
.	-	-	1	1	-	-	-

Table 11: Overview of PoS tags in Epics and modules of case 1.

In Chapter 3.6.1 the mandatory and optional linguistic structures were discussed. For Epics these include a subject, action verb and direct object, which can be considered a noun, a verb and a noun respectively. Modules consist of at least a noun and possibly another noun and a verb. In this case, **mostly nouns are used in module names (1.2)**. However, in case of Epics, these words are only used to formulate the motivation constituent. For the other two constituents no mandatory or optional words have been defined, even though the situation constituent is of importance when creating modules (explained in more detail later in this chapter). Given basic sentence structures it is reasonable to assume that both constituents require nouns and verbs. Furthermore, looking at the Epics in the case, the situation constituent oftentimes only consists of verbs and nouns. The expected outcome constituent takes a slightly different approach. Again, verbs and nouns are needed to formulate a comprehensible phrase. In addition, prepositions are often included as well as adjectives. The module names do not adhere to the template rules in all cases, since twice an adjective is included, while only nouns and verbs should be used. However, when looking at the linguistic links in table 12, an adjective can mean the difference between a unique or duplicate match. Since this adjective was matched with a preposition in the Epic, the nouns in the situation constituent should be allowed to include prepositions to facilitate such matching. In addition, one can argue that information is lost when comparing the PoS tags observed in Epics and those observed in modules. For instance, personal pronouns can indicate that a noun belongs to someone, or in this case something. Similarly, any activities performed on objects are specified by verbs and thus not explicitly included in module names.

Two modules contained template terms that corresponded with words in the Epic. In case of the 'harvest predictor' module, the term 'predictor' can be related to the term 'prediction' in the Epic, which is not an identical match. Similarly, the term 'planner' in the 'action planner' module can be linked to the term 'plan' in the Epic, which is, again, not a precise match. Out of the 31 modules and Epics, only one pair did not contain any corresponding terms. The others have been analyzed to see whether there was an exact match and in which part or parts of the Epic the matching term could be observed. Excluding the words related to the module name template, 21 module names contained only one word and the other ten included two. Out of these ten names with two words eight can be considered compound nouns and the other two contain an adjective that is related to the noun. The results of the linguistic matching are shown in table 12.

To illustrate how the table should be read, consider the following example of an Epic (3-3), in which the parts of the template are in brackets:

"*[When] receiving a timer[, I want to] turn on the* ***window sprays****[, so that] the greenhouse cools down*"

It has been mapped to the following module, of which the template word is put in brackets:

"***window spray*** *[control]*"

The words in bold can be matched: the words "window" and "sprays" in the Epic and the words "window" and "spray" in the module. Both words, including their PoS tag are then included in the table. In addition, since the combination of these words can be seen as a compound noun, the 'c' in parentheses is noted to suggest that this match is only unique if the compound noun is considered. Finally, the match is labelled as 'near-precise', since they do not use the exact same words. The Epic uses a plural, while the module uses singular.

ID	Epic term(s)	PoS tag	Module term(s)	PoS tag	Match	Unique?
1-1	data	NNS	data	NNS	Precise	Yes
1-2	yield	NN	harvest	NN	Synonym	Yes
1-3	action	NN	action	NN	Precise	Yes
1-4	-	-	-	-	-	-
2-1	(light, lighting)	(NN, NN)	light	NN	(Precise, partial)	No
2-2	screen	NN	screen	NN	Precise	Yes
2-3	lighting	NN	light	NN	Partial	No
3-1	temperature	NN	temperature	NN	Precise	No
3-2	heating	NN	heating	NN	Precise	Yes
3-3	window, sprays	NN, NNS	window, spray	NN, NN	Near-precise	Yes (c)
3-4	windows	NNS	window	NN	Near-precise	Yes
4-1	humidity	NN	humidity	NN	Precise	No
4-2	(moisture; humidity), pump	(NN; NN), NN(S)	moisture, pump	NN, NN	(Precise; synonym), precise	Yes (c)
4-3	ventilation	NN	ventilation	NN	Precise	Yes
5-1	CO2	NN	CO2	NN	Precise	No
5-2	CO2, pump	NN, VB/NN	CO2, pump	NN, NN	Precise (partial)	Yes (c)
6-1	plant	NN	plant	NN	Precise	Yes
6-2	growth, model	NN, NN	growth, model	NN, NN	Precise	No
6-3	growth, model	NN, NN	growth, model	NN, NN	Precise	No
7-1	wind	NN	wind	NN	Precise	Yes
7-2	pressure	NN	pressure	NN	Precise	Yes
7-3	sunshine	NN	radiation	NN	Synonym	Yes
7-4	outside, temperature	IN, NN	external, temperature	JJ, NN	Synonym, precise	Yes (c)
7-5	cloud	NN	cloud	NN	Precise	Yes
8-1	evaporation	NN	evaporation	NN	Precise	Yes
8-2	soil, moisture	NN, NN	soil, moisture	NN, NN	Precise	Yes (c)
8-3	humidity	NN	humidity	NN	Precise	No
8-4	CO2	NN	CO2	NN	Precise	No
8-5	temperature	NN	temperature	NN	Precise	No
8-6	light	NN	light	NN	Precise	No
8-7	light, absorption	NN, NN	light, absorption	NN, NN	Precise	Yes (c)

Table 12: Linguistic matches between Epics and modules in case 1.

For clarity, the term 'pump' in Epic 4-2 has both the NN and NNS tag, since it is present more than once, as both a singular and plural noun. Similarly, in 5-2, the term 'pump' is included as both a verb and a noun. Matches are considered precise if the terms in the Epic and the module are exactly the same, with the exception of plurals of nouns, which are considered near-precise matches. For example, in Epic 3-3, the term 'sprays' is plural, while in the module name its singular form is present. Matches are deemed to be partial if the root word is the same, but the terms used are not (see 2-3 for an example). Some matches are based on synonyms, for example in case of the terms 'sunshine' and 'radiation' in 7-3, these words are considered synonyms according to the thesaurus. Finally, some matches are unique, while others are not. Matches are considered unique when an Epic can be mapped to only one module, exclusively based on the matched terms. For nineteen pairs, the match is unique. For six of those, the nouns have to be considered compound nouns in order to result in a unique match (indicated by a 'c' in parentheses). Finally, there exists a possibility that a match is not unique when solely considering the terms. However, if such a match is contained within another match, for example in a compound noun, it can still be considered unique if there are no other similar matches. To illustrate: if only one Epic and module mentioned

the term 'CO2', it can be considered unique, since it is the only match left after eliminating the compound noun match of 'CO2 pump'. So, a **process of elimination can lead to more unique matches and subsequent linguistic links (1.3)**. While this did not occur in table 12, it should be taken into consideration in future analyses. While adjectives are not included in the linguistic structure of modules, they can be used to establish links. For instance, in case of 7-4, this **match can only be considered unique when taking the adjective into account (1.4)**.

Table 13 shows in which constituents of the Epic template the matches can be found. It should be noted that compound nouns are considered one match and not two. Only one Epic contained two matches in one constituent (5-2), since the word 'pumps' was also present as a verb. The term 'outside temperature' was also considered a compound noun.

Epic constituent	*Situation*	*Motivation*	*Expected outcome*
Number of matches	23	9	10

Table 13: Number of matches divided over the three constituents of the Epic template in case 1.

As is apparent, **the matches can most often be found in the first constituent of the Epic (1.5)**. This, however, can also introduce misunderstandings, since some situation constituents refer to other modules to explain dependencies and information flows. For instance, Epic 1-3 refers to 'yield prediction' in the situation constituent, which seems to be related to the 'harvest predictor' module, while it is in fact mapped to the 'action planner' module. This is, oddly, in conflict with what Blessinga mentioned in his master thesis. He stated that the motivation constituent (the second part) should be functional, so that it can be mapped to a module (Blessinga, 2018). However, this constituent could be mapped to a module only nine times. The expected outcome constituent could be mapped to a module ten times using linguistic matches, out of which only three matches would lead to a unique mapping to a module. The other seven Epics with matches could be mapped to more than one module.

Out of the 23 matches within the first constituent of the Epic, the matched term was the only noun or part of the only compound noun in nineteen instances. When only a compound noun was present, the noun that was matched with the module name was the first noun in the compound noun all but once (in Epic 7-2 it was the last noun, but including solely the first noun or the full compound noun would change little for the module name). In two Epics more than one (compound) noun could be identified and twice the matched noun was part of the last compound noun. Only once could the matched noun be identified as the first noun in the constituent. Oddly enough, out of the seventeen Epics that contain compound nouns (not including the Epics that include an adjective or preposition), nine of them lose part of the compound noun in the module name. While the meaning is not lost and no ambiguity is introduced, **the module names become slightly less detailed (1.6)**. Furthermore, if more modules were to be introduced in the future the names might lead to confusion or erroneous links. An example of a less specific module name is the 'humidity monitor' module, which does not disclose what kind of humidity is meant or where it is being monitored. In the Epic however, this is formulated as 'greenhouse humidity', which is more specific. Two other examples are 'pressure monitor' and 'wind monitor', which are described in the Epics as 'air pressure' and 'wind direction' respectively. The opposite is true once, for an Epic in which merely the word 'plant' is mentioned, while the module defines a 'plant type'. Module names are generally more concise than Epics, but in this case it is not a rare occurrence that a module consists of three words, which means that including the full compound noun cannot have been a decision for the sake of brevity. Moreover, there is sufficient space for longer names in the architectural elements of the FAM.

Table 14 shows the positions of the matched terms per Epic constituent. In this table, only unique matches (so linguistic links) were included.

	Situation constituent		**Motivation constituent**		**Outcome constituent**	
	Frequency	*Percentage*	*Frequency*	*Percentage*	*Frequency*	*Percentage*
First noun	8	66,7	5	55,6	2	100
First c. noun	3*	25	3	33,3	-	-
Second noun	-	-	1	11,1	-	-
First noun if ext. to c.	1	8,3	-	-	-	-

Table 14: Positions of the linguistic matches in Epics in case 1.

In this table 'c.' stands for 'compound'. In the situation constituent, nearly all linguistic matches are the first noun or first compound noun. in one Epic, the first compound noun consisted of a preposition and a noun. Moreover, the one match that was not in the position of the first noun can be slightly altered. By extending the matching noun to a compound noun, without changing the meaning: from 'pressure' to 'air pressure'. If that were the case, 100% of the linguistic links in the first constituent are the first (compound) noun. However, this is possible in just barely the majority of all the Epics, since there are a total of 23 matches in the situation constituents. This is quite different in case of the motivation constituent. Since eight out of nine of the unique linguistic matches are positioned as the first noun or first compound noun. Finally, only two of the matches in the expected outcome constituent were unique, but both were the first noun.

5.1.3 USs & Features

Table 15 shows which PoS tags could be identified in each US constituent and the USs in general, as well as the tags for the few features that were defined. Only tags that could be found in at least one constituent or feature have been included in the table. In total 96 USs, five composite and five atomic features were analyzed.

PoS tag	Role	Action	Benefit	Total	Feature name
CC	-	14	10	24	-
CD	-	5	-	5	-
DT	-	66	66	132	-
IN	-	59	39	98	-
JJ	4	47	52	103	1
JJR	-	2	-	2	-
MD	-	4	49	53	-
NN	210	158	125	491	15
NNS	-	71	52	123	4
NNP	-	2	-	2	-
PRP	-	7	43	50	-
PP$	-	3	9	12	-
RB	-	9	26	35	-
RP	-	10	4	14	-
TO	-	20	5	31	4
VB	-	112	65	177	9
VBG	-	3	4	7	-
VBN	-	20	41	61	-
VBP	-	9	26	35	-
VBZ	-	10	32	42	-
WP	-	5	2	7	-
WRB	-	12	8	20	-
,	-	1	1	2	-
.	-	-	1	1	-

Table 15: Overview of PoS tags in USs and features in case 1.

It is immediately apparent that the role constituent nearly exclusively consists of singular nouns. In four out of 96 cases, an adjective was included as well. Also, since there are 96 USs and after checking the USs, all USs included a compound noun as a role, mostly with two nouns and sometimes three. The action constituent is less straightforward to formulate, as many different tags could be observed. When looking at which words occur in half the USs, the nouns, verbs, determiners, prepositions and adjectives are left (even though the latter could be observed in one less than half). The use of verbs (base form verbs to be precise) is easy to explain since the action constituent template is "I want to", which needs to be followed by a verb. Furthermore, the verb should refer to something, which explains the necessity of nouns. In some cases the 'minimal' action is extended by another verb phrase containing more nouns, as well as prepositions and adjectives. Finally, the benefit constituent is hard to pin down. A wide range of PoS tags could be identified, with varying degrees of frequency. Again, determiners are popular, since they accompany nouns.

In addition, prepositions and adjectives are also frequent occurrences. Taking into account both singular and plural nouns and any form of a verb, a US contains on average six nouns (rounded down) and three verbs (rounded down). **Features consist of nearly exclusively nouns and verbs (1.7)**, which is in agreement with the linguistic structure for feature names.

Looking at the single feature diagram that was included, some linguistic comparisons can be made between the USs and feature names. Fortunately, due to the hierarchy of Epics and USs, it is self-explanatory which USs are related to the feature diagram. Strangely, however, some composite features include only one feature, defeating the purpose of the former feature being composite. Table 16 provides an overview of the linguistic matches found in the USs and their corresponding features. The first three digits in the feature ID indicate to which US they are related. As an example, consider the following US (2-1:1), again, the parts in the template are put in brackets:

"*[As a/* ***light*** *manager[, I want to/ use a* ***light instruction*** *to determine if a zone needs* ***light*** *or shadow[, so that/ the correct* ***lighting*** *is implemented*"

The following feature (F 2-1:1) can be mapped to said US:

"*Process* ***light instructions***"

Like before, the terms that could be matched are shown in bold. Compound noun matches are indicated by a 'c' in parentheses. Furthermore, if two terms could be matched based on synonymy, this is expressed by including the two terms that are synonyms followed by 'syn.' in parentheses.

ID	US term(s)	PoS tag	Feature term(s)	PoS tag	Match	Unique?
F 2-1:1	light, instruction, lighting	NN, NN, NN	light, instructions	NN, NNS	Precise, near-precise, partial	Yes (c)
F 2-1:1-1	instruction, zone	NN, NN	instruction, zone	NN, NN	Precise, precise	Yes
F 2-1:1-1-1	instruction	NN	instruction	NN	Precise	Yes
F 2-1:1-2	-	-	-	-	-	-
F 2-1:2	sensors	NNS	sensors	NNS	Precise	Yes
F 2-1:2-1	light	NN	light	NN	Precise	Yes
F 2-1:3-1	check, screens	VB, NNS	monitor, screens	VB, NNS	Syn., precise	Yes
F 2-1:3-1-1	state, screens	NN, NNS	state, screen	NN, NN	Precise, near-precise	Yes
F 2-1:3-2	check, lamps	VB, NNS	monitor, lamps	VB, NNS	Syn., precise	Yes
F 2-1:3-2-1	light, state	NN, NN	light, state	NN, NN	Precise, precise	Yes

Table 16: Linguistic matches between USs and features in case 1.

All USs that could be linguistically matched to a feature have at least one match in the action constituent, while only three are and four are observed in the role and benefit constituent respectively. This means that the **action constituent is the most important of the three to identify matches (1.8)**. The 'c' in parentheses indicates whether the words can be considered a compound noun. Furthermore, it is shown when two words are considered synonymous. As is apparent in the table, US 2-1:4 cannot be (linguistically) related to any of the composite or atomic features included in the diagram, even though it should be included given the Epic-module relationship. There are three possible explanations for this. Firstly, the US is actually part of a different Epic and therefore a different module. Secondly, the US describes some implicit functionality that is not included in the features explicitly or is part of multiple features. Thirdly, the functionality described in the US is not at all included in the architecture and can thus be considered redundant or pointless. As was the case with the modules and features, some detail is lost in naming the features. Again, compound nouns are changed into just one of the two nouns. For example, in US2-1:2, the Epic describes 'light sensors', while the feature merely includes the term 'sensors'. In this case, it would presumably be better to include the full compound noun to avoid confusion and ambiguity, since more than one type of sensor can be found in the set of USs.

As was explained previously, 'light' and 'lighting' can be considered synonymous. Furthermore, the terms 'check' and 'monitor' are synonyms according to WordNet, the Merriam-Webster dictionary and the thesaurus. This means that two features can be fully derived from the US they are related to. In US 2-1:1, the word 'use' can be replaced with 'process' without changing the meaning of the US and without breaking any template rules. This would lead to a more accurate and unique match with feature 2-1:1. However, WordNet, the Merriam-Webster dictionary and the thesaurus do not consider 'use' and 'process' to be synonyms. In similar fashion, on the topic of US 2-1:2, 'use output' and 'read' could be considered synonymous. Again, since the chance does not impact

the meaning of the US or feature and does not violate any template rules. Unfortunately, WordNet and the thesaurus do not acknowledge these terms as being synonyms. The Merriam-Webster dictionary does allude to it, by stating that 'read' can be defined as "*to acquire (information) from storage*". Still, 'output' is not considered storage and thus the terms are considered synonymous.

US 2-1:3 can be linguistically related to more than one feature, to two composite features to be precise. Arguably, it would be better to split this US into two USs based on two reasons. Firstly, the US now describes two features and according to the quality criteria it should describe exactly one feature. Secondly, the current formulation can lead to confusion in traceability, which can possibly be avoided. For example, should the US be modified or deleted, it is not clear whether both features should be affected or not. It could be beneficial to split the action constituent into two USs with the same role and benefit constituents.

Considering the linguistic matches shown in the table, it can be reasonably assumed that the action constituent of a US is the most important one when identifying linguistic links. This is also the safer option compared to the benefit constituent, since its inclusion is optional.

5.1.4 Semantic Frames

In addition to quantitative analyses on the PoS tags, more qualitative, or rather semantic, investigation is also desirable. One such semantic analysis is identifying semantic frames. To recap, a semantic frame basically describes which words a certain word is usually associated with or related to. For the semantic frames, only nouns and verbs are considered, since, for instance, adjectives can be related to any noun. Additionally, only frames that were observed multiple times are included in this chapter.

The verb 'use' is present in the action constituent of the USs twenty times (as the first word in that constituent). FrameNet distinguishes two meanings for the word, namely 'using' in general and 'using a resource'. In the USs, 'use' is frequently accompanied by a noun (a resource), such as data. FrameNet states that "an agent has access to a finite amount of some resource and uses it in some way to complete a purpose". In this case, the agent is the role that is specified in the role constituent of the US and the purpose is what is described in the motivation constituent, **which means that a motivation constituent would become mandatory (1.9)**. Sometimes the words 'information', 'knowledge' or 'output' are the resource. In twelve of the 31 Epics, the verb 'send' could be observed in the motivation constituent. According to FrameNet, the word 'send' needs at least a sender, a theme and a recipient or goal. A goal is not meant as an objective here, but as a destination. In Epics the sender is described by 'I' in the template part of the motivation constituent, which, an sich, is not very specific. However, if an Epic can be mapped to a module, the module can help describe the sender. For instance, consider the following Epic:

"*When wind direction changes, I want to send out a notification, so that the data is available for the rest of the system.*"

From just the Epic, it is not clear who 'I' refers to, but it can be mapped to the "wind monitor" module. So the sender is the wind monitor. Using the aforementioned Epic as an example, 'a notification' is the theme (or the thing that is being sent) and the recipient is 'the rest of the system'. It is also possible to define a path, which is the route that the theme takes when sent. In addition, a method of transportation can also be expressed.

Some semantic frames are less apparent or more difficult to identify, since the words do not have an exact match in the FrameNet index. For example, the verb 'activate' is not present in the index, but can be described by the semantic frame that describes changing an operational state. This frame contains an agent who activates (or turns on) a device. Again, the agent can only be accurately determined by the module to which the Epic is related. In the five times the word activate or turn on is used in the 31 Epics, the device that is activated is the (compound) noun that follows the verb in all cases. In nine of the USs and four of the Epics, the verb 'determine' can be observed in the second constituent. This verb is comparable to the choosing frame, which states that a cognizer (the one who chooses), decides upon the chosen (which is an item or a course of action) out of a set of possibilities and may or may not contain an explanation of the decision made. In USs the cognizer is the role, while in Epics this is determined by the module to which the Epic is linked. The chosen entity is the noun that succeeds 'determine'. In some cases a vague explanation of how the decision is made it also contained within the second constituent. An example of such a vague explanation is "based on predictions".

A sharing frame can also be identified in verbs such as 'provide' and 'store'. This frame specifies two protagonists, one who shares something and the second who receives said something. This something is called an entity, the thing that is shared. The construction of this frame in a US is mostly contained in the action constituent. The first protagonist is the role of the US, the entity follows the verb in question (so 'provide' or 'store') and the recipient or second protagonist succeeds the entity. The latter is the last (compound) noun in the action constituent in this case. However, as per the semantic frame, it is not obligatory to define a separate first and second protagonist. It is also possible that there is a collective set of protagonists, a simple example of this is "they share an office", in which 'they' refers to the protagonists. This makes matters slightly more complicated, so consider the following US:

"*As an effect monitor, I want to store proposed actions and executed actions separately, so that I can later compare planner accuracy.*"

In this case, the 'effect monitor' is both the first and second protagonist, which is not a problem in itself, because when there is a lack of a second protagonist, the first can be considered the collective protagonists. However, in this US, the entity compound noun is followed by another compound noun. So, at first glance this latter compound noun would be considered the second protagonist, while it is actually part of the entity. A way to remedy this might be to include coordinating conjunctions (e.g. 'and').

The value of semantic frames may not be immediately apparent from an SA perspective. However, it could prove to be useful in communications between requirements engineers and architects. Consider, for instance, the need for a purpose in the benefut constituent related to the verb 'use' in combination with a resource. This can provide a semantic explanation of what the resource should (be able to) gather, collect or store. This information can help architects decide which data needs to be stored and how the involved modules and/or features should manage this. So, it is possible to **infer architectural constraints (1.10)** without taking a solution-oriented approach in the requirements. In addition to constraints, **semantic frames can help to model information flows or messages related to communications between modules (1.10)**. The sharing frame is an example of this, since it describes what kind of data or information is shared and who the source and recipient are.

5.1.5 Synonyms & Homonyms

In the Epics and modules, four synonyms could be identified and three were observed in the USs and features. This means that, if ontology matching was used, these seven links would possibly be omitted, since ontology matching focuses on exact linguistic term matches. If synonyms had not been considered in the linguistic matches, two features could not have been properly matched to the USs. In case of the Epics, four modules could only be mapped to their respective Epics through identifying synonyms. In two cases these module terms could have been confused with terms found in other Epics (and modules), meaning that mapping would have been more difficult. The synonymous terms can vary looking at their definitions, but are **used interchangeably in the documentation of the system (1.11)**. Only one homonym could be observed, namely 'cloud', which is used to both describe a fluffy thing in the sky and a cloud solution related to computing. However, within the context of the system it is clear which definition is used.

5.2 Case 2

As was mentioned previously, a subset of the system is investigated during the case study due to the system's size and complexity. The subset is created by examining the Epics and selecting those that are, or are suspected to be, related to the client/user side of the system. This includes account management, data management, data storing, interface, etc., focused on the web application itself and some back-end functionality. The YouthData part of the system is also excluded, since it offers similar functionality. The I-Lab Catalog is also not included, since it is a separate entity. Subsequently, the USs are reviewed in the same manner and categorized in Epics, if that has not been done already. If, after examination of the USs, Epics have not been assigned any USs, it is removed from the case study prior to analysis. Epics that only contain unimplemented USs are also removed from the subset. As a result, the following Epics were not included: Cosmetics, Distribution, JsonMetadata, Youth, Windows webdav driver, Youth Data Request, Test

Environment, Integrity, I-Lab Catalog, Tools, Monitoring, Intake and Development Manual.

In the context of this case, Epic is a problematic term. These so-called Epics are in fact more akin to themes, so they more accurately represent a general description of a part of the system or its functionality and are not written as a story (as explained in Chapter 3.4.3). To remedy this, the themes are formulated into Epics, only using information gathered from the RE documents, to prevent validity concerns using the SA documentation as input would introduce. The number of USs is decreased further by removing USs that have not yet been implemented. If the functionality is not present in the code, its architectural elements cannot be recovered and thus not linked to USs. In case of Epics, not only finished Epics could be included, in fact, all Epics are still in development. This can be explained by the USs and their features that are still under construction. The recovered FAM, modeled using the GUI, architecture documentation and code as input can be found in Appendix B (figure 39). The parts of the system that were excluded from analysis are also not included in the recovered architectural artifacts.

5.2.1 Dependency Analysis

First and foremost, the set of USs contains USs with two roles and one action and benefit. Confirming one of the suspicions raised in Chapter 3.4.5. According to the QUS framework, this US should be split into two separate USs to ensure that each US contains only one role. However, that would lead to two nearly identical USs with an identical feature as result. In addition, four USs are contained in two Epics, which implies that the feature that are described by these USs are used by two different modules. This means that the one-to-one cardinality from US to Epic is inaccurate, since USs can be part of more than one Epic and the same holds true for the relationship between modules and features. Logically speaking this makes sense, since it would be redundant to implement a feature twice if it offers the exact same functionality in both instances. Theoretically, an example of a feature that has more than one instance could be a search function. The searching functionality can be called on by more than one module.

The USs were grouped in Epics based on documentation provided by the Yoda developers. Most of the Epics included a clear overview of which USs they contain. However, some of the Epics did not have this overview, so after receiving clarification from the developers what the differences between similar Epics were (such as 'Vault' and 'Manage Vault'), the remaining USs were categorized manually. The following Epics are included in the analysis: Group Manager, Research, Metadata, Revisions, Vault, Statistics, User Management, Manage Vault, Publish Data Package, External User, Metadataschema and Export. Out of these twelve Epics, only Vault did not include an overview of USs.

The recovered architectural artifacts were linked to their requirements engineering counterparts by starting at the highest level of abstraction. Epics were connected to modules by considering the functionalities described by the individual artifacts and matching them based on similarity. Out of twelve Epics all twelve could be linked to a module. However, the formulation of the Epic did not specify whether the module it described would be a sub-module or not and neither could this be inferred. Subsequently, the number of USs per Epic and the number of features per modules were compared (since there are as many Epics as modules, they are not included in the comparison), the results are shown in table 17.

	USs	Features
YDA-02	5	16
YDA-09	20	24
YDA-94	17	56
YDA-95	4	4
YDA-96	2	0
YDA-156	3	1
YDA-157	2	6
YDA-321	6	19
YDA-800	22	3
YDA-2110	4	5
YDA-2208	9	5
YDA-2694	2	-
Total	96	139

Table 17: Number of USs and atomic features contained in each Epic or module in case 2 according to provided documentation.

According to table 17, **more features can be extracted from the GUI than are described in the USs (2.1)**. The lack of features contained in the Vault Space module (YDA-96) can be explained by the fact that this module is only concerned with giving users the ability to save data in the vault (which is not included in the web application and thus the scope of this case study). Manipulation of said data is handled by the Vault Management module (YDA-321). Due to miscommunication, YDA-2694 was included in the USs selection, but as it turns out none of the USs have been implemented. This Epic and its USs are excluded from analysis for the remainder of this research. USs categorizations as specified by the developers were used whenever possible, otherwise reasonable assumptions have been made (as was explained earlier). Only atomic features (features where degree = 0) are included in the count. In addition, features concerned with facilitating navigation were excluded (such as showing the next or previous page of a list). Duplicate features within feature diagrams are excluded, but duplicate features between feature diagrams are included. Choices presented in drop down menus are also excluded if they are modeled as a number in the feature diagram. For instance, when the option to select a language is presented, language selection is considered a feature, the individual language choices are not. Choices made by the system succeeding an action are also not included as atomic features. For example, when adding a new user it is possible to add an existing or new user, but this distinction is made automatically by checking the email address. As is evident, in all but one case the number of USs does not match the number of features. After reading the USs and comparing them to the feature diagrams, there are a few possible explanations for this discrepancy:

1. Some USs describe behavior or reactions, as opposed to functions.
2. Some USs describe a requirement for more than one feature.
3. Multiple USs describe similar functionality.
4. Features have been included based on common sense or practice and have not been explicitly mentioned in the requirements.
5. The Epic-USs categorization does not match the module-feature categorization.

Relating features to USs within the same Epic and module resulted in 26 features related to 30 USs (in some cases one feature could be linked to multiple USs or vice versa). Going by these relations, 66 USs are not yet related to features and the opposite is true for 110 features. To remedy this, the remaining USs and features are related regardless of Epic/module categorization. Note that they will not be related based on linguistics, but rather based on which functionality is described in the USs and which feature contains that functionality.

To see which of the hypothesized explanations is or are correct, the USs are also categorized based on which features they are related to and, using the categorization of features, which module and by extension which Epic. In addition to the five expected discrepancies, two others were identified as well, namely: incorrect level of abstraction and not within the scope of this research. Surprisingly, one US could not be related to any feature, even though the functionality described seems to be within the scope of the research. The USs document exported from Jira has marked

the status of this US (YDA-1677) as done, so either the requirement is not actually satisfied by the system or it could not be found or related. Table 18 provides an overview of the issues that were encountered while relating features to USs and how often they occurred.

Issue	Frequency
Behavioral requirements	33
More than one feature described	7
Functionality described by more than one US	16
Not explicitly described in requirements*	42
Epic-US categorization does not match mod-f categorization	8
Suspected incorrect level of abstraction	4
Outside of scope	6

Table 18: Overview of the issues encountered while relating features to USs and their frequency in case 2.

Note that the frequency refers to the number of USs, for instance, seven USs described more than one feature. Only in case of the features not explicitly described in requirements does the frequency refer to the number of features (indicated by the asterisk). Overlap between the issues is possible. However, if a US is considered a behavioral requirement (in the remainder of this study also referred to as a 'behavioral US'), it does not count towards duplicate functionality. Since checking whether every feature is described by at least one USs is time consuming and prone to error, an estimation is calculated instead: 64 unique features could be related to USs, meaning that 75 features could not be related (going by the original set of 139 features as specified in table 17). Removing the atomic features that were only described by means of their common composite feature (21), 54 features are unrelated to USs. Note that atomic features in a selection are only not included if none of the options have been specified in a US. Removing the duplicates, leftover features between the feature diagrams allows for a decrease by nine. Therefore, it is reasonable to assume that ≈ 42 features are not described by USs. This estimate indicates that $\approx 69{,}8\%$ of features are described by USs. After relating the remaining USs and features it became apparent that USs are not always related to atomic features, but sometimes to composite features. For instance, when the composite feature describes functionality that results in various options that serve as the related atomic features.

An interesting Epic-US/module-feature categorization discrepancy is between the modules Vault Management and Publication Space. According to the 'Asynchronous and Privileged Execution' document, the Publication Space is only used after a datapackage is approved for publication. This means that the process that leads up to a publication, such as submitting for publication, confirming to the agreement and approving the publication, is technically still part of the Vault Management module. Therefore, it seems logical to include such steps in the process in the Publish Data Package Epic, even though according to the architecture they are part of the Manage Vault Epic. This means that the **Epic-US dependencies do not always match the module-feature dependencies (2.2)**. A possible explanation for this variation is that the intended architecture is not always identical to the implemented architecture.

To give an example of a suspected incorrect level of abstraction, consider the following US:

"*As a researcher, I want to add metadata to a datapackage, so that I can satisfy information queries on that datapackage.*"

This US describes a requirement for more than one feature. In fact, it describes an entire module, namely the Metadata Form module. Moreover, the Epic that was formulated to encompass the USs related to metadata is very similar:

"*When I am storing research data, I want to include metadata about the content, so that I can document my data.*"

The US could be refined to suit its benefit constituent better. For instance, by describing metadata that is used specifically for search queries or information that is required to describe the datapackage. Even so, the US would probably need to split into multiple USs.

5.2.2 Epics & Modules

In table 19 the PoS tags of the Epics and modules observed in case 2 are presented.

PoS tag	Situation	Motivation	Expected outcome	Total	Module name	Module template	Total
CC	-	-	3	3	-	-	-
DT	5	6	4	15	-	-	-
EX	2	-	-	2	-	-	-
FW	-	1	-	1	-	-	-
IN	8	2	3	13	-	-	-
JJ	-	4	1	5	2	-	2
MD	-	-	9	9	-	-	-
NN	10	8	7	25	9	12	21
NNP	1	-	2	3	-	-	-
NNS	6	10	10	26	2	-	2
PRP	9	4	10	23	-	-	-
PP$	3	5	4	12	-	-	-
RB	-	2	5	7	-	-	-
TO	1	5	3	9	-	-	-
VB	1	14	10	25	-	-	-
VBG	7	-	-	7	-	-	-
VBN	2	-	4	6	-	-	-
VBP	9	-	2	11	-	-	-
VBZ	2	-	1	3	-	-	-
,	-	-	2	2	-	-	-

Table 19: Overview of PoS tags in Epics and modules in case 2.

As was done in the first case, the last word of the module name is considered the template word. Again, **modules names consist of mostly nouns (2.3)**. Notably, these module names also contained adjectives, like the first case, although these are not included in the linguistic structure of a module. Again, no verbs are included in the modules, even though they are allowed according to the linguistic structure and are used in the Epics. It is possible that some information is lost when naming the modules, but the information may be implicit or not relevant for the architecture.

The linguistic matches observed between the Epics and modules of case 2 are presented in table 20 below.

ID	Epic term(s)	PoS tag	Module term(s)	PoS tag	Match	Unique?
YDA-02	group, manage	NN, VB	group, manager	NN, NN	Precise, partial	Yes
YDA-09	research	NN	research	NN	Precise	Yes
YDA-94	metadata	NNS	metadata	NNS	Precise	Yes
YDA-95	revisions, managing	NNS, VBG	revision, management	NN, NNS	Near-precise, partial	Yes
YDA-96	vault	NN	vault	NN	Precise	Yes
YDA-156	statistics	NNS	statistics	NNS	Precise	Yes
YDA-157	-	-	-	-	-	-
YDA-321	vault, manage	NN, VB	vault, management	NN, NN	Precise, partial	Yes
YDA-800	publish	VB	publication	NN	Partial	Yes
YDA-2110	outside	IN	external	JJ	Synonym	Yes
YDA-2208	(metadata, schema)	NNS, NN	metadataschema	NN	Near-precise	Yes (c)
YDA-2694	export	VB	export	JJ	Partial	Yes

Table 20: Linguistic matches between Epics and modules in case 2.

Only one Epic did not contain any linguistic matches with its corresponding module. This means that, technically, this still results in a unique 'link', since only one Epic and one module will be left when all others have been linked. This module is called "User Management" and there is no mention of a 'user' in the Epic. The closest match is the term 'others', but this seems to be too much of a stretch to justify a linguistic match. It should also be noted that YDA-94 and YDA-96 **can only result in a unique matching through a process of elimination (2.4)**. YDA-2208 and YDA-321 would have to be linked first, since the former two cannot yet be linked with any confidence. In the case of YDA-2208, the terms in the Epic need to be seen as a compound noun (denoted by the 'c' in parentheses), since it is considered a single word in the module name. **Twice adjectives in the functional architecture components were essential for establishing a linguistic match and link (2.5)**, like in the first case.

In table 21, the number of matches per constituent are presented. Note that some terms occur more than once in an Epic and that all instances of the matched term are included in this count.

Epic constituent	*Situation*	*Motivation*	*Expected outcome*
Number of matches	7	8	1

Table 21: Number of matches divided over the three constituents of the Epic template in case 2.

Out of the eleven Epics with linguistic matches, most contained a match in either the situation constituent, the motivation constituent, or both. As is apparent, **most matches were found in the motivation constituent (2.6)**, unlike in the first case. Only once was a match found in the expected outcome constituent. Notably in this case, the matched word was 'manage', which was also present in the motivation constituent, meaning that this match can be considered inessential. Table 22 shows where the linguistic matches were found in terms of word positioning. The expected outcome constituent is omitted, since it contained only one match, which also happened to be a duplicate match.

	Situation constituent		**Motivation constituent**	
	Frequency	*Percentage*	*Frequency*	*Percentage*
First noun	2	28,6	3	37,5
Second noun	2	28,6	2*	25
First verb	2	28,6	3	37,5
Second prep.	1	14,3	0	0

Table 22: Positions of the linguistic matches in Epics in case 2.

The asterisk indicates that one of the second noun matches in the motivation constituent could be a first noun match, if it were extended to a compound noun, without changing the meaning or formulation of the Epic. For instance, from simply 'statistics' to 'storage statistics'. In this table, n is the number of matches for each of the constituents, since all identified matches were unique. As is apparent from the table, the positions of the matched terms differs and there is no safe bet. Based on the positions of case 2, it is a guess as to which terms should be extracted in order to generate module names.

5.2.3 USs & Features

In table 23 the PoS tags per US constituent and per feature are provided. Due to the larger number of features than in the previous case, the numbers have been divided into PoS tags used in composite and atomic features. Like with the feature count, duplicate features within feature diagrams are counted once, while they are included between feature diagrams.

	User Stories				Features		
PoS tag	**Role**	**Action**	**Benefit**	**Total**	**Composite**	**Atomic**	**Total**
CC	1	10	6	17	-	-	-
CD	-	-	1	1	-	2	2
DT	-	111	38	149	-	1	1
FW	-	-	1	1	-	-	-
IN	-	90	38	128	4	20	24
JJ	4	51	25	80	2	21	23
JJR	-	-	2	2	-	-	-
MD	-	7	64	71	-	-	-
NN	105	180	78	363	31	151	182
NNS	12	105	63	180	4	20	24
NNP	3	11	5	19	-	7	7
PRP	-	2	76	78	-	-	-
PP$	-	39	27	66	-	-	-
RB	-	3	19	23	-	-	-
SYM	-	4	3	7	-	-	-
TO	-	48	13	61	2	6	8
VB	-	117	72	189	31	141	172
VBG	-	14	6	20	-	-	-
VBN	-	48	30	78	-	15	15
VBP	-	3	14	17	-	-	-
VBZ	-	5	15	20	-	-	-
WDT	-	-	2	2	-	-	-
WRB	-	1	-	1	-	-	-

Table 23: Overview of PoS tags in USs and features in case 2.

As expected, the USs contain a variety of PoS tags. The features, however, include more tags than just verbs and nouns, which where the predicted tags according to the linguistic structures as prescribed in table 7. Of all the atomic features, 21 did not contain any nouns, but instead included an additional verb, an adjective or, in four cases, consisted of just one word. The vast majority of atomic features, 83, did adhere to the linguistic structure and **contained nothing more than a verb and a (compound) noun (2.7)**. Six USs did not include a benefit constituent.

Table 24 shows the linguistic matches that were observed in 87 USs. Note that USs that could not be linked to a feature (for instance because they were not included in the scope or because they could simply not be found in the implemented system) as well as the USs that should be considered Epics are omitted. While the behavioral requirements (marked with an asterisk) are included to examine the linguistic relationship, they are not taken into consideration while establishing linguistic links (determining unique matches), since **only USs that describe a requirement for a function can be accurately mapped to a feature (2.8)**. Like in the previous case, only nouns, verbs and adjectives were analyzed.

ID	US term(s)	PoS tag	Feature term(s)	PoS tag	Match	Unique?
YDA-03	members	NNS	user	NN	Syn	Yes
YDA-10*	metadata	NNS	metadata	NNS	Precise	-
YDA-12*	metadata, formats	NNS, NNS	metadata, form	NNS, NN	Precise, syn	-
YDA-13	metadata	NNS	metadata	NNS	Precise	No
YDA-14	-	-	-	-	-	-
YDA-15	metadata, file	NNS, NN	metadata, form	NNS, NN	Precise, syn	Yes
YDA-16	discard, metadata	VB, NNS	delete, metadata	VB, NNS	Precise, precise	Yes
YDA-17	metadata	NNS	metadata	NNS	Precise	No
YDA-19	metadata	NNS	metadata; metadata	NNS; NNS	Precise; precise	No
YDA-21	metadata, search	NNS, NN	metadata, search	NNS, VB	Precise, partial	Yes
YDA-76	select, file	VB, NN	select, file	VB, NN	Precise, precise	Yes
YDA-77	-	-	-	-	-	-
YDA-78*	metadata	NNS	metadata	NNS	Precise	-
YDA-82	protect, folder	VB, NN	lock, folder	VB, NN	Syn, precise	Yes (f)
YDA-84	submitted, **vault**	VBN, NN	submitted, vault	VBN, NN	Precise, precise	Yes
YDA-87*	accepted, folder	VBN, NN	accept, folder	VB, NN	Near-precise, precise	-
YDA-146	-	-	-	-	-	-
YDA-162	submit, folder	VB, NN	submit, folder	VB, NN	Precise, precise	Yes (f)
YDA-164*	metadata	NNS	metadata	NNS	Precise	-
YDA-170	unlock, folder	VB, NN	unlock, folder	VB, NN	Precise, precise	Yes (f)
YDA-185	unprotect, folder	VB, NN	unlock, folder	VB, NN	Syn, precise	Yes (f)
YDA-245	update, password	VB, NN	change, password	VB, NN	Syn, precise	Yes
YDA-303*	category	NN	category; subcategory	NN, NN	Precise; near-precise	-
YDA-318	enter, usernames	VB, NNS	enter, username	VB, NN	Precise, near-precise	Yes
YDA-322*	revisions	NNS	revision	NN	Near-precise	-
YDA-324*	submitted, folder	VBN, NN	submit, folder	VB, NN	Near-precise, precise	-

YDA-338*	read-only, access, **group's**	JJ, NN, PP$	read, access, group	NN, NN, NN	Partial, precise, near-precise	-
YDA-380	unlock, folder	VB, NN	unlock, folder	VB, NN	Precise, precise	Yes (f)
YDA-381	unlock, folder	VB, NN	unlock, folder	VB, NN	Precise, precise	Yes (f)
YDA-382	submit, folder	VB, NN	submit, folder	VB, NN	Precise, precise	Yes (f)
YDA-383	submit, folder	VB, NN	submit, folder	VB, NN	Precise, precise	Yes (f)
YDA-384	(undo, **submission**), folder	(VB, NN), NN	(unsubmit), folder	VB, NN	Syn, precise	Yes
YDA-385	lock, folder	VB, NN	lock, folder	VB, NN	Precise, precise	Yes
YDA-388*	accepted, folder	VBN, NN	accept, folder	VB, NN	Near-precise, precise	-
YDA-397	restore, file	VB, NN	restore, file	VB, NN	Precise, precise	Yes
YDA-432	give, read-only, **access**, research, group	VB, JJ, NN, NN, NN	grant, read, access, research, group	VB, NN, NN, NN, NN	Syn, partial, precise/partial, precise, precise	Yes
YDA-441	specify, (spatial, coverage)/**location**	VB, (JJ, NN)/NN	enter, location	VB, NN	Syn, (syn)/precise	Yes (f)
YDA-451*	search	VB	search	VB	Precise	-
YDA-452	restoration	NN	restore	VB	Partial	Yes
YDA-453*	folder	NN	folder	NN	Precise	-
YDA-475	revoke, read-only, access, research, group	VB, JJ, NN, NN, NN	revoke, read, access, research, group	VB, NN, NN, NN, NN	Precise, partial, precise/partial, precise, precise	Yes
YDA-524	history	NN	provenance	NN	Syn	Yes
YDA-606	search, status	VB, NN	search, status	VB, NN	Precise, precise	Yes
YDA-682	specify, accessibility/**access**, (dataset)	VB, NN/NN, NN	select, access, (data, package)	VB, NN, (NNS, NN)	Syn, (near-precise, precise), (syn)	Yes
YDA-801	metadata	NNS	metadata	NNS	Precise	No
YDA-866	approve, publication	VB, NN	approve, publication	VB, NN	Precise, precise	Yes
YDA-867	view, status	VB, NN	show, log	VB, NN	Syn, syn	Yes
YDA-918*	-	-	-	-	-	-
YDA-919	publication	NN	publication	NN	Precise	Yes
YDA-921	confirm, agree	VB, VBP	confirm, agreement	VB, NN	Precise, partial	Yes
YDA-922*	search, status	NN, NN	search, status	VB, NN	Partial, precise	-
YDA-923	cancel, publication	VB, NN	cancel, publication	VB, NN	Precise, precise	Yes
YDA-942*	-	-	-	-	-	-
YDA-985	enter, **date**	VB, NN	enter, date; enter, date	VB, NN; VB, NN	Precise, precise; precise, precise	Yes
YDA-987	system, metadata	NN, NNS	system, metadata	NN, NNS	Precise, precise	Yes (c)
YDA-989*	published/**publication**	VBN/NN	publication	NN	Partial/precise	-
YDA-990	published/**publications**, depublished	VBN/NNS, VBN	publication, depublish	NN, VB	Near-precise/partial, near-precise	Yes
YDA-991*	depublished, **publications**	VBN, NNS	depublish, publication	VB, NN	Near-precise, near-precise	-
YDA-1000*	published	VBN	publication	NN	Partial	-
YDA-1003	specify, **data**, **classification**	VB, NNS, NN	select, data, classification	VB, NNS, NN	Syn, precise, precise	Yes
YDA-1013*	approved, publication	VBN, NN	approve, publication	VB, NN	Near-precise, precise	-
YDA-1031	-	-	-	-	-	-
YDA-1034*	-	-	-	-	-	-
YDA-1068*	publication	NN	publication	NN	Precise	-
YDA-1389	specify, (spatial, coverage)/**location**	VB, (JJ, NN)/NN	enter, (location)	VB, (NN)	Syn, (syn)/precise	Yes (f)
YDA-1463*	location	NN	location	NN	Precise	-
YDA-1464*	-	-	-	-	-	-
YDA-1465*	-	-	-	-	-	-
YDA-1486*	-	-	-	-	-	-
YDA-1605	republish	VB	republish	VB	Precise	Yes (f)
YDA-1606	republish	VB	republish	VB	Precise	Yes (f)
YDA-1823*	published	VBN	publication	NN	Partial	-
YDA-1998	fold, unfold, (categories, subcategories)	VB, VB, (NNS, NNS)	hide, (category); show (category)	VB, (NN); VB, (NN)	Syn, (precise, near-precise); syn, (precise, near-precise)	Yes
YDA-2001	groups	NNS	group	NN	Near-precise	Yes
YDA-2111	invite, external, user	VB, JJ, NN	enroll, external, user	VB, JJ, NN	Syn, precise, precise	Yes (c, f)
YDA-2112	provision, external, user	VB, JJ, NN	enroll, external, user	VB, JJ, NN	Syn, precise, precise	Yes (c, f)
YDA-2163	(set, new), password (b)	(VB, JJ), NN	reset, password	VB, NN	(Syn), precise	Yes
YDA-2164*	metadata	NNS	metadata	NNS	Precise	-
YDA-2165	transform, Yoda-metadata/metadata	VB, NNS/NNS	transform, metadata	VB, NNS	Precise, near-precise	Yes (f)
YDA-2166*	view, transformation, result	VB, NN, NN	view, transformation, result	VB, NN, NN	Precise, precise, precise	-
YDA-2286	indicate, datatype	VB, NN	select, type	VB, NN	Syn, partial	Yes
YDA-2360*	metadata, file	NNS, NN	metadata, form	NNS, NN	Precise, syn	-
YDA-2361*	metadata, files	NNS, NNS	metadata, form	NNS, NN	Precise, syn	-
YDA-2362	convert, metadata	VB, NNS	transform, metadata	VB, NNS	Syn, precise	Yes (f)
YDA-2363	metadata	NNS	metadata	NNS	Precise	No
YDA-2644	view, transformation, result	VB, NN, NN	view, transformation, result	VB, NN, NN	Precise, precise, precise	Yes
YDA-2645	accept, reject, (transformation)	VB, VB, NN	accept, (transformation); reject, (transformation)	VB, (NN); VB, (NN)	Precise, precise, (precise)	Yes

Table 24: Linguistic matches between USs and features in case 2.

Only seven USs had a linguistic match in the benefit constituent (such matches are in bold) and three of these contained a partial linguistic match for the same term in the action constituent as well (YDA-682, YDA-989 and YDA-990), so **most matches can be found in the action constituent (2.9)**. Linguistic links have been established slightly differently from before. In addition to considering compound nouns, **multiple USs that describe the same feature (2.10)** are also taken into account, indicated by an 'f' in parentheses. Out of the 57 USs that describe a function, 48 have a unique linguistic match and can thus be said to have a linguistic link. In order to establish these links, **the process of elimination is crucial (2.3)**. Starting with the most reliable or certain links allows other, weaker matches to be considered unique as well. Five USs could not be uniquely linked to a feature and four USs did not have a linguistic match with their related feature at all.

Behavioral requirements in table 24 are not linked to the feature that they describe, since they

do not technically describe a requirement for a feature. Instead, they have been mapped to the feature of which they extend the functionality or for which they describe some response or action, **which can lead to more than one relation (2.11)**. Oftentimes, they are related to features that are already described by a functional US. However, it is possible to identify linguistic matches for behavioral USs.

Table 25 shows which terms could be linguistically matched in the context of their position in the US. For this analysis only the action constituent of the US is considered, since that is where 80 out of 87 linguistic matches were found.

		Frequency	**Percentage**
Verbmatch	*First verb*	32	61,5
	First verb excl. have/be	6	11,5
	CC tag second verb	4	7,7
	Other verb	4	7,7
Nounmatch	*First noun*	31	59,6
	First compound noun	1	1,9
	First noun if ext. to compound	7	13,5
	First 'partial' match	2	3,8
	Other noun	5	9,6

Table 25: Positions of the linguistic matches in USs in case 2.

Note that it is possible that the percentages do not add up to 100%, since not all USs contained a verb match or noun match. Only USs that could be uniquely matched to a feature are included in the analysis. In the event that multiple features are linked to the same US, the US was duplicated to avoid introducing a skew. In the table, 'CC tag second verb' refers to the instances where a CC tag is present in the US (such as the word 'and', a comma or a forward slash) and the verb after the CC tag is the one that could be matched to the verb in the feature. For the nouns, 'first noun if ext. to compound noun' refers to situations in which the linguistic match is the first noun, if the noun in the US is extended to a compound noun. Finally, 'first partial match' means that the linguistic match was considered partial according to table 24.

In relation to the conceptual model of a US, as shown in figure 17, the terms required to derive feature names are located in the action constituent of a US, more particularly the action verb and direct object. In the example, visualized in figure 33, the terms that need to extracted from a US are highlighted.

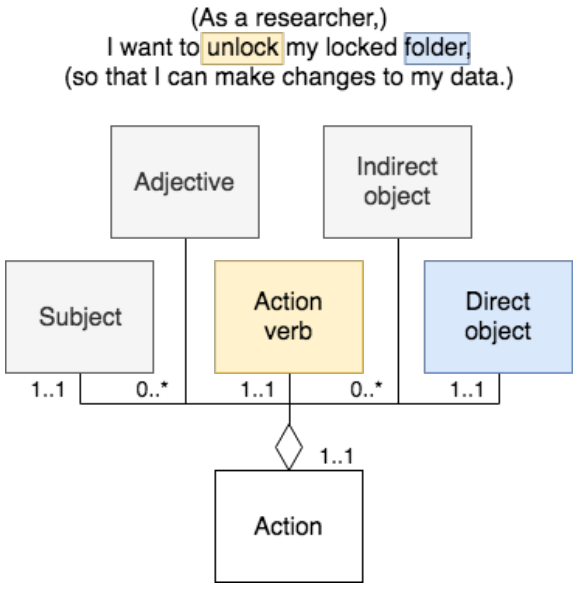

Figure 33: Derivation of feature names visualized in the conceptual model of the action constituent of a US.

The subject, adjectives and indirect objects are grayed out, since they are not used to derive a feature name, according to table 25. Furthermore, the selected US does not contain a compound noun, however, compound nouns would be considered a direct object as well.

If linguistic matches and links were to be identified automatically, the words themselves and their PoS tags would be considered, meaning that this information is of no value to that endeavor. However, the objective of this research also included support for generating (partial) artifacts. In particular, how to generate features, or at least the names of features, by extracting terms from USs. While 61,5% does not seem to be the safest bet, the accuracy can be increased by excluding verbs such as 'have' and 'be', which often do not describe any specifics about a feature, but are required to formulate a proper phrase. Even then, 73% accuracy is still questionable. In the case of nouns there is more uncertainty, with a frequency of 59,6%. Although, the first compound noun can be considered as well and partial matches are still matches, bringing the percentage up to 78,8%, which is an improvement. However, not in all cases can the the noun match be extended to the first compound noun, in five cases (9,6%), this could not be done. In the context of this case this would mean that 27% of the verbs extracted from the USs would be incorrect or inappropriate and 21,2% of the nouns. On the other hand, **combining the first verb and the first noun to create a feature name is a decent starting point 2.12)**. The accuracy can then be improved by evaluating the feature names that were formed as such and by fine-tuning the extraction of terms. Another approach would be to teach requirements engineers to write USs to fit the feature derivation rules, but that would require them to think solution-oriented, while they should only focus on the problem.

5.2.4 Semantic Frames

To get some direction in the search for semantic frames, the semantic frames identified in case 1 are examined for case 2 first. Like before, frames are only included if they can be observed more than once.

The only frames case 1 and 2 have in common are the frames 'using resource' and 'store'. The former varies slightly from case 1, it more specifically mentions reusing a resource as opposed to using one. The frame using resource is described by FrameNet as: "an agent has access to a finite amount of some resource and uses it in some way to complete a purpose". In case 2, the resource is either data or a data package, which refer to the same kind of resource. The purpose can differ, since in one instance an outsider wants to use the data for their own research and in the other the owner of the data wants to make it publicly available (for example to adhere to the FAIR principles), but does **require a benefit constituent (2.13)**. The semantic frame 'store' can be observed five times in the Epics. FrameNet states that "*a supply of a resource is kept safe and available for future use*". In addition, there may be a possessor who owns (possesses) said supply. In this context, the supply always concerns (research) data. They are kept safe and made available by storing them in the Yoda system (either the group storage or the vault). The possessor is the owner of the data, so in this context that would be a researcher.

The semantic frame 'change accessibility' was observed multiple times, in relation to user permissions. This frame states that "an agent causes a useful location to become accessible (or non-accessible) to a theme to a certain degree". In this case, the agent is a user with the rights to promote or demote other users, the useful location is a specific data package (either in the vault or in group storage), the theme refers to other users with specified permissions and the degree refers to no access, read-only access and read/edit permission. Two USs contain all the previously mentioned elements:

1. "*As a data manager, I want to give read-only access on a data package in the vault to group users of the corresponding research group, so that they can access this package.*"
2. "*As a data manager, I want to revoke read-only access on a data package in the vault to group users of the corresponding research group, so that they can access this package.*"

Other USs concerned with access and/or permissions contain only a subset of the elements. For instance: "*As a group member, I need read-only access to vault data, so that I can access all our group's archived data packages.*" This US does not specify an agent, only the location, the theme and the degree.

In similar fashion, the semantic frame 'choosing' could be observed, albeit in multiple forms using different words. The frame is described as follows: "a cognizer decides upon the chosen

(either an item or a course of action) out of a set of possibilities". In the Epics and USs, the word 'choose' is never used, but 'select' and 'indicate' are used instead. To a lesser degree, 'specify' and 'enter' can be related to this frame, but both of these terms are not accompanied by any options.

Forgiving the fact that one of the elements is implied, the 'publishing' frame, as could be expected, is also present. It contains the following elements: "a publisher makes a work of an author (or authors) available to some public audience for general enjoyment, examination, and reference". The publisher is not explicitly mentioned in the Epics or USs, but is instead implied as it is the system (Yoda). The author refers to the role that wants to publish something, in most cases a researcher, and the work is research data or a data package. The audience is the general public, as the data is made publicly accessible. It is reasonable to say that the semantic frame 'searching scenario' (defined as: "a cognizer seeks a phenomenon within the ground") is used, although the ground is never specified, since it concerns the system, or the storage space of the user within the system. The cognizer is always the role, so either a general user or a researcher and the phenomenon tends to refer to a folder, file or data package. Like in the previous case, **semantic frames can provide additional constraints or information on how the system should be developed and implemented (2.14)**.

5.2.5 Synonyms & Homonyms

Multiple synonyms could be found, as is clear from tables 20 and 24. The synonyms are not limited to those identified in the aforementioned tables, since synonyms within the artifacts are possible too. A confusing inconsistency, which may simply consist of synonyms, is the use of the words 'data', 'dataset', 'data package' (with or without a space), 'folder' and 'file', seemingly without any obvious pattern or reason. If all these terms can be used interchangeably, there is no issue. At this point in time, that is, since future extensions and modifications could refer to specific entities. Currently, according to the developer, the naming convention is that data (data in general, a file or a folder) is referred to as a data package as soon as it enters the vault. However, in case of the US that is concerned with searching, this is not the case, because it is also possible to search in the research module. The term 'dataset' is only used in the context of metadata, but seems interchangeable with folder, file, data and data package (depending on the location of the data). Moreover, the term 'folder' is also used for data packages (contained in an accepted folder) in the vault, which may lead to confusion. The terms 'format' and 'form' appear to be interchangeable as well, even though they are not clearly synonymous. Arguably, a 'format' refers to the general makeup (according to the Merriam-Webster dictionary), while a 'form' is a typed document that requires someone to fill in information. However, in this case the format is contained in a form. In the current implementation of the system, the two terms refer to the same object, but if other forms or formats are added in the future, this may lead to confusion. These terms become more puzzling given the fact that the term 'file' is sometimes also used to refer to a 'form' and in other cases to indicate a 'regular' file or document that contains some data or information.

A set of less obvious synonyms is 'specify', 'enter' and 'select'. From a linguistic standpoint (in relation to definitions) they seem quite different, but in the context of software (systems) they can be similar. For instance, the term 'select' is often used when there are options to choose from and 'enter' is used when the user types something. However, in the case of 'enter start/end date', the user is not asked to type a date, but can select a day and month from a calendar pop-up. To make matters more complicated, the term 'indicate' is sometimes also used to refer to 'select'. For instance, when in the RE documentation 'indicating a datatype' is mentioned, this is related to 'selecting a datatype' from a drop-down menu in the architecture and GUI. Whenever there are options involved, the synonymous terms tend to adhere to the 'choosing' semantic frame, so 'specify' and 'enter' do not belong in the list of synonyms, unless they are implemented with options.

More obvious synonyms include 'history' and 'provenance', 'unprotect' and 'unlock', 'spatial coverage' and 'location' and 'undo + submission' and 'unsubmit'. Although the latter introduces a challenge, since the terms cannot be considered a compound noun and would therefore need to be identified separately. The terms 'state' and 'status' are also synonymous as they both refer to a certain condition: either 'being in a certain state' or 'having a certain status'. The terms can be used interchangeably in the USs.

The terms 'external' and 'outside' are synonymous and were also identified in case 1. In both cases, 'outside' was used in an Epic and 'external' in the corresponding module, so it seems that the latter is a more technical term. Oftentimes terms used in Epics and USs are translated into a more

technical, synonymous term in their architectural counterparts. Other such synonyms include (RE term first, followed by SA term):

1. members - users
2. view - show
3. status - log
4. fold - hide
5. unfold - show
6. set + new - reset
7. invite/provision - enroll
8. give - grant

Arguably, 'members' and 'users' are not synonymous. Especially when the context concerns IT, one can argue that users do not pay and members do. However, in this case, one US mentions "enter usernames of new members", which confirms they are synonyms. Twice the SA term seems to be the less technical option, namely in the cases of 'update' and 'change' and 'convert' and 'transform'. The latter set of terms are synonymous, the former not necessarily. However, in this case updating refers to providing a newer version of a file or form that contains changes. Finally, 'subproperties' and 'attributes' seem to refer to the same implemented functionality and are both related to metadata. Moreover, the Merriam-Webster dictionary lists attribute as a synonym for property. The feature names could not be linguistically linked to the USs that mention the terms, which is why it is unclear whether the terms are synonymous or refer to different values/information. Furthermore, the term 'field' is also used in combination with metadata. Although this term seems to refer to any kind of information related to metadata that can be filled in, this might also contain subproperties and attributes. No homonyms could be observed in any of the four artifacts.

To summarize, synonyms are not solely qualified on their definitions (linguistics), but also on their meaning within the context of the system and what they refer to (software development). **Linguistically speaking some terms may differ, while they can and are used interchangeably in the implementation and vice versa (2.15)**.

5.2.6 Deriving Feature Names

The Yoda system is up and running, yet still evolving. Several USs that were included in the documentation have not been developed yet and will be addressed in upcoming sprints. These backlog USs provide an excellent opportunity for testing the correctness and relevance of deriving feature names. According to table 25 the majority of the links with feature names consist of the first verb and the first (compound) noun in the USs. In the context of a small-scale experiment, the USs marked as 'to do' will be used to derive feature names, following the first verb + first (compound) noun structure. The USs selection process will follow mostly the same criteria as before, but to be clear they are listed below:

1. No enabler stories
2. No behavioral requirements
3. No quality requirements
4. Only roles that were included in Chapter 5.2.3
5. Only USs with status 'to do'
6. Only USs that are a well-formed sentence
7. Only USs that describe a requirement for exactly one feature

The results of applying the linguistic link structure (first verb + first (compound) noun) to derive feature names is provided in table 26. Note that the verb and noun do not necessarily appear in this order in the USs. Feature names derived from 24 USs are included in the table.

ID	US	*Feature name*
YDA-2641	As a researcher, I want to **view** the **status** of my submitted data requests, so that I am informed of its progress.	*View status*
YDA-2640	As a data manager, I want to **view** all **research proposals**, so that I know which data may be released.	*View research proposals*
YDA-2639 YDA-2508	As a researcher, I want to **submit** my full **research proposal**, so that it can be appraised.	*Submit research proposal*
YDA-2547	As an admin, I want **users deprovisioned** from Yoda that are not member of any group.	*Deprovision users*
YDA-2564	As a datamanager, I want to **know** that a **group** within my category has not been active for 3 months, so that I can detect inactive groups.	*Know group*
YDA-2511	As a datamanager (board secretary), I want to **accept** a submitted **research proposal**, so that research can start.	*Accept research proposal*
YDA-2510	As a datamanager (board secretary), I want to **view** a submitted **research proposal**, so that I can appraise it.	*View research proposal*
YDA-2509	As a researcher, I want to **submit** a **data request**, so that the data manager can give me access to the data.	*Submit data request*
YDA-2507	As researcher, I want an overview of my **research proposals**, so that I can **track** their status.	*(track) research proposals*
YDA-2296	As a Yoda admin, I want to **have** an **overview** of all user autorisation changes, so that I can analyse incidents.	*Have overview*
YDA-2289	As a datamanager, I want to **know** to which **groups** a user belongs, so that I can manage my community.	*Know groups*
YDA-2213	As a researcher, I want to **specify** my **data package type**, so that I can have types other than dataset.	*Specify data package type*
YDA-2167	As a researcher, I can **view** original **metadata** alongside transformed metadata, so that I can make an informed decision on transformation acceptance.	*View metadata*
YDA-2001	As a datamanager, I want to **see** the **statistics** of all groups in my categories, so that I can supervise storage usage and costs.	*See statistics*
YDA-1715	As a user, I want to **reference** a **data object** in my package as the value of a metadata field, so that I can describe the function of an object in my data package.	*Reference data object*
YDA-1641	As a system admin, I want to **verify** all SSL related **info**, so that I know SSL certs are configured properly.	*Verify (SSL) info*
YDA-1628	As a researcher, I want to **select** a **single point** using a map in order to specify a single coordinate.	*Select single point*
YDA-1624	As a group manager, I want to **inform** new **Yoda users** how to use Yoda, so that they can get started.	*Inform Yoda users*
YDA-1585	As a datamanager, I want to **process vault data** and save result in vault.	*Process vault data*
YDA-1083	As a researcher, I want to **delete** a named **shopping bag**, so that I can manage my bags.	*Delete shopping bag*
YDA-1080	As a researcher, I want to **view** a **list** of all items in my selected shopping bag.	*View list*
YDA-1079	As a researcher, I want to **click** on a **data package** to add it to my shopping bag.	*Click data package*
YDA-1078	As a researcher, I want to **select** a named **shopping bag**, so that all further shopping actions relate to this bag.	*Select shopping bag*
YDA-1076	As a researcher, I want to **create** a named **shopping bag**, so that I can add selected data packages.	*Create shopping bag*

Table 26: Feature names derived from nouns and verbs extracted from USs in case 2.

YDA-2639 and YDA-2508 are considered duplicates, since they describe a requirement for the same functionality. Arguably, some of the derived feature names are not particularly useful and/or relevant due to unspecific verbs used in the USs, examples of this are: YDA-2564, YDA-2296, YDA-2289. All three can be easily rewritten in such a way that they lend themselves to the feature name derivation:

YDA-2564: "*As a datamanager, I want to view inactive groups, so that I can detect groups within my category that have not been active for 3 months.*"

YDA-2296: "*As a Yoda admin, I want to view all user autorisation changes, so that I can analyse incidents.*"

YDA-2289: "*As a datamanager, I want to view a user's groups, so that I can manage my community.*"

The new formulations of the USs do not include any solutions and are therefore appropriate for the problem space. The other 21 feature names, from an outsider perspective, seem relevant and clear and could prove to be useful to software architects and developers.

Two feature names contain terms in brackets. Firstly, in the case of YDA-1641, 'SSL' is not included since it is a proper noun. Furthermore, it is not directly adjacent to 'info', which makes it difficult to consider 'SSL info' a compound noun. Instead of 'info', 'SSL' could also be used, if no distinction is made between 'regular' nouns and proper nouns. Secondly, the first verb in YDA-2507 is 'track', but it is positioned in the benefit constituent, which is not mandatory. As an alternative, a partial match could be established by using the word 'view' derived from 'overview'. However, this would make automatically deriving feature names more difficult, since overview is a noun and considered as such in other USs and feature names (for instance in YDA-2296). As a countermeasure, requirements engineers could be asked to consider the use of an action verb in every action constituent.

To confirm which derived feature names are useful and/or relevant and which are not, the derived names were assessed by a Yoda developer. The results are presented in table 27. Whenever a feature names was considered irrelevant or useless, a rationale was provided to explain why.

US ID	*Feature name*	Useful/relevant	Rationale
YDA-2641	*View status*	No	Too generic
YDA-2640	*View research proposals*	Yes	-
YDA-2639/2508	*Submit research proposal*	Yes	-
YDA-2547	*Deprovision users*	Yes	-
YDA-2564	*Know group*	No	Inaccurate
YDA-2511	*Accept research proposal*	Yes	-
YDA-2510	*View research proposal*	Yes	-
YDA-2509	*Submit data request*	Yes	-
YDA-2507	*(track) research proposals*	Yes	-
YDA-2296	*Have overview*	No	Too generic
YDA-2289	*Know groups*	No	Too generic
YDA-2213	*Specify data package type*	Yes	-
YDA-2167	*View metadata*	No	Inaccurate
YDA-2001	*See statistics*	No	Too generic
YDA-1715	*Reference data object*	No	Vague
YDA-1641	*Verify (SSL) info*	Yes	-
YDA-1628	*Select single point*	No	Too generic
YDA-1624	*Inform Yoda users*	Yes	-
YDA-1585	*Process vault data*	Yes	-
YDA-1083	*Delete shopping bag*	Yes	-
YDA-1080	*View list*	No	Too generic
YDA-1079	*Click data package*	No	Inaccurate
YDA-1078	*Select shopping bag*	Yes	-
YDA-1076	*Create shopping bag*	Yes	-

Table 27: Assessment of the derived feature names in case 2.

According to the developer, fourteen out of 24 feature names were relevant and/or useful for development, which makes **extracting the first verb and first (compound) noun a promising approach (2.12)**. Out of the ten irrelevant/useless feature names, this was to be expected in three cases (YDA-2564, YDA-2296 and YDA-2289). Six were considered too generic, three inaccurate and one too vague. To start with the latter, the vagueness could possibly be solved by taking the Epic/module it belongs to into account. The developer's rationale is: "*important that reference is in metadata*". If the feature is placed in the Metadata Form module, this would be clear enough. In the case of the remaining generic feature names, the issue could potentially also be solved by considering the Epic/module. The rationale for rejecting the feature name derived from YDA-1079 is: "*real feature is adding it to the shopping bag*", which is a reasonable judgment. However, this might imply that the US contained a technical solution to the problem, by stating that 'clicking on a data package' is how the data package should be added to the shopping bag. A US focused exclusively on the problem space would then be: "*As a researcher, I want to add a*

data package to my shopping bag." Finally, the feature name derived from YDA-2167 is just too inaccurate according to the developer's rationale: "*the feature is viewing original and transformed metadata alongside*".

5.3 Case 3

As was stated previously, due to the sensitive materials used in case 3, no details about this case may be disclosed publicly.

5.4 Functionality in RE & SA

In the context of SA, or rather functional architectures, functionality would have to be related to some architectural element(s), otherwise it would have to be implicit functionality, which is not immediately apparent in a functional architecture. As such, explicit functionality is relatively easy to identify: it is described by functional requirements, or more specifically, requirements that describe functions. This can be done through USs for RE and features as their explicit architectural counterparts. what is left is implicit functionality, which seems, according to the case studies, more akin to behavior and responses (for the sake of brevity both will be referred to as behavior or behavioral) rather than functions. Behavior can still be described in requirements, so through USs as well. So how can behavioral requirements be captured in architectural elements? The obvious solution is to not limit SA to functional architectures only. As is apparent from the terminology, a functional architecture captures functional requirements, so behavioral requirements need to be translated into an architecture using a behavioral view, as opposed to a functional one.

Rozanski and Woods present an information view in addition to a functional view (Rozanski & Woods, 2011). While an information view does capture how information is handled and stored in the system, this view takes a more technical approach (similar to database modeling). Moreover, while it does describe how information is used and manipulated, it does not specify when and how this happens by default, so it is disconnected from the functional view. Information flows in the information view should be consistent with those in the functional view, however, the functional view does not allow for detailed information flows, since that is not its purpose. Including detailed information flows for every piece of information in the system would increase the complexity of a FAM, simultaneously defeating its purpose of providing a relatively simple overview of the internal structure of a system. The concurrency view, on the other hand, maps tasks to functional elements and also models communication. Consider the following US (taken from case 2) as an example:

"*As a researcher and datamanager, I want an email notification of newly published data packages, so that I am informed about its publication.*" (YDA-989)

This US does not describe a functional requirement, since there is no specific function or user interaction associated with it. Instead, it extends the behavior of another functional requirement, namely:

"*As a data manager, I want to approve publication of a data package that is in the vault, so that it can be found publicly.*" (YDA-866)

This US describes a function, namely being able to approve publication of a data package. In a concurrency view, or to be specific a Petri net, this can be modeled using a state that says that there should be a data package awaiting approval for publication in the vault then the option to either approve or reject that data package, followed by the data package being published or not (depending on the decision made). When the data package is published, two transitions can be used: one that describes that a data package is made accessible to the public and a second one that describes a message is sent to the owner(s) of the data package so that they know it was published. A trade-off analysis shows that there are as many pros as there are cons, as shown in table 28.

Benefits	Drawbacks
- ensures that behavioral requirements are satisfied by the system	- requires additional modeling
- communication/behavior of the system is visualized, which might improve stakeholder communication	- does not directly translate to specific parts of code (possibly multiple locations and instances involved)
- allows for analysis, since Petri nets can be tested on their quality	- may require training in order to be done correctly

Table 28: Benefits and drawbacks of using a concurrency view in addition to the functional view.

A potential solution to the issues of the difficulty of Petri nets and the 'technical' aspect of class diagrams would be to use sequence diagrams, which are not included as architectural models by Rozanski and Woods. Sequence diagrams, however, show both the information that is transferred between points and also the order (sequence) in which steps or activities are performed. Moreover, they allow for a simple distinction between user interaction and system behavior.

As a small experiment, the behavioral requirements related to the "*As a data manager, I want to approve publication of a data package that is in the vault, so that it can be found publicly*" (YDA-866) US, linked to the "approve for publication" feature and categorized in the 'Publish Data Package' (YDA-800) Epic/module, are modeled in a sequence diagram. This selection includes the following USs:

1. YDA-918: "*As a datamanager, I want the vault copy of an accepted data package to be registered with the state unpublished, so that their status is accounted for.*"
2. YDA-942: "*As a researcher, I want to have Datacite compliant metadata derived from selected Yoda metadata, so that later my metadata can be found.*"
3. YDA-1000: "*As a datamanager, I want a published data package to have DOI, so that it can be found and referenced publicly.*"
4. YDA-1013: "*As a researcher, I want a package approved for publication to be published, so that others can find my data.*"
5. YDA-1034: "*As a researcher, I want my data package shown on a themed landingpage, so that it has a professional image.*"
6. YDA-1068: "*As a publisher, I want a license file to be added to a data package upon publication, so that downloaders are informed of the license conditions.*"
7. YDA-1823: "*As a user, I want a detailed data package published notification, so that I am informed.*"

Since it is unclear which of these responses/behaviors are concurrent and which sequential, they are modeled according to the numerical order of the USs. the resulting sequence diagram is presented in figure 34.

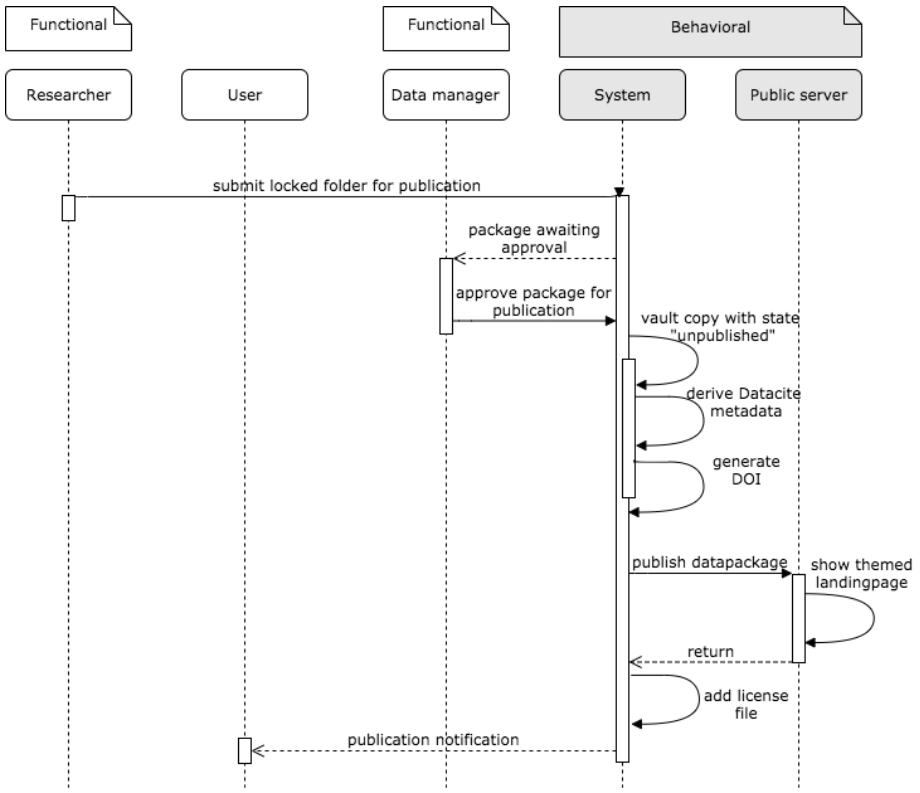

Figure 34: Example of a sequence diagram to model behavioral USs.

In the diagram, a distinction is made between roles (people) in white boxes and systems in grey boxes. In this case "system" refers to Yoda as a whole (to reduce complexity the individual modules are not included). For the sake of completeness the trigger of the aforementioned USs is included as well, which is when a researcher submits a locked folder for publication. Another US that describes a function that is included is the approval of a package by the data manager. All the other messages shown in the sequence diagram illustrate either system behavior or system responses. Utilizing a sequence diagram is a relatively simple approach, since every US in the selection can be mapped to a message (with the exception of the "return" message, which is only included so as not to interrupt the sequence). In addition, by modeling this behavior it became evident that the vault copy of a datapackage is registered with the state "unpublished", but that this state is never updated after publication, or at least not according to the USs.

On the contrary, features that are not described in any USs were also observed. These features can be seen as implicit functionality that are implemented based on common sense or practice. Consider features such as those that realize GUI functionality. Fine examples are features related to navigation, for instance being able to click on the next page if items in a list do not fit on one screen or clicking on the logo to go to the home screen. Nearly all modules, and thus most the feature diagrams, contained features that enable navigation, yet no USs covered this functionality.

6 Results

In this chapter, the results of two of three case studies are discussed. Firstly, the findings of each case are listed separately. Subsequently, the qualitative results (the aforementioned findings) of the analyses are compared to see which are in agreement and which contradict each other. Then, the quantitative results of two cases are combined to provide an overview of the outcome of the research as a whole. Finally, the quality of the Epics and USs analyzed during the study is discussed. Please note that the only the results from cases 1 and 2 are included, due to confidential information present in case 3, which may not be disclosed publicly and is omitted here. This means that all results, findings and conclusions are based exclusively on case 1 and 2.

To start, the qualitative (preliminary) findings of the first case are summarized:

1.1 There is not necessarily a one-to-one relationship between USs and features.

1.2 Module names consist of mostly, if not all, nouns.

1.3 The process of elimination is crucial when establishing as many linguistic matches and subsequent links as possible.

1.4 Adjectives can be essential for establishing (unique) linguistic matches.

1.5 Most Epic-module matches are found in the situation constituent.

1.6 Module names are less specific than their description by the related Epic, due to compound nouns not being included in full.

1.7 Feature names mostly contain nouns and verbs.

1.8 Most US-feature matches are found in the action constituent.

1.9 When the verb 'use' is present in the action constituent of and it refers to a resource, the benefit constituent is mandatory according to the semantic frame of 'use', which is the case in the USs.

1.10 Semantic frames can provide additional input for development of the architecture of a system.

1.11 Linguistically speaking some terms may differ, while they can and are used interchangeably in the implementation and vice versa.

The last finding refers to the fact that the definition of linguistic terms may not be synonymous, such as 'information' and 'data', which are arguably not the same, but are used interchangeably in the documentation regardless.
Secondly, the qualitative (preliminary) findings of the second case are listed below:

2.1 Not nearly all (atomic) features are explicitly described in USs, so feature diagram recovery utilizing the GUI results in a more complete set of features.

2.2 The dependencies between Epics and USs are not necessarily the same as the dependencies between their respective modules and features.

2.3 Module names consist of mostly, if not all, nouns.

2.4 The process of elimination is crucial when establishing as many linguistic matches and subsequent links as possible.

2.5 Adjectives can be essential for establishing (unique) linguistic matches.

2.6 Most Epic-module matches are found in the motivation constituent.

2.7 Feature names mostly contain nouns and verbs.

2.8 Only USs that describe functions, not behavior or reactions, can be accurately mapped to features in the architecture.

2.9 Most US-feature matches are found in the action constituent.

2.10 There is not necessarily a one-to-one relationship between USs and features.

2.11 Behavioral USs can be related to more than one feature and both explicit functions and results of a particular (set of) feature(s).

2.12 When deriving feature names, extracting the first verb and first (compound) noun from a USs seems a promising and accurate approach.

2.13 When the verb 'use' is present in the action constituent of and it refers to a resource, the benefit constituent is mandatory according to the semantic frame of 'use', which is the case in the USs.

2.14 Semantic frames can provide additional input for development of the architecture of a system.

2.15 Linguistically speaking some terms may differ, while they can and are used interchangeably in the implementation and vice versa.

There are various speculated explanations for finding 2.1. Firstly, as was mentioned previously, front-end features that are solely related to the GUI, such as navigation buttons, are not included in the USs. Secondly, oftentimes if a composite feature contains multiple atomic features, only the composite feature is described in a US. This was confirmed by a fellow researcher, who also said that feature diagrams recovered from the GUI are more complete than only taking into consideration the front-end features described in USs. Based on this it seems that the features contained in an alternative structure (mainly drop-down menus) in feature diagrams are not included in the requirements documentation, but only their parent is. The one exception that can be found is when there is a Boolean choice, such as accepting or rejecting changes or confirming or canceling an agreement. Finally, a situation in which atomic features in an alternative structure are explicitly described in USs, is when the option of the alternative structure cannot be reasonably assumed. For instance, a composite feature called 'personal details' can easily be expanded with different pieces of information that can be assumed, such as name, date of birth, address, etc. The 'search by status' composite feature in case 2 (refer to figure 40) is not as self-explanatory and is specific to this system, which is why some of its atomic features are covered by USs. Arguably, the accuracy of this finding heavily depends on how a feature is defined, since recovering interactive elements from the GUI is one of the most granular approaches to identifying features. Either way, feature diagrams recovered from the GUI provide a more detailed representation of the implemented system, but the question is whether this is desirable and to what extent.
Thirdly, the qualitative (preliminary) findings of the third case are not shared publicly.

First and foremost, it is apparent that most linguistic matches between USs and features were found in the action constituent of a US. This was to be expected, since this is where the requirement for a particular feature is described. Secondly, modules names contain almost exclusively nouns, as specified for module naming, but sometimes adjectives can be included as well. Similarly, features names consist mostly of nouns and verbs, which is also in line with the naming conventions. Furthermore, while not investigated in detail, it seems that semantic frames may potentially provide additional input for software architects for development of the architecture. However, it is unclear whether this input is self-explanatory and therefore redundant or if it can be useful. Additional research is needed to evaluate this finding. Finally, in both case 1 and 2 synonyms were identified (results from case 3 are omitted due to sensitive materials). However, the terms were synonymous the way they were used in the implementation, while their dictionary definitions may differ. In the current state of the systems, this does not have any consequences, but in the future the terms may become inaccurate or confusing.

Most of the Epic-module matches in case 1 can be found in the situation constituent. However, in case 2 most matches could be found in the motivation constituent, so the results are somewhat conflicting as shown in table 29.

		Epic constituent		
		Situation	*Motivation*	*Expected outcome*
Number of matches	*Case 1*	23	9	10
	Case 2	7	8	1

Table 29: Number of matches in cases 1 and 2 divided over the three constituents of the Epic template.

Overall the most matches can be found in the situation constituent, although this may not be desirable, as is discussed in Chapter 6.1.

Even though they are not included in the naming conventions for modules, adjectives can sometimes be crucial to establishing a linguistic link. In similar fashion, using a process of elimination when establishing links is essential to determining as many unique matches as possible. In both case 1 and 2, the semantic frame 'using resource' was observed. This frame specifies a purpose for the used resource, meaning that the benefit constituent of the US becomes mandatory. In all USs that used the frame, a benefit was mentioned. An interesting obstacle is the mapping of behavioral USs, since they do not describe functions. Therefore, only USs that describe requirements for functions can be accurately mapped to features. To complicate matters further, it is sometimes possible to relate a behavioral US to more than one feature. In case 2, the feature diagrams needed

to be recovered from the GUI. This leads to more (atomic) features than were described in the USs.

Findings that were observed once are not discussed any further, especially because most of them have been explained in the previous chapters already. The only exception is made for finding three in the second case, as it directly contradicts a hypothesized cardinality in the RE4SA model. Finding three in case 2 is stated as follows: "*There is not necessarily a one-to-one relationship between USs and features.*" While it became clear that behavioral USs cannot be mapped to a single feature or any feature at all, this can also be the case for USs that describe a function according to case 2. This would result in a one-to-many relationship between USs and features. On the other hand, the quality framework for USs specifies that a US should describe a requirement for one feature only, so the question is which relationship is desirable. Another question is whether practice should adapt to research or vice versa.

To summarize the quantitative results of two case study analyses, three tables and their descriptions are provided. Firstly, to recap which artifacts were studied, table 30 provides a selection of descriptive statistics in terms of the Epics, USs, modules and features included in the two cases.

	Numbers				**Coverage**	
	Epics	*USs*	*Modules*	*Features*	*Modules*	*Features*
Case 1	31	96	31	N/A	100%	N/A
Case 2	12	96	12	139	100%	69,8%

Table 30: Descriptive statistics of the first two cases.

In this table the number of features is only concerned with atomic features. Since only one feature diagram was included in the documentation of the first case, the number of features could not accurately be determined, the same is true for the feature coverage. The coverage is determined by calculating how many modules or features were covered by Epics or USs, respectively. Therefore, the percentage is only concerned with modules or features, not stories. According to the table, modules are more frequently covered by Epics than features by USs. The lack of feature coverage in case 2 can be explained by the fact that not all features are explicitly described in requirements, such as requirements concerned with design and navigation.

The coverage of modules and features does not provide any insight into whether they could linguistically be related to their respective Epics and USs. Table 31 shows how many Epics and USs could be related to modules and features respectively, per case. In addition, the table also shows how many linguistic matches could be identified per case and how many of those were unique and thus a linguistic link.

	Epics			**USs**		
	Relations	*Matches*	*Links*	*Relations*	*Matches*	*Links*
Case 1	31	30	19	10	9	9
Case 2	12	11	11	87	77	48*

Table 31: Statistics of the linguistic relationships in the two cases.

The asterisk indicates that the remaining 29 matches were not necessarily common, but that for four USs there was no relation, so no possible link, for another five the match was simply not unique and the rest are behavioral USs, which were not included in the identification of unique matches.

Finally, the relations, matches and links in the two cases are presented as percentages of the total, or of the number of relations or matches, shown in table 32. All percentages are to the nearest decimal.

	Relations % of total	**Matches % of total**	**Matches % of relations**	**Links % of total**	**Links % of relations**	**Links % of matches**
Epic	100	95,3	95,3	69,8	69,8	73,2
US	50,5	44,8	88,7	29,7	58,8	66,3

Table 32: Percentage of total relations, matches and links in the two cases.

It is important to note that these percentages were calculated from the RE perspective, meaning they consider the total number of Epics and USs, not the total number of modules and features. In addition, the relations, matches and links found in case 3 are not included. For results on the relationship between RE and SA it is better to refer to table 30. The number of relations between USs and features compared to the total number of USs is heavily influenced by the fact that a limited number of USs in case 1 could be related to features, due to the lack of feature diagrams. Therefore, it is more insightful to consider the matches and links, which are not as affected by the lack of features in the first case.

6.1 Story Quality

Throughout the linguistic analyses, the quality of the Epics and USs was briefly mentioned on several occasions, but never discussed in detail. Many aspects of the story formulation can be examined, so to keep it succinct the stories are evaluated using the QUS framework (Lucassen et al., 2016a). As became apparent while relating USs to features, the atomic criterion was violated more than once, as oftentimes the action constituent contained the word 'and'. To avoid such violations it would be better to split these USs into two and thus not use the word 'and'. Strictly speaking, the minimal criterion was not met in all USs either, since several USs contain two roles. However, this may not be a bad approach, because splitting these USs would have lead to more duplicates, resulting in a violation of the unique criterion. Other factors that play a role are more closely related to the architecture than the USs in and of themselves. For instance, it would be beneficial to always start the action constituent with an action verb that actually describes some task or activity. In case 2, verbs such as 'know' and 'have' were used, which are not specific enough when deriving a feature name or linking to a feature. Circling back to the atomic criterion, if a US describes a selection, options, choices or anything of the sort, with more than two possibilities, it would be better to include the possibilities as additional information. In case of only two possibilities it is feasible to write two USs, but when describing a feature that allows a user to select a language, you may have over fifty language options. In such cases, it would be more efficient to just describe the composite feature (i.e. 'select language') and not describe every language option separately. A more questionable comment on the US quality is that not all USs focus on user interaction or functions of the system, but rather also included behavioral requirements. Perhaps it would be desirable to make a distinction between USs that describe actions the user takes using the system and the system's responses and behavior. Finally, in case 2, the template was often changed. The action constituent template was modified 32 times, 23 times 'to' was left out and nine times 'want to' was omitted. In the benefit constituent template 'that' was removed once. In the action constituent modifications of the template may influence the ability to derive feature names, since 'want to' enforces the use of a verb.

Furthermore, it is difficult to identify any particular words that indicate that a US describes a behavioral requirement. Oftentimes, according to case 2, these USs contain the verb 'have', for instance: "*As a researcher, I want to have a simple Yoda-conform dialogue for restoration of a revision* [...]". The feature that allows for revisions to be restored is described in a different US. Similarly, the verb 'to be' is frequently present in "behavioral USs", as is any reference to the system in question and the omission of 'to' in the action constituent of the template. However, these four patterns can also be observed in USs that clearly describe a feature. In order to develop a method for automatically identifying behavioral USs, additional research is required. As of yet, this remains a manual assessment.

The quality of the Epics is more difficult to evaluate, since no framework exists and the only guidance that is provided is included in the explanation of how to write an Epic. However, based on whether the Epics were any use in regards to the architecture and how accurately they could be linked to modules, some guidelines can be defined. Firstly, for the (problematic) situation constituent, it is probably best if a trigger, an incoming message or the result of a completed task is stated here. Mostly because this would facilitate the modeling of input flows in the functional architecture. In addition, this would also provide a better introduction to the motivation. Similarly, the description of ongoing activities, such as "when I am working..." or "when I am managing...", should be avoided. Phrases like these do not describe a situation or problem in detail and are also not helpful when linking Epics to modules. Secondly, the motivation constituent should focus on the activity that solves the aforementioned problem or is the correct response to a particular situation. However, it should not be too detailed or specific, since it should categorize multiple USs

and it might be useful to leave room for extensions in the future. Thirdly, the expected outcome constituent should be formulated as output, so the result of the action and the solution to the original problem. Another good practice would be to try to formulate the outcome in such a way that it can serve as the input or trigger for another Epic or at least leads up to the (problematic) situation of another Epic. Finally, a guideline that is related to all three constituents, is to avoid circular reasoning. An example of such reasoning is: "*When I receive an email, I want to write a response, so I can reply to my email.*"

7 Discussion

In this chapter, the benefits as well as the limitations of the research are discussed. Future research directions and hypotheses are presented at the end of this chapter. Note that all information regarding case 3 was omitted in this chapter.

7.1 Benefits

Two novel approaches were utilized to extract architectural information from requirements documentation: derivation of feature names from USs and the identification of semantic frames in USs. To start with the former, while only a preliminary and small-scale experiment was conducted, the results seem promising. More than half of the derived feature names in case 2 were considered useful and/or relevant. In addition to this sufficient quality, the formulation of the feature names was easy and not at all time consuming. If the USs are PoS tagged, it takes approximately ten seconds to extract the first verb and first (compound) noun and transform them to the template "action verb noun" by hand. The activity can be made more efficient by automating it, although in some cases the extracted verb needs to be put into a different tense or person, which is a bit more challenging. If this linguistic structure proves to be applicable in USs in other contexts and other cases, automatically deriving such feature names is quite a straightforward technique and the effort of transforming the verbs would be worth it. Finally, deriving feature names from USs in this way facilitates (requirements) traceability, since the terms used in the feature names can be easily linked to the USs. Secondly, it became apparent during the analysis phase that additional information about architectural components and their relationships can be extracted from USs by considering semantic frames, although their use has not yet been tested. What has also been observed is that semantic frames can prescribe the need for a benefit constituent in a US.

Another asset of this research is the experience gained with architecture recovery (specifically, FAMs and feature diagrams) from an implemented system utilizing the GUI. Not only is extracting features from the GUI a relatively quick and simple task, it is also hypothesized to result in a more complete overview of features than when considering the requirements only. Moreover, it should theoretically provide a more accurate representation than when using the requirements as input, since it is concerned with the implemented architecture rather than the intended architecture. Other experience gained includes the formulation of Epics. For one case, Epics needed to be written based solely on the USs provided. This Epic formulation practice aided the process of devising guidelines and tips for writing Epics, similar to the QUS framework (Lucassen et al., 2016a), as discussed in Chapter 6.1.

Finally, this study benefited from use of implemented systems and real-life documentation. Not only does this ensure that the analyses conducted can be applied to industry projects, it also shows that the approaches are generalizable to more than one project or context. The cases were provided by different companies and provided real-world materials. Additionally, the real-world examples allowed for a quality assessment of USs in practice, which has shown that companies are able to provide USs suitable for academic research.

7.2 Limitations

The limitations of this research are discussed according to the four aspects of validity for case studies: construct validity, internal validity, external validity and reliability (Wohlin et al., 2012; R. K. Yin, 1981).

Firstly, in relation to construct validity, no clear threats can be identified, due to following the prevention tactics. Multiple sources were used to gather data for analyses. Moreover, a chain of evidence was established by explicitly describing the various steps of data preparation and subsequent analysis. Additionally, since all analyses were performed by one researcher, there is no risk of different interpretations of data and results.

Secondly, several threats to the internal validity can be observed. There is a chance that the degree to which linguistic matches and links could be identified is affected by other factors. For instance, the requirements artifacts used are of sufficient quality (if not high quality). This quality mainly refers to adherence to the template and the main quality criteria in the context of USs. There are two options to remedy this, either linguistic links assume that artifacts are of sufficient quality or the approach needs to be adjusted in order to facilitate the use of artifacts with questionable quality too. The latter is complicated if not nigh impossible. All manner of factors

would need to be taken into account and the basic principles on which the approach is based would be lost. Secondly, it is probably better to produce artifacts of higher quality for organizations regardless of the analyses they wish to perform, making the former option more attractive. To summarize there is a trade-off between accuracy and precision. When identifying linguistic links it is arguably better to be roughly right than precisely wrong. Moreover, the linguistic terms used in the artifacts are influenced by the writing style and experience of the requirements engineer(s) or researcher in case of reconstructed artifacts. The latter means that they were formulated with knowledge of and experience on what should and should not be included in an Epic. Be that as it may, the quality of the Epics was barely considered in the analysis, since only their words were used. These words were based only on what was already written by the owners of the documentation, so this should not introduce any bias. Similarly, the architecture is dependent on the implementation of developers. Developers may have used their personal experiences and styles to implement the system, which affects the as-is architecture and can cause it to diverge from the to-be (or intended) architecture. Furthermore, it is possible that the provided documentation was non-exhaustive, for example due to non-documented interviews, meetings or other personal communications about the implementation of the requirements. Finally, the choice of granularity in the context of features may have impacted the feature coverage. If the developers used a different definition or granularity for the term feature than is applied in this study, a discrepancy may occur.

Thirdly, on the topic of external validity, there are few threats. In an attempt to improve the generalizability, multiple cases were examined, replicating the same process each time. Moreover, cases 1 and 2 originated from different companies or institutions, so two different systems were investigated. The only threat to generalizability is the fact that the two companies and institutions were based in the Netherlands.

Finally, threats to reliability are discussed. The preparation and analyses activities were documented in detail, following a protocol, and little interpretation was required. Whenever interpretation was needed, the rationale was explicitly explained. The naming of architectural elements and the formulation of Epics may be dependent on the researcher, but as many words and names were extracted from the provided documentation and implemented system to prevent this threat. In addition, the reconstructed and recovered artifacts were mostly validated by a stakeholder of the system. Another activity in which replications of this research may differ is the process of establishing linguistic matches and subsequent links, since the Epics and modules and USs and features were manually linked beforehand. So, the mapping may vary between researchers. Following this approach, during linguistic analysis, exclusively terms shared between the artifact were considered. However, if all nouns, verbs and adjectives are utilized to form matches and not just the nouns, verbs and adjectives the US and its respective feature have in common, it is possible that more matches can be identified and that there are fewer linguistic links (unique matches). One can hypothesize that the number of possible links can be restricted by prohibiting links between Epics/modules and only allowing links within Epics/modules. On the other hand, this does require that the Epic/module categorization is done correctly. Statistically speaking, there are not enough data points to produce any conclusive results, but due to the exploratory nature of the study, this was to be expected. However, this research does provide interesting findings and suggestions for future work. Finally, one minor drawback is the probability of human error in the analyses. In relation to the former, most of the analysis, except for PoS tagging, was done manually and can therefore introduce bias.

7.3 Future Research

The future research directions can roughly be divided into four categories: the RE4SA model, the applications of this model, the linguistic relationship between requirements and architectures and hypotheses for individual artifacts. At the end of the chapter, an overview of the discussed hypotheses can be found (table 33).

The hypothesized relationship between the Epics and modules is one-to-one, but according to the analysis, this is not always the case for USs and features in practice. Frequently, more than one US describes the same feature or two features are described in one US. In order to accurately support requirements traceability, it is important to either discern what the cardinalities are like or prescribe guidelines for what they should be like. The expected cardinalities between the two levels of abstraction are mostly accurate according to this study. All USs belong to one Epic and features belong to one module, except for duplicate features, which can be included in

multiple modules. It could also be beneficial to remove duplicate USs. In some cases, the USs that describe the same features were nearly identical, so including only one of them in the requirements documentation would have been enough. To avoid confusion about whether a US has already been implemented/covered in a sprint or not, USs can be checked for duplicity based on a similarity score. Since USs can describe the same functionality without being identical, looking for perfect matches in terms of words is not sufficient. However, a similarity score can handle non-identical yet similar USs. This similarity score should also take into account synonyms and partial matches (as described in the introduction to Chapter 5).

As of yet, the RE4SA model exclusively relates the requirements to the software architecture, but other software artifacts exist, such as source code, acceptance tests/test scripts, user manuals and release planning among others. Ideally, all software artifacts are linked and requirements traceability permeates throughout all of them. Therefore, theoretical as well as practical research should examine whether the RE4SA model can (and should) be applied to additional software product management activities. An example of how the RE4SA model can be extended is by relating modules and features to classes and methods in source code.

The objective of this research was to discover whether there is a linguistic relationship between Epics and modules and USs and features, respectively. However, establishing these relationships was done manually so far, as was discussed in the limitations previously (Chapter 7.2). In preparation for the analysis, the artifacts were linked and then their linguistic terms were compared. This means that the artifacts may contain more relevant terms that were excluded during this research, which may lead to additional linguistic matches with other artifacts that were ignored. In future research, the linguistic matches and subsequent links should be established automatically based on the results of this research. However, the artifacts should not be related manually beforehand in order to determine how beneficial this approach is and whether unique matches can still be found. To limit the number of possibilities, matches should only be observed within an Epic or module, as opposed to between.

Earlier in this chapter the perceived benefits of deriving feature names from USs were discussed, however, these can be extended by hypothesized advantages. For instance, the derived feature names are expected to prevent misunderstandings and misinterpretations among stakeholders, especially between requirements engineers and software architects and developers, by using similar, if not the same, terminology. Furthermore, it is hypothesized that the approach saves time, since only (approximately) two words need to be extracted from a US. In addition, even if the derived feature name is too generic or inaccurate, as was sometimes the case in the assessment, it provides architects and developers with a starting point. Arguably, it is easier to reason about whether a feature name is right or wrong than to ascertain and formulate a feature from scratch.

Similarly, semantic frames were identified in an attempt to analyze linguistic structures. As was stated before, semantic frames can help requirements engineers to determine whether they should use a benefit constituent in their USs. Hypothetically, semantic frames can improve the communication between requirements engineers and software architects, as semantic frames can provide additional information about what the (technical) solution should look like. Likewise, semantic frames provide more rules and/or guidelines for architectures. The question is whether architects need this additional information to be made more explicit or whether it is self-explanatory and does not require any specific attention.

The analysis of linguistics matches has shown that the system artifacts often have words in common. Especially in the architecture this can help to group similar functionality. Therefore, one can hypothesize that new features can be positioned in an implemented/existing system architecture by taking into account linguistics. A simplified example is presented using figure 35. The white boxes represent features that have already been implemented in the system. The gray boxes indicate an extension of the existing functionality, so new features.

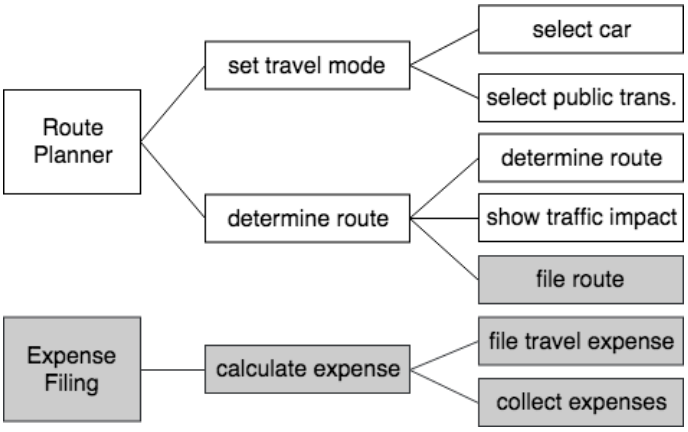

Figure 35: Example of a feature diagram of a navigation app.

The new feature "file route", can easily be positioned in the 'Route Planner' module and subsequently the composite feature "determine route", since they all have the word 'route' in common, while the other module and other features do not contain this word. Similarly, the other two new atomic features contain the word 'expense' and can therefore be positioned in the 'Expense Filing' module. This example, however, concerns an imaginary system, so industry research is necessary to confirm whether this is a feasible approach.

Furthermore, behavioral requirements were excluded from analysis in this study, since they cannot always be accurately linked to functionality in the architecture (features). In the instances where they can be linked, they are superfluous, because there is already a functional requirement in place to describe the functionality. Still, behavioral requirements provide additional, relevant information for development, so how can architectural information be extracted from them? Behavioral requirements describe how data is handled, as well as responses and behavior performed by the system. Therefore, it would make sense to use an architectural viewpoint that reflects these characteristics. In the context of data, the information viewpoint can be used, as it "*describes the way that the system stores, manipulates, manages, and distributes information*" (Rozanski & Woods, 2011). To model behavioral requirements concerning data, ERDs and class diagrams can be used. Similarly, the concurrency viewpoint "*describes the concurrency structure of the system and maps functional elements to concurrency units [...] and how this is coordinated and controlled*" (Rozanski & Woods, 2011). Statecharts, sequence diagrams and Petri nets allow for modeling system responses and behaviors. The latter seem a better option, since they provide more detail and can be analyzed on quality. Theoretical research as well as case studies should determine if information and concurrency viewpoints can be utilized for modeling behavioral requirements and, if this is the case, whether they provide any valuable contributions or benefits.

An unrelated hypothesis is that architecture models, such as the functional architecture and feature diagrams, can be utilized to assess the usability of a system. The functional architecture, for instance, can help to identify the presence of 'God elements'. Such an element has the majority of responsibilities in the system, making it not just complex to understand and maintain, but may also lower performance, reliability and scalability (Rozanski & Woods, 2011). Feature diagram assessment serves a different purpose, namely determining the quality of the GUI. Seffah et al. have defined 25 usability criteria for software systems (Seffah, Donyaee, Kline, & Padda, 2006). Three of these criteria can potentially be assessed by analyzing feature diagrams, namely:

1. Minimal action: minimal number of steps required to carry out a specific task.
2. Consistency: uniformity of elements of the GUI.
3. Navigability: ability to navigate the application in an efficient way.

The hypothesis is that feature diagrams can support the assessment of these usability aspects through relatively simple statistics. Firstly, whether the application requires minimal action can be determined by carrying out a use case and counting how many features in a feature diagram need to be used in order to complete the task. Similarly, navigability can be evaluated by analyzing the feature diagrams to see how easy it is to get from one feature (or one feature diagram) to another. Finally, consistency can be assessed based on the degrees and depths of feature diagrams. Degrees specify the number of child features for each parent feature. If this number is more or

less equal for all parent features in all feature diagrams, you can assume consistency. The depth refers to how many steps it takes to get from the starting point (a module) of a feature diagram to a specific atomic feature. Again, if the number of steps is (nearly) equal across feature diagrams, their structure can be considered consistent. For larger systems, it may be useful to consider the standard deviation in order to determine consistency, since it is reasonable that not all degrees and depths are equal. Moreover, for some features it is not uncalled for that they require more effort to reach. An example of such a 'rare usage' feature is an airbag in a car. It is important that is it there, but is not used often. In software, a similar feature would be deleting an account. It does not happen frequently in daily usage, but it is still required. However, such a feature may be positioned at a lower depth than other, more frequently used features. Finally, the number of atomic features (where the degree is zero), per (sub) module can help to recognize bad code smells. For instance, when a module consists of a much higher number of atomic features than other (sub) modules, this could be an indication that too much functionality and/or responsibility is contained in one module. Again, this is referred to as a 'God element' and can make the system more difficult to maintain.

During the feature diagram recovery process, it became apparent that there are no syntactic elements that facilitate modeling conditionality in feature diagrams. Conditionality in feature diagrams refers to the fact that some features can only be accessed or used if another feature was used prior to the first one. Visualizing conditionality is necessary for accurately modeling the system's features and might help to document user interaction or user manuals. For instance by showing what a user needs to do in order to enable a certain other feature. In addition, it could, hypothetically, support release planning, since it makes dependencies among features (that are perhaps not in the same 'branch' of the feature tree) explicit. The only syntax that is somewhat similar to conditionality is the 'required' relationship. This, however, does not take a sequence of events or actions into account, but simply states that a feature can only exist or be present if another exists or is present. An example is that a feature that allows a user to take a picture with their smartphone requires that this smartphone has a camera lens. This required relationship covers a hardware requirement, while the conditionality issue in the feature diagrams was only concerned with software. The 'required' relationship can only be utilized if its meaning is extended with an order or sequence of actions that enable and/or disable features. Finally, it seems that feature diagrams recovered from the GUI are more extensive or detailed than when only taking the front-end features explicitly described by USs into consideration. The cause of this completeness discrepancy is that the options in an alternative structure (drop-down menu) are oftentimes not covered in USs individually and that features related to navigation are not explicitly included as requirements either.

In addition to these feature diagram analyses, the small-scale feature SLR is still a promising and interesting study. In future research, this SLR should be expanded, analyzed in more detail and properly finished.

The aforementioned future research directions have been formulated into hypotheses and are presented in table 33. They have been divided into five categories: the structure of the RE4SA model, applications of the RE4SA model, linguistics, architecture and feature diagrams. Finally, the research approaches with which the hypotheses will be tested are included.

Hypothesis	Research approach
RE4SA model	
A one-to-one relationship between USs and features is desirable.	Case study
USs and features belong to one Epic or module respectively, the only exception being duplicate features.*	Case study
Applications of RE4SA	
The RE4SA model can be extended and applied to other software development activities, such as testing and release planning.*	Literature & case study
Epics and modules and USs and features can be mapped to classes and methods in source code respectively.	Literature & case study
Linguistics	
Duplicate USs can best be identified by using a similarity score.	Case study
Linguistic links can be established automatically, if the Epic/module categorization of USs and features is taken into account.	Case study
Feature names can be derived from USs by extracting the first verb and the first (compound) noun.	Case study
Deriving feature names is more efficient than creating them from scratch and improve communication between stakeholders.	Expert interview & case study
New features can easily be positioned in an architecture by considering linguistics.*	Case study
USs can be classified as either functional or behavioral USs based on their formulation.	Literature study & case study
Architecture	
Semantic frames identified in requirements documentation provide useful/relevant additional information to software architects.	Expert interview & case study
Behavioral requirements can be satisfied in information and/or concurrency viewpoints of the software architecture.	Literature & case study
Modules that contain much more atomic features than other modules contain too much functionality/have too many responsibilities.	Literature & case study
Feature diagrams	
Feature diagrams can be utilized to assess the usability of a GUI, based on their depths and degrees.	Literature & case study
Feature diagrams would more accurately represent a system if conditionality was taken into account.	Case study
Recovering feature diagrams from the GUI results in the most complete and accurate collection of features of the front-end of a system.	Case study

Table 33: Hypotheses for future research with envisioned research approaches.

Hypotheses marked with an asterisk indicate that they were also included in the paper submitted to RE@Next!, which can be found in Appendix D.

8 Conclusion

In this research, the linguistic relationship between RE and SA, based on the RE4SA model, was analyzed in two phases. In the first phase, literature research was done to provide a theoretical background to the case study done in the second phase, as well as to investigate the terms 'feature' and 'functionality'. Subsequently, a sequential, direct replication multiple-case design was used to research the main RQ as well as the sub-RQs presented in Chapter 2 in detail. The cases were analyzed on the following components: dependencies, linguistic relationship between Epics and modules, linguistic relationship between USs and features, semantic frames and synonyms and homonyms. Additionally, based on findings discovered during the analyses, a small experiment on the derivation of feature names and the modeling of behavioral USs has been performed. Finally, the results were used to determine directions for future research, of which some have been formulated into hypotheses.

While not explicitly included in the RQs, one of the objectives of this research was to assess the validity of the RE4SA model. In cases 1 and 2 it became apparent that there is indeed a relationship between Epics and USs and modules and features, respectively (results from case 3 are left out due to confidentiality). During the preparation of the analyses, it was possible to categorize USs in Epics and features in modules. The cardinalities of these relationships are discussed later, in sub-RQs four and five. Behavioral USs, however, introduce obstacles in both relating USs to features and categorizing USs in Epics. In case of the former, it can be difficult to ascertain which feature a behavioral US describes or whether they describe a requirement for any specific feature at all. It was also observed that some behavioral USs can be linked to more than one feature, since they require multiple features to exist or be used in order to satisfy a requirement. The biggest challenge that needs to be overcome when applying the RE4SA model to a product or system is that, according to this study, only one of four artifacts is available, namely the USs. The Epics need to be reconstructed based on USs and other requirement documentation, while the modules and features need to be recovered using the GUI, source code and architecture documentation.

To conclude, the sub-RQs and MRQ are answered and discussed in order, based on the findings and results presented in Chapter 6. Firstly, sub-RQ 1 posed the following question: "*What is understood as a feature in literature and in practice?*" In the context of literature, no definitive answer can be found. Many definitions of the term exist and one is not necessarily better or more accurate than the next. The meaning of the term is largely dependent on its purpose, be it for requirements, architecture, development, modeling, target audience or otherwise. To complicate matters further, the viewpoint can influence the definition. In this research a distinction was made between problem-oriented (abstract) and solution-oriented (technical). The only aid that can be provided when selecting a definition is the popularity of the definition, the reputation of its authors, the research field and/or context, viewpoint and intended audience. Although even then multiple options may be available. In practice, the answer to this question in the case study is twofold. On the one hand, features are considered GUI elements the user of the system can interact with. On the other hand, features are the requirement for specific functionality described in a US and the former and the latter are not always one and the same. The difference between the two is best illustrated by figure 36.

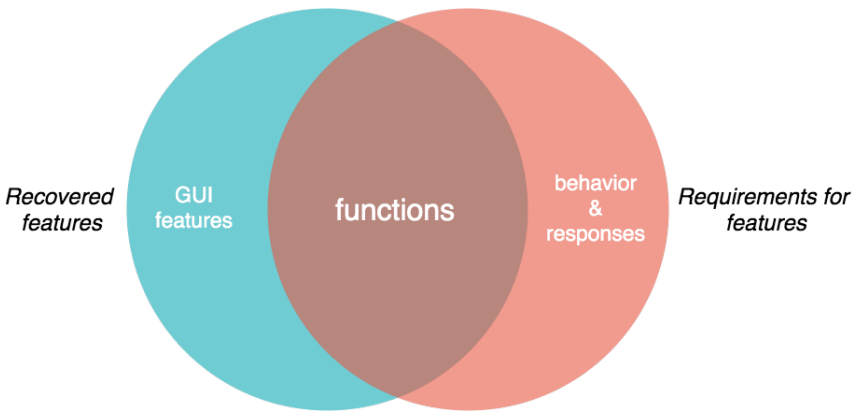

Figure 36: Visualization of the differences and overlap between features recovered from the GUI and features described in requirements.

Both recovered features and features described in the requirements are concerned with functions. However, the former does not explicitly include behavior and responses of the system and the latter tends not to cover all GUI elements.

Secondly, sub-RQ 2 is stated as follows: "*What is understood as functionality in literature and in practice?*" According to literature, functionality is described in functional requirements in the context of RE and are contained in architectural components in the context of SA. Unfortunately, the term 'architectural components' is not very specific, but since both modules and features are considered architectural components, functionality is present on both levels of abstraction in the architecture. On the other hand, since functional requirements are described in USs, which are on the lowest level of abstraction, which describe features, it seems that functionality is mostly addressed in USs and features. In practice, the meaning of functionality is mostly the same, although a distinction can be made between implicit and explicit functionality. Explicit functionality refers to functional requirements and how they are satisfied by architectural components in the SA. Implicit functionality, however, is described in USs as behavior or responses of the system, but such behavioral USs cannot be translated into a feature. Instead, a different architectural view is required in addition to the functional view. Hypothesized views are information and/or concurrency views. As a small experiment on the feasibility of this approach, a selection of behavioral USs from case 2 were used to model a sequence diagram. Furthermore, not all features are explicitly described by USs. For instance, features related to the design of the GUI (such as buttons for navigation the system) cannot be mapped to USs. In a sense, these can also be considered implicit functionality, since they are included based on experience or common sense and not covered in the requirements documentation.

Thirdly, sub-RQ 3 asked the question: "*Is there a linguistic relationship between the names and descriptions of Epics and USs and modules and features?*" The short answer is yes. In total, 95,3% of Epics in cases 1 and 2 could be linguistically matched to a module. Note that the findings from case 3 are omitted here, since they are confidential. For USs the percentage is lower at 50,5%, which can be explained by the lack of feature diagrams in case 1. A more insightful number is the percentage of linguistic matches that could be identified in the mapped artifacts, 95,3% and 88,7% for Epics and USs, respectively. The only concern here is that the mapping was done manually beforehand and not automatically based on linguistics. However, this does not disprove the evidence for the linguistic relationships between Epics and modules and USs and features. The linguistic relationship between artifacts is most prevalent in nouns, verbs and, to a lesser extent, adjectives. To be more specific, both compound and 'regular' nouns are used in linguistic links. In the case of verbs, these are mostly action verbs and oftentimes the present form of verbs. As was stated previously, no discernible patterns could be observed across the two cases. However, the largest dataset, case 2, has a clear trend of using the first verb and first noun in linguistic matches and subsequent links.

Fourthly, the following was asked in sub-RQ 4: "*Are the dependencies between Epics and USs the same as their corresponding modules and features?*". In most cases this is true, but there have been some exceptions. In case 2, eight times USs could not be mapped to features, since the Epic-US categorization did not match the module-feature categorization. It is unclear which of the two categorizations is correct in these cases. However, it seems more likely that the module-feature categorization deviates, since the intended and implemented architectures are not necessarily identical.

The final sub-RQ was stated as follows: "*Is there a one-to-one relationship between USs and features?*" Not taking into account behavioral USs, the answer is yes in most cases. However, in case 2, there were seven (functional) USs that described a requirement for more than one feature and in case 1 there were also some instances in which there was a one-to-many relationship. Whenever the words 'and' or 'or' are used, the solution is simple, namely split the US into two USs. In other circumstances, this may not be a desirable approach. For instance, if a US describes a composite feature (e.g. personal details), it technically describes more than one atomic feature, since it encompasses all atomic features contained in the composite feature 'personal details'. It is possible to split the US in such a way that all atomic features are described explicitly, but that would lead to many, similar USs, that can be summarized by the one mentioned earlier.

Finally, the MRQ was: "*Can linguistic links be identified between the two domains of RE4SA using application reconstruction?*" In both case 1 and 2 it was possible to identify linguistic matches and subsequently establish linguistic links between RE and SA (again, results from case 3 are omitted due to confidentiality). The exact nature of these relationships, however, is not

yet defined. Some progress has been made by analyzing which types of words can be matched and where these words are positioned, but the results are not fully conclusive. Twice the Epics needed to be reconstructed and the functional architecture recovered. While both activities can be time-consuming, they are quite simple and valuable for linguistic analysis.

Acknowledgements

First and foremost, I want to thank Prof. Dr. Sjaak Brinkkemper and Dr. Fabiano Dalpiaz for their guidance, comments, feedback and lengthy discussions. Without their time, effort and input, this research would not have made half as much sense. Secondly, a special thanks to Remmelt Blessinga, the Yoda development team and the anonymous product manager of the third case for contributing their documentation, time and access to their systems to this research. Finally, I would like to thank Martijn van Vliet, Lientje Maas, Mitchell van Winsum, all the students of office 5.86 and the Grimm research group for their immeasurable support, valuable contributions and scintillating conversations.

References

Adams, P. (n.d.). The dribbblisation of design. Retrieved from https://www.intercom.com/blog/the-dribbblisation-of-design/ (Accessed: 11-12-2018)

Ali, N., Baker, S., O'Crowley, R., Herold, S., & Buckley, J. (2018). Architecture consistency: State of the practice, challenges and requirements. *Empirical Software Engineering*, *23*(1), 224–258.

Andam, B., Burger, A., Berger, T., & Chaudron, M. (2017). Florida: Feature location dashboard for extracting and visualizing feature traces. *In: Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems*, 100–107.

Androutsopoulos, I., Lampouras, G., & Galanis, D. (2013). Generating natural language descriptions from owl ontologies: the naturalowl system. *Journal of Artificial Intelligence Research*(48), 671–715.

Apel, S., Batory, D., Kästner, C., & Saake, G. (2013). *Feature-oriented software product lines: Concepts and implementation*. Springer, Berlin, Heidelberg.

Apel, S., & Kästner, C. (2009). An overview of feature-oriented software development. *Journal of Object Technology*, *8*(5), 49–84.

Apel, S., Lengauer, C., Batory, D., Möller, B., & Kästner, C. (2007). An algebra for feature-oriented software development. *Department of Informatics and Mathematics, University of Passau, Tech. Rep. MIP-0706*.

Baker, C., & Ruppenhofer, J. (2002). Framenet's frames vs. levin's verb classes. *In: Annual Meeting of the Berkeley Linguistics Society*, *28*(1), 27–38.

Batory, D. (2004). Feature-oriented programming and the ahead tool suite. *In: Proceedings of the 26th International Conference on Software Engineering*, 702–703.

Batory, D., Benavides, D., & Ruiz-Cortes, A. (2006). Automated analysis of feature models: challenges ahead. *Communications of the ACM*, *49*(12), 45–47.

Batory, D., Sarvela, J., & Rauschmayer, A. (2004). Scaling step-wise refinement. *IEEE Transactions on Software Engineering*, *30*(6), 355–371.

Berger, T., Lettner, D., Rubin, J., Grünbacher, P., Silva, A., Becker, M., ... Czarnecki, K. (2015). What is a feature?: a qualitative study of features in industrial software product lines. *In: Proceedings of the 19th International Conference on Software Product Line*, 16–25.

Berry, D., Kamsties, E., & Krieger, M. (2003). From contract drafting to software specification: Linguistic sources of ambiguity (version 1.0). *Technical report, University of Waterloo*.

Bettencourt, L., & Ulwick, A. (2008). The customer-centered innovation map. *Harvard Business Review*, *86*(5), 109–116.

Blessinga, R. (2018). Designing the automated greenhouse: Matching requirements and architecture for startup product specification using epic stories. *Unpublished master thesis, Utrecht University*. Retrieved from https://dspace.library.uu.nl/handle/1874/374865

Bosch, J. (2000). *Design and use of software architectures: adopting and evolving a product-line approach*. Pearson Education.

Bosch, J. (2004). Software architecture: The next step. *In: European Workshop on Software Architecture*, 194–199.

Bouillon, E., Mäder, P., & Philippow, I. (2013). A survey on usage scenarios for requirements traceability in practice. *In: International Working Conference on Requirements Engineering: Foundation for Software Quality*, 158–173.

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

Brinkkemper, S. (n.d.). Impact forecasting. *Internal Documentation, Utrecht University*.

Brinkkemper, S. (2018). RE4SA. *Internal Documentation, Utrecht University*.

Brinkkemper, S., & Pachidi, S. (2010). Functional architecture modeling for the software product industry. *In: European Conference on Software Architecture*, 198–213.

Canfora, G., Di Penta, M., & Cerulo, L. (2011). Achievements and challenges in software reverse engineering. *Communications of the ACM*, *54*(4), 142–151.

Cechticky, V., Pasetti, A., Rohlik, O., & Schaufelberger, W. (2004). XML-based feature modelling. *In: International Conference on Software Reuse*, 101–114.

Chen, K., Zhang, W., Zhao, H., & Mei, H. (2005). An approach to constructing feature models based on requirements clustering. *13th IEEE International Conference on Requirements Engineering (RE'05)*.

Chen, P. (1976). The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, *1*(1), 9–36.
Chikofsky, E., & Cross, J. (1990). Reverse engineering and design recovery: A taxonomy. *IEEE Software*, *7*(1), 13–17.
Christensen, C., Anthony, S., Berstell, G., & Nitterhouse, D. (2007). Finding the right job for your product. *MIT Sloan management review*, *48*(3), 38–47.
Christensen, C., Hall, T., Dillon, K., & Duncan, D. (2016). "know your customers'"jobs to be done"". *Harvard Business Review*, *94*(9), 14.
Cimiano, P., Mädche, A., Staab, S., & Völker, J. (2009). Ontology learning. *In: Handbook on ontologies*, 245–267.
Classen, A., Heymans, P., & Schobbens, P. (2008). What's in a feature: A requirements engineering perspective. *In: International Conference on Fundamental Approaches to Software Engineering*, 16–30.
Cleland-Huang, J., Gotel, O., Huffman Hayes, J., Mäder, P., & Zisman, A. (2014). Software traceability: trends and future directions. *In: Proceedings of the on Future of Software Engineering*, ACM, 55–69.
Cleland-Huang, J., Gotel, O., & Zisman, A. (2012). *Software and systems traceability*. Springer, Berlin, Heidelberg.
Cohn, M. (2004). *User stories applied: for agile software development*. Addison Wesley, Boston.
Cole, M., & Avison, D. (2007). The potential of hermeneutics in information systems research. *European Journal of Information Systems*, *16*(6), 820–833.
Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM*, *31*(11), 1268–1287.
Czarnecki, K., & Eisenecker, U. (2000). *Generative programming: methods, tools, and applications (vol. 16)*. Reading: Addison Wesley.
Czarnecki, K., Helsen, S., & Eisenecker, U. (2005). Formalizing cardinality `[U+2010]` based feature models and their specialization. *Software process: Improvement and practice*, *10*(1), 7–29.
Dalpiaz, F., van der Schalk, I., & Lucassen, G. (2018). Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and nlp. *In: International Working Conference on Requirements Engineering: Foundation for Software Quality*, 119–135.
Dit, B., Revelle, M., Gethers, M., & Poshyvanyk, D. (2013). Feature location in source code: a taxonomy and survey. *Journal of software: Evolution and Process*, *25*(1), 53–95.
Fairbanks, G. (2010). *Just enough software architecture: a risk-driven approach*. Marshall & Brainerd.
Fernández, D., Wagner, S., Kalinowski, M., Felderer, M., Mafra, P., Vetrò, A., ... Wieringa, R. (2017). Naming the pain in requirements engineering. *Empirical software engineering*, *22*(5), 2298–2338.
Fillmore, C. (1982). *Frame semantics*. Cognitive linguistics: Basic readings.
Fillmore, C., & Baker, C. (2001). Frame semantics for text understanding. *In: Proceedings of WordNet and Other Lexical Resources Workshop, NAACL*.
FrameNet data. (n.d.). Retrieved from `https://framenet.icsi.berkeley.edu/fndrupal/frameIndex` (Accessed: 09-01-2019)
Gacto, M., Alcalá, R., & Herrera, F. (2011). Interpretability of linguistic fuzzy rule-based systems: An overview of interpretability measures. *Information Sciences*, *181*(20), 4340–4360.
Gilb, T., & Finzi, S. (1988). *Principles of software engineering management (vol. 11)*. Reading, MA: Addison-wesley.
Glinz, M. (2007). On non-functional requirements. *In: 15th IEEE International Requirements Engineering Conference (RE 2007)*, 21–26.
Goldberg, A. (2010). Verbs, constructions and semantic frames. *Syntax, lexical semantics, and event structure*, 39–58.
Gregory, F. (1993). Cause, effect, efficiency and soft systems models. *Journal of the Operational Research Society*, *44*(4), 333–344.
Grüber, T. (1995). Toward principles for the design of ontologies used for knowledge sharing? *In: International journal of human-computer studies*, 907–928.
Guarino, N. (1997). Semantic matching: Formal ontological distinctions for information organization, extraction, and integration. *In: International Summer School on Information Extraction*, 139–170.
Guerra, S., Ryan, M., & Sernadas, A. (1996). Feature-oriented specifications.

Hirst, G. (2009). Ontology and the lexicon. *In: Handbook on ontologies*, 269–292.

Hofmeister, C., Kruchten, P., Nord, R., Obbink, H., Ran, A., & America, P. (2007). A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, *80*(1), 106–126.

Hosseini, M., Breaux, T., & Niu, J. (2018). Inferring ontology fragments from semantic role typing of lexical variants. *In: International Working Conference on Requirements Engineering: Foundation for Software Quality*, Springer, Cham, 39–56.

Hughes, D., Dwivedi, Y., Rana, N., & Simintiras, A. (2016). Information systems project failure–analysis of causal links using interpretive structural modelling. *Production Planning Control*, *27*(16), 1313–1333.

Jansen, N., & van Rhijn, J. (2018). utrecht architecture description language. *Internal Documentation, Utrecht University*.

Jeffries, R. (2001). Essential XP: card, conversation, confirmation. *XP Magazine*, *30*.

Kang, K., Cohen, S., Hess, J., Novak, W., & Peterson, A. (1990). Feature-oriented domain analysis (FODA) feasibility study. *Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.*.

Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., & Huh, M. (1998). Form: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, *5*(1), 143–168.

Kästner, C., Apel, S., & Kuhlemann, M. (2008). Granularity in software product lines. *In: Proceedings of the 30th international conference on Software engineering*, 311–320.

Khan, S., Greenwood, P., Garcia, A., & Rashid, A. (2008). On the impact of evolving requirements-architecture dependencies: An exploratory study. *In: International Conference on Advanced Information Systems Engineering*, Springer, Berlin, Heidelberg, 243–257.

Klement, A. (2013a). 5 tips for writing a job story. Retrieved from `https://jtbd.info/5-tips-for-writing-a-job-story-7c9092911fc9` (Accessed: 11-12-2018)

Klement, A. (2013b). Replacing the user story with the job story. Retrieved from `https://jtbd.info/replacing-the-user-story-with-the-job-story-af7cdee10c27` (Accessed: 11-12-2018)

Klement, A. (2016). Your job story needs a struggling moment. Retrieved from `https://jtbd.info/your-job-story-needs-a-struggling-moment-c03de87c6026` (Accessed: 11-12-2018)

Klement, A. (2018). *When coffee and kale compete*.

Krüger, J., Gu, W., Shen, H., Mukelabai, M., Hebig, R., & Berger, T. (2018). Towards a better understanding of software features and their characteristics: a case study of marlin. *In: Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems*, 105–112.

Kung, C., & Sölvberg, A. (1986). Activity modeling and behavior modeling. *In: Proc. of the IFIP WG 8.1 working conference on Information systems design methodologies: improving the practice*, 145–171.

Kühne, T. (2006). Matters of (meta-) modeling. *Software & Systems Modeling*, *5*(4), 369–385.

Lampouras, G., & Androutsopoulos, I. (2018). Extracting linguistic resources from the web for concept-to-text generation.

Lee, K., Kang, K., & Lee, J. (2002). Concepts and guidelines of feature modeling for product line software engineering. *In: International Conference on Software Reuse*, 62–77.

Levin, B. (1993). *English verb classes and alternations: A preliminary investigation*. University of Chicago press.

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 ParallelDistributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on*, 1–6.

Lo, S., & Chen, N. (2017). IEEE 42010 and agile process-create architecture description through agile architecture framework. *In: Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, 149–155.

Lucassen, G., Dalpiaz, F., van der Werf, J., & Brinkkemper, S. (2015a). Bridging the twin peaks: the case of the software industry. *In: Proceedings of the Fifth International Workshop on Twin Peaks of Requirements and Architecture*, 24–28.

Lucassen, G., Dalpiaz, F., van der Werf, J., & Brinkkemper, S. (2015b). Forging high-quality user stories: towards a discipline for agile requirements. *In: Requirements Engineering Conference*

(RE), 2015 IEEE 23rd International, 126–135.

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

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

Lucassen, G., Dalpiaz, F., van der Werf, J., Brinkkemper, S., & Zowghi, D. (2017). Behavior-driven requirements traceability via automated acceptance tests. *In: 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, 431–434.

Lucassen, G., van de Keuken, M., Dalpiaz, F., Brinkkemper, S., Sloof, G., & Schlingmann, J. (2018). Jobs-to-be-done oriented requirements engineering: A method for defining job stories. *In: International Working Conference on Requirements Engineering: Foundation for Software Quality*, Springer, Cham, 227–243.

Lüdeling, A., & Kytö, M. (2008). *Corpus linguistics: An international handbook*. Walter de Gruyter GmbH.

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., & McClosky, D. (2014). The stanford corenlp natural language processing toolkit. *In: Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 55–60. Retrieved from `https://stanfordnlp.github.io/CoreNLP/index.html` (Accessed: 29-01-2019)

Marcus, A., & Maletic, J. (2003). Recovering documentation-to-source-code traceability links using latent semantic indexing. *In: Proceedings of the 25th international conference on software engineering*, 125–135.

Marcus, M., Marcinkiewicz, M., & Santorini, B. (1993). Building a large annotated corpus of english: The Penn Treebank. *Computational linguistics*, *19*(2), 313–330.

Martens, S., Brinkkemper, S., & Dalpiaz, F. (2018). Matching of domain concepts to enable ontological traceability for software products. *Working paper, University of Utrecht*.

Miller, G. (1998). *Wordnet: An electronic lexical database*. MIT press.

Murphy, G., Notkin, D., & Sullivan, K. (1995). Software reflexion models: Bridging the gap between source and high-level models. *ACM SIGSOFT Software Engineering Notes*, *20*(4), 18–28.

Müter, L., Deoskar, T., Mathijssen, M., Brinkkemper, S., & Dalpiaz, F. (2018). Refinement of user stories into backlog items: Linguistic structure and action verbs. *Accepted paper, University of Utrecht*.

Niu, N., Brinkkemper, S., Franch, X., Partanen, J., & Savolainen, J. (2018). Requirements engineering and continuous deployment. *IEEE software*, *35*(2), 86–90.

North, D. (2006). Behavior modification. *Better Software Magazine*.

Nurmuliani, N., Zowghi, D., & Powell, S. (2004). Analysis of requirements volatility during software development life cycle. *In: Software Engineering Conference, 2004. Proceedings. 2004 Australian*, IEEE, 28–37.

Nuseibeh, B. (2001). Weaving together requirements and architectures. *Computer*, *34*(3), 115–119.

Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering: a roadmap. *In: Proceedings of the Conference on the Future of Software Engineering*, ACM, 35–46.

Okoli, C., & Schabram, K. (2010). A guide to conducting a systematic literature review of information systems research. *Sprouts: Working Papers on Information Systems*, *10*(26).

OMG Group. (2002). Meta object facility (MOF) specification (version 1.4). Retrieved from `https://www.omg.org/spec/MOF/1.4` (Accessed: 18-03-2019)

Patton, J. (2005). Finding the forest in the trees. *In: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 266–274.

Pohl, K., Böckle, G., & van der Linden, F. (2005). *Software product line engineering: foundations, principles and techniques*. Springer Science Business Media.

Rodriíguez, P., Mendes, E., & Turhan, B. (2018). Key stakeholders' value propositions for feature selection in software-intensive products: An industrial case study.

Rozanski, N., & Woods, E. (2011). *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley.

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, *14*(2), 131–164.

Schach, S., & Tomer, A. (2000). A maintenance-oriented approach to software construction. *Journal of Software Maintenance: Research and Practice*, *12*(1), 25–45.

Seffah, A., Donyaee, M., Kline, R., & Padda, H. (2006). Usability measurement and metrics: A consolidated model. *Software quality journal*, *14*(2), 159–178.

Shaw, M., & Gaines, B. (1989). Comparing conceptual structures: consensus, conflict, correspondence and contrast. *Knowledge acquisition*, *1*(4), 341–363.

Shekaran, C., Garlan, D., Jackson, M., Mead, N., Potts, C., & Reubenstein, H. (1994). The role of software architecture in requirements engineering. *In Requirements Engineering, 1994., Proceedings of the First International Conference on*, 239–245.

Ulwick, A. (2003). The strategic role of customer requirements in innovation. *Strategyn inc*, *13*(12), 1–24.

Ulwick, A. (2018). Alan klement's war on jobs-to-be-done. Retrieved from `https://jobs-to-be-done.com/alan-klements-war-on-jobs-to-be-done-dad8eaed567c` (Accessed: 12-12-2018)

Ulwick, A., & Hamilton, P. (2016). The jobs-to-be-done growth strategy matrix. *Technical report, Strategyn*, 1–12.

Wautelet, Y., Heng, S., Kolp, M., Mirbel, I., & Poelmans, S. (2016). Building a rationale diagram for evaluating user story sets. *In: Research Challenges in Information Science (RCIS), 2016 IEEE Tenth International Conference on*, 1–12.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.

Yin, R. (2017). *Case study research and applications: Design and methods*. Sage publication.

Yin, R. K. (1981). The case study as a serious research strategy. *Knowledge*, *3*(1), 97–114.

Zave, P. (1997). Classification of research efforts in requirements engineering. *ACM Computing Surveys*, *29*(4), 315–321.

Zave, P. (2003). An experiment in feature engineering. *In: Programming methodology*, 353–377.

Zowghi, D., & Nurmuliani, N. (2002). A study of the impact of requirements volatility on software project performance. *In: Ninth Asia-Pacific Software Engineering Conference*, 3–11.

Appendix A

This appendix contains the artifacts of the first case that were gathered. In this case these include a FAM and a feature diagram (Blessinga, 2018).

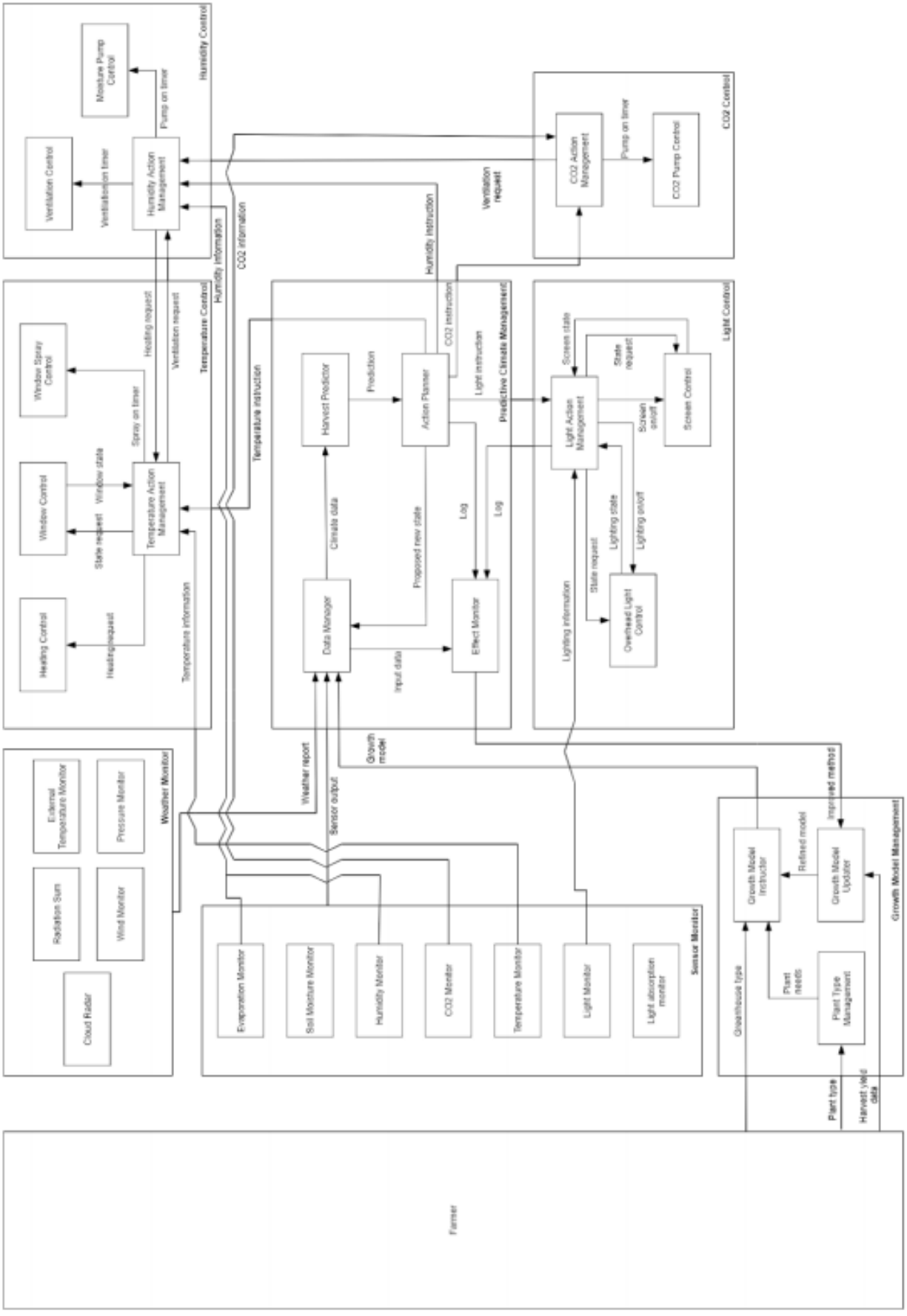

Figure 37: FAM of greenhouse system of case 1.

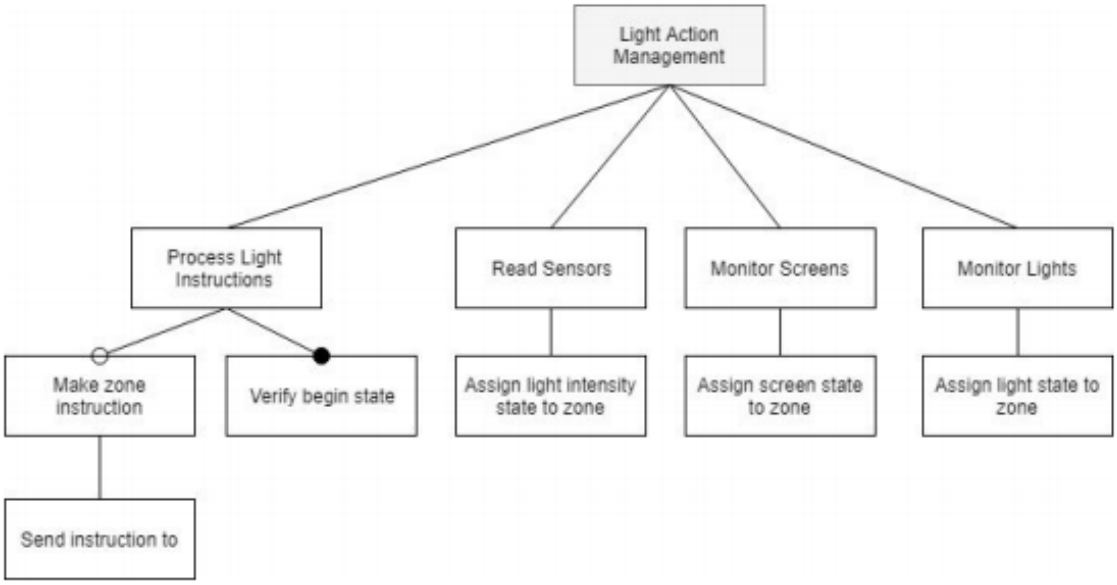

Figure 38: Feature diagram of the light action management module in case 1.

Appendix B

This appendix contains the artifacts of the second case that were recovered. In this case these include a FAM and multiple feature diagrams.

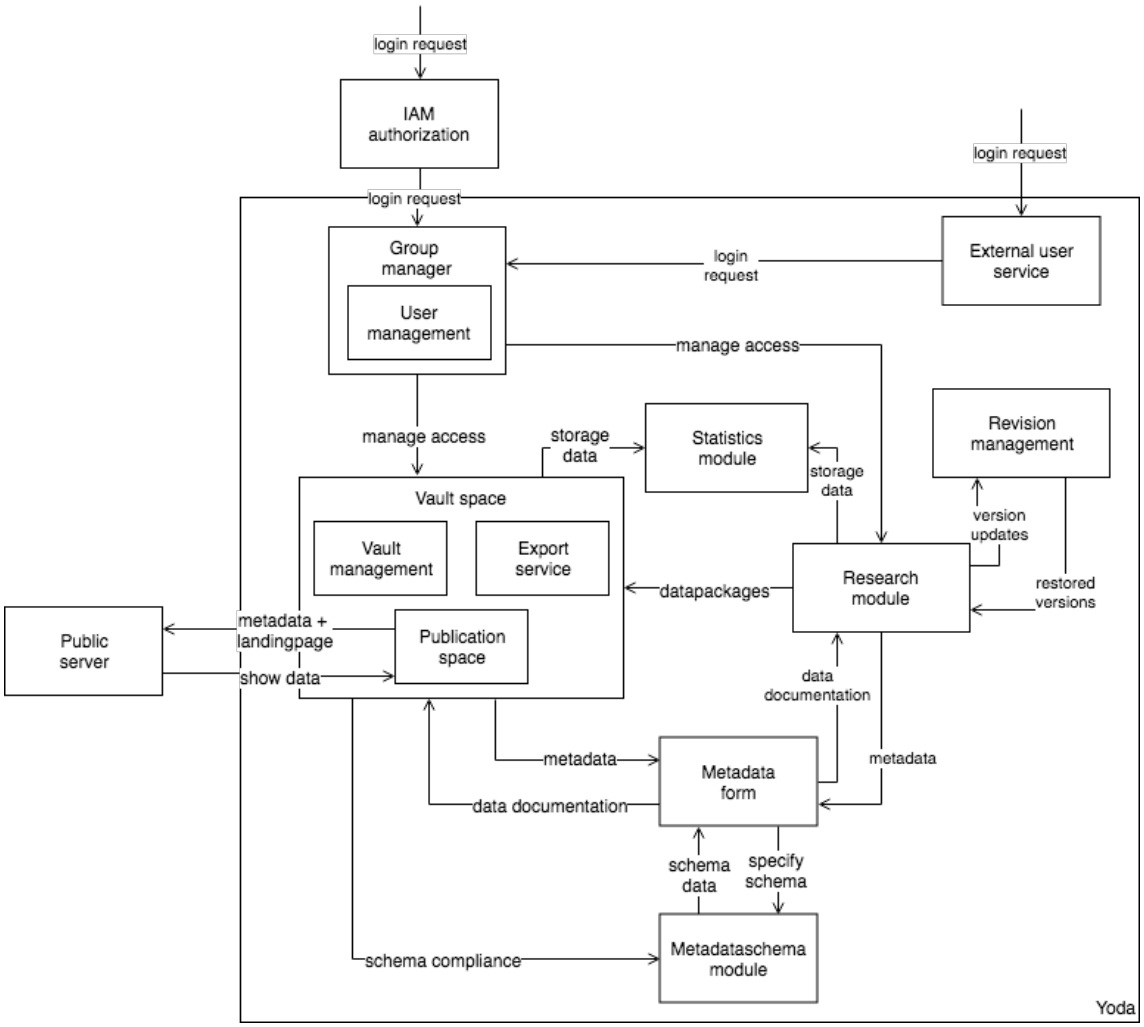

Figure 39: FAM of the Yoda system of case 2 (front end-only).

As was mentioned previously, the FAM was modeled in collaboration with a Yoda developer and also validated by them. Theoretically, including only one sub module in a module makes little sense, but since the implementation of the system was used to model the functional architecture, it still has been modeled as such here.

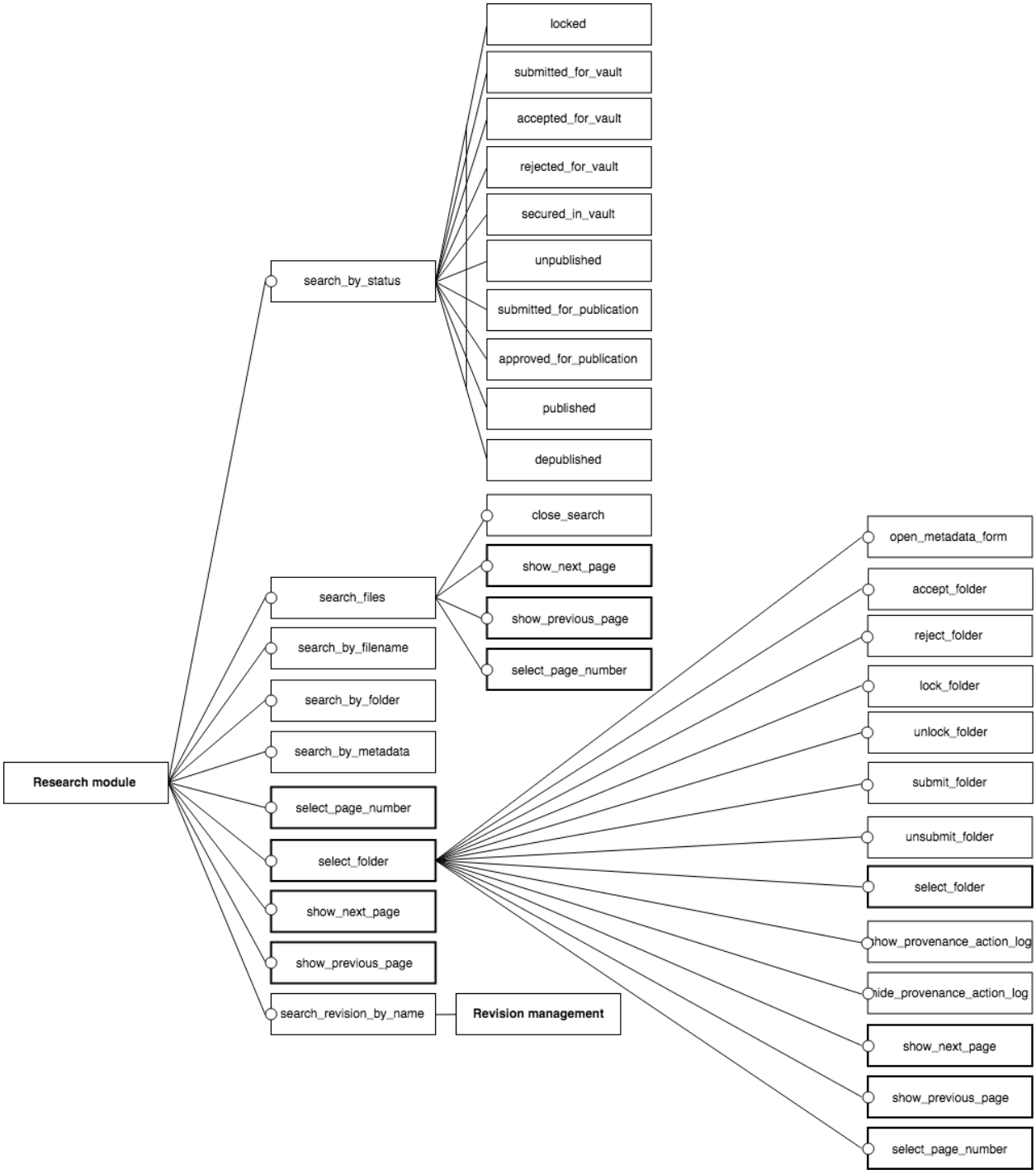

Figure 40: Feature diagram of the Research module (YDA-09).

Figure 40 shows all the features contained in the Research module. Duplicate features (within the module, not between modules) are indicated by the thick outlines of the boxes. Since the Revision management module can be accessed through the Research module it is shown here, but its full feature diagram is presented in figure 43.

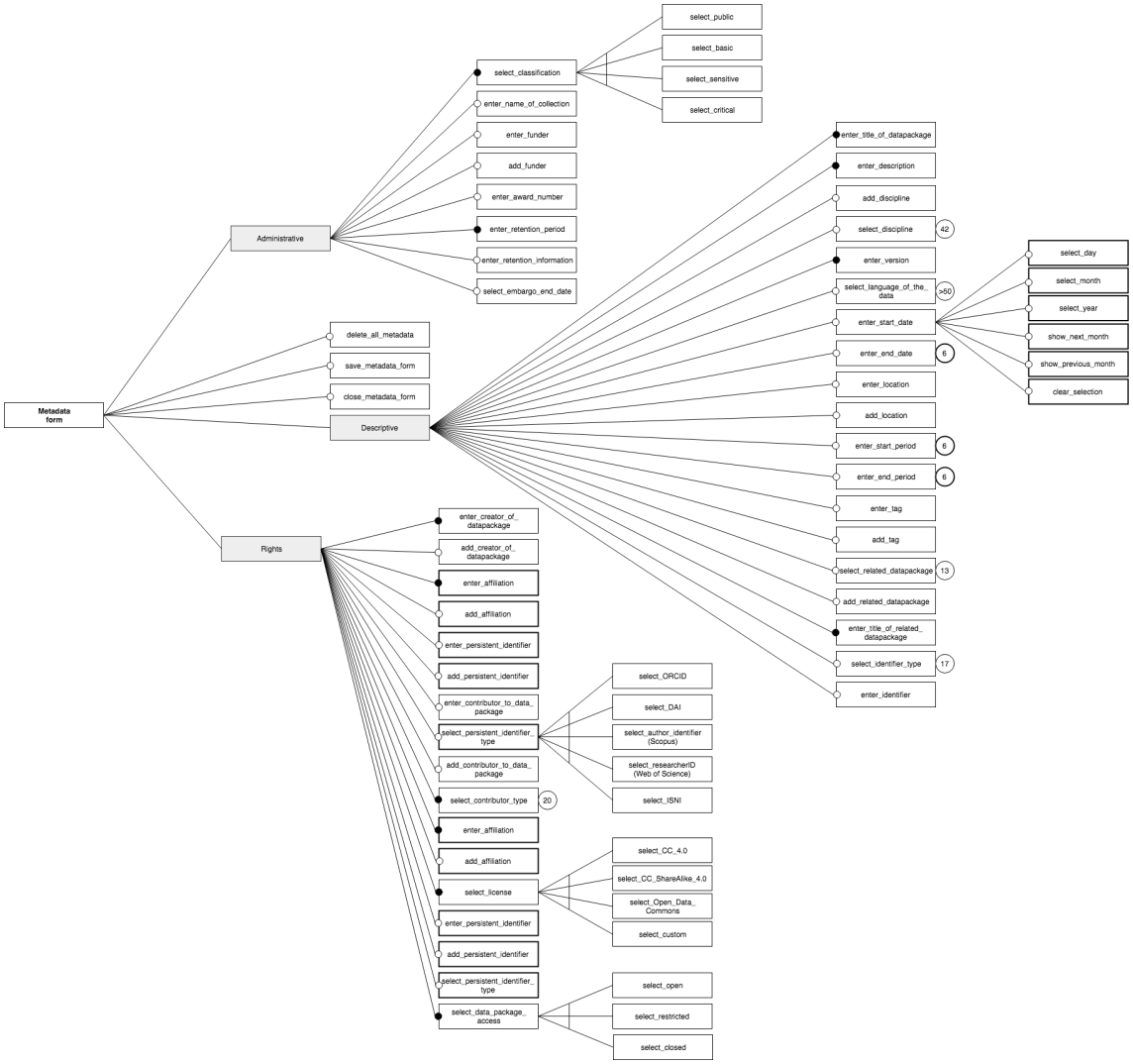

Figure 41: Feature diagram of the Metadata form module (YDA-94).

In order to improve legibility in figure 41, the GUI headers are included in gray boxes. Again, duplicate features (within the diagram) are indicated by the thicker outlines for the boxes. The numbers in circles refer to the number of options the feature presents after clicking on it. They have been omitted here, since they would only clutter the diagram and are not specified in individual USs. The only exceptions are the circles with a thick outline and a six in them, because these refer to duplicate features that are presented by the atomic features contained in the 'enter start date' composite feature.

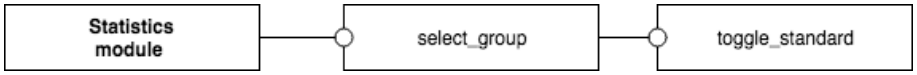

Figure 42: Feature diagram of the Statistics module (YDA-156).

The Statistics module, shown in figure 42, only includes two features as of yet.

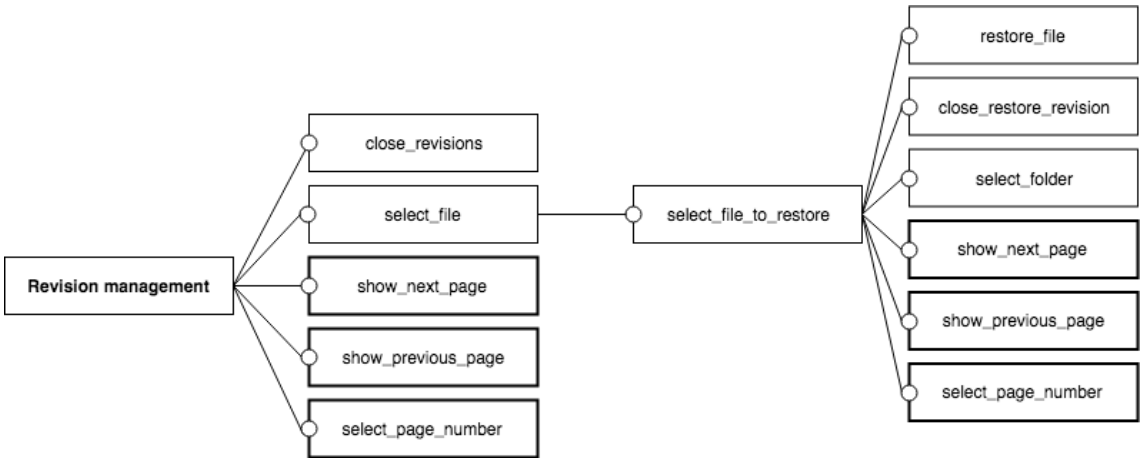

Figure 43: Feature diagram of the Revision management module (YDA-95).

Figure 43 visualizes the features contained in the Revision management module. Like before, duplicate features within the diagram are presented as a box with a thick outline.

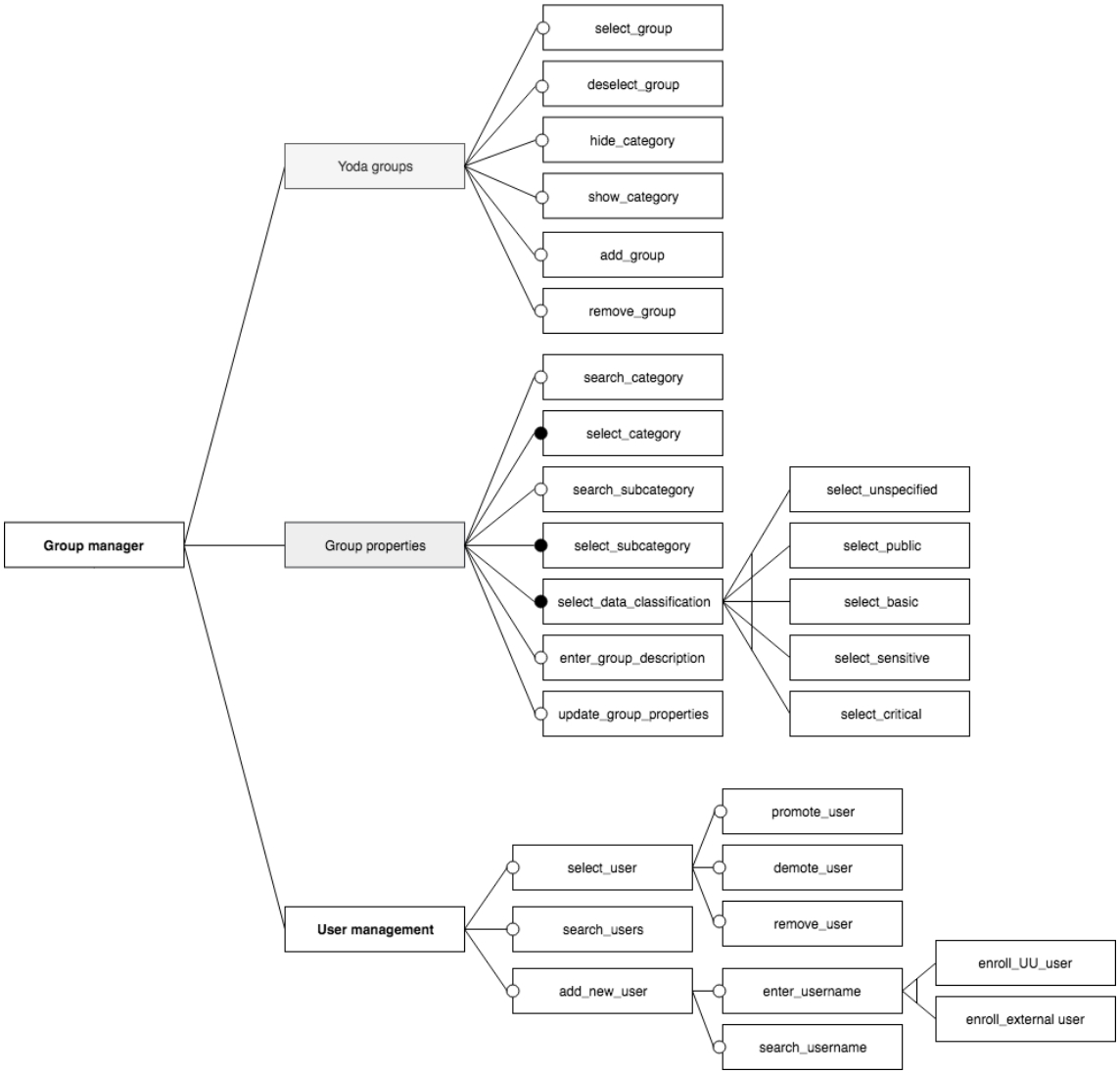

Figure 44: Feature diagram of the Group management module (YDA-02), which also contains the User management sub module (YDA-157).

As was the case in figure 41, the gray boxes in figure 44 are used to improve legibility and refer to headers in the GUI. Since the Group manager module also contains the User management sub module, the feature diagram of the latter is included too.

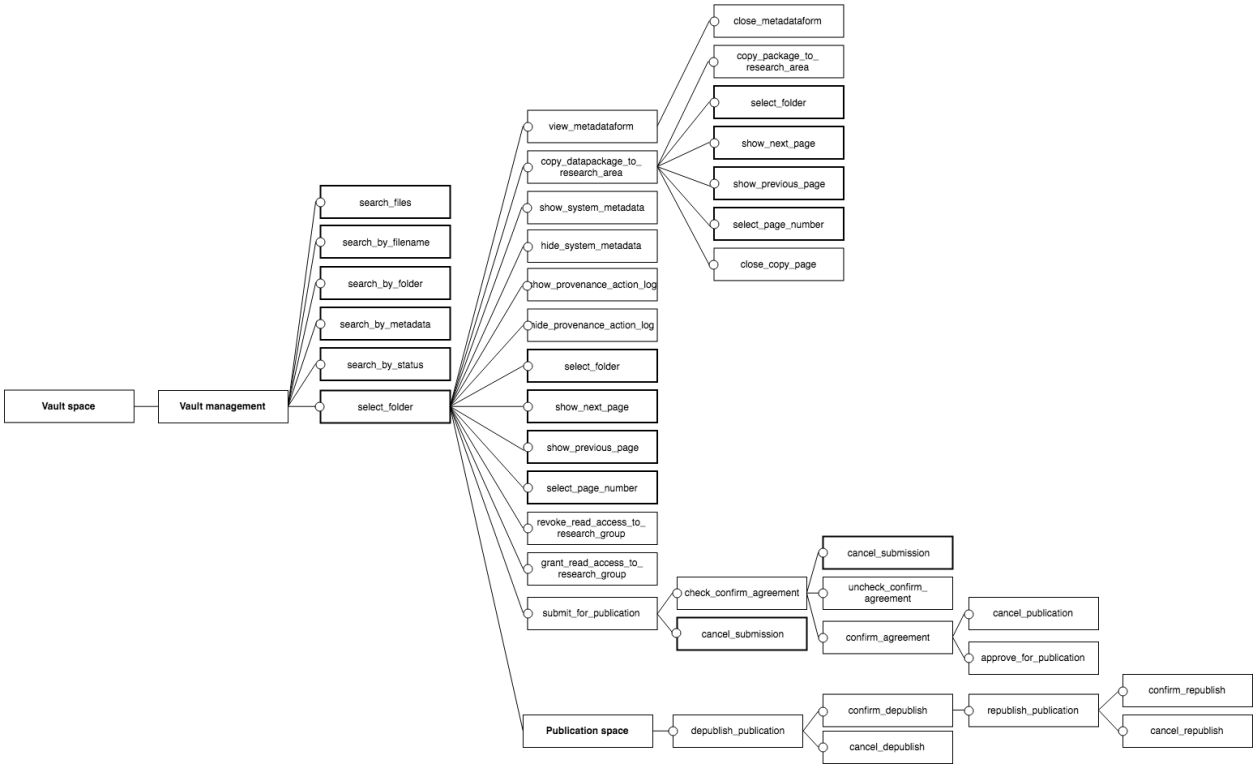

Figure 45: Feature diagram of the Vault space module (YDA-96), which also contains the Vault management (YDA-321) and Publication space (YDA-800) sub modules.

The Vault space module itself does not contain any front-end features itself, but instead contains the Vault management and Publication space modules, as visualized in figure 45. Taking into account the structure in the diagram, it may seem that the Publication space sub module is part of the Vault management module while this is not the case, the former is merely accessed through the latter. Again, duplicate features within the diagram are indicated by boxes with thick outlines.

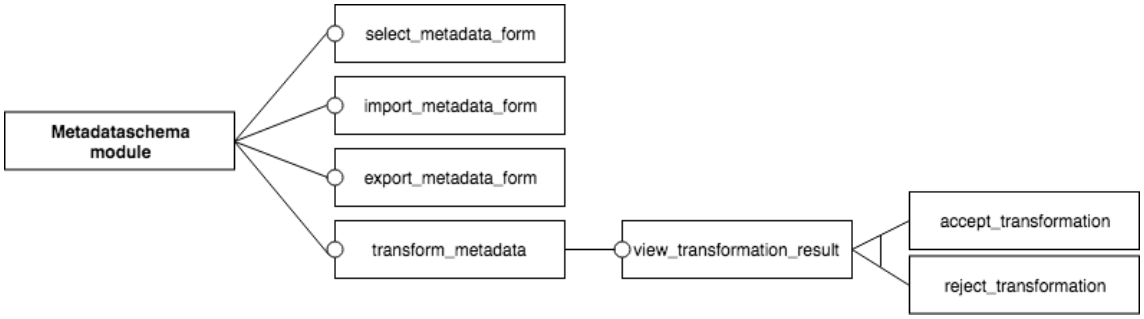

Figure 46: Feature diagram of the Metadataschema module (YDA-2208).

Figure 46 shows the feature diagram for the Metadataschema module, without duplicate features and sub modules.

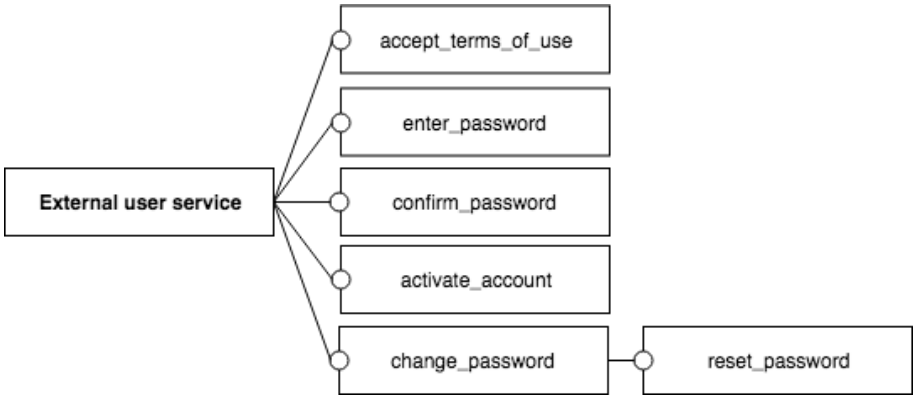

Figure 47: Feature diagram of the External user service module (YDA-2110).

The feature diagram of the External user service module is presented in figure 47, also without any duplicate features and sub modules.

Appendix C

Due to the inclusion of sensitive materials, the FAM and feature diagrams of case 3 may not be disclosed.

Appendix D

The paper that was submitted to RE@Next! is included in this appendix in full.

On the Nature of Links between Requirements and Architectures: Case Studies on User Story Utilization in Agile Development

Anonymous

***Abstract*—Communication between requirements engineers and software architects is experienced as problematic. In this paper we present the Requirements Engineering for Software Architecture (RE4SA) model as a tool that supports the communication between these two roles. In the RE4SA model, requirements are expressed as epic stories and user stories, which are linked to modules and features, respectively, as their architectural counterparts. By applying the RE4SA model to a multi-case study, we investigate the nature of the relationships between the requirements and the architectural artifacts. Based on the gained experience, we put forward nine hypotheses for further research on the utilization of user stories in agile RE.**

I. Introduction

Communication flaws within a development team are considered one of the most important issues in requirements engineering (RE) and are sometimes identified as the main cause for project failure, according to the NaPiRE project [1]. In a 2014 study, Smith and colleagues found that 47% of unsuccessful projects failed due to poor requirements management [2]. Similarly, volatile requirements have been called one of the main issues in the software industry in recent history [3], [4]. The 'evil circle' principle [5] states that problems in the RE process do not remain isolated, but rather echo throughout a development process. Other major contributors to project failure include scope creep due to inadequate requirements, poor communications within the project team or among the stakeholders, and key internal stakeholders leaving the project [6]. Finally, proper architecture documentation can help prevent architectural drift and erosion, reduce costs and improve software quality [7].

Nuseibeh recognized that requirements specification and design cannot be separated due to their inter-dependencies [8]. The Twin Peaks model describes how requirements and architecture are defined concurrently, yet being separate specifications, with the former guiding the latter and the latter constraining the former. The Reciprocal Twin Peaks extends this work for agile development and explains why the synergy between requirements and artifacts matters. In short, a development process has to manage a continuous flow of requirements, as well as a continuously changing architecture [9]. Consistency between the two helps prevent misunderstandings in the development team. However, realizing this consistency should not burden the involved stakeholders with excessive work as the improvement in communication is meant to prevent incorrect implementations, rework and wasting time, money and potential other resources. Therefore, tools are needed to achieve consistency between the artifacts. Software architecture demands good requirements engineering, which can only be guaranteed if proper communication exists.

While Nuseibeh and Lucassen identified challenges and explained how RE and SA can support each other, they did not provide any specific approaches for tackling these challenges. As a remedy, we present explicit concepts and relationships that can be utilized to link requirements and architectures.

The remainder of this paper is structured as follows. In Section II, we present the RE4SA model, alongside its theoretical background, rationale and objectives. Subsequently, we discuss the feasibility and applicability in Section III by means of a multi-case study. This section also explains how the model can be applied, our main findings and the observed benefits. The empirical work serves as the basis for us to draw nine hypotheses that will guide future work (Section IV). Finally, Section V summarizes our contribution and discusses the main challenges we have identified.

II. The RE4SA Model

In an attempt to facilitate good communication within the development team, we propose the Requirements Engineering for Software Architecture (RE4SA) model, visualized in Fig. 1. RE4SA was assembled on the basis of tight collaboration with industrial partners in the software products domain, and it combines artifacts (like user stories and features) that we found often employed in their work practices.

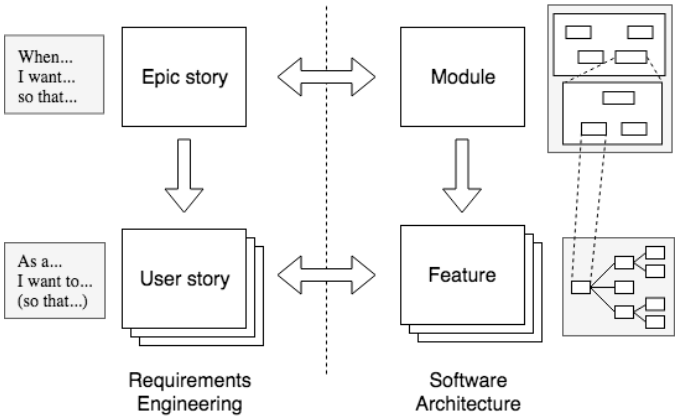

Fig. 1. The Requirements Engineering for Software Architecture model.

A. Concept

Similar to the Twin Peaks model, the RE4SA model links the RE process of a software product to its Software Architecture (SA). More specifically, it describes the links between Epic Stories (ESs) [10] and User Stories (USs) [11] in the requirements and modules and features in the functional architecture [12], respectively. Essentially, the problem space, which describes the requirements and their intended behavior, is related to the solution space that defines how intended behavior is implemented in a system and thus how requirements are satisfied [13]. ESs can be used to describe the modules in the architecture, while USs introduce more detail by describing the features of a software product. We define a functional architecture as a description of "*the system by its functional behavior and the interactions observable with the environment*" [12]. Examples of functional architectures are illustrated in the top two levels of Fig. 4 and Fig. 5.

B. Illustration

As an example, consider a navigation app. Some of its implemented features (white boxes) are visualized in Fig. 2. An ES, consisting of the three parts problematic situation,

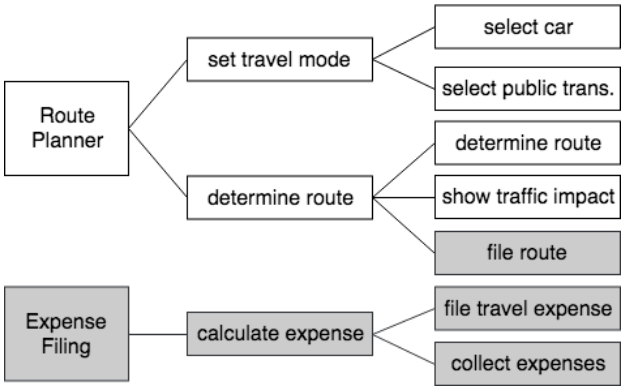

Fig. 2. Example of a feature diagram extension.

motivation and expected outcome, can be written to extend the functionality of the app: "*When I have to file for expenses to my employer, I want to have all my routes and local expenses registered, so that I can collect my expense data and minimize effort for filing.*" Using this ES it is possible to include the "Expense Filing" module in the software architecture. Going into the requirements process, the aforementioned ES can be refined by the following two USs. "*As a consultant, I want to file my travel expenses, so that I have complete expense and travel data with minimal effort*", using the verbs and nouns of the US this results in the "file travel expense" feature, which is part of the "Expense Filing" module. Subsequently, "*As a consultant, I want to collect my expenses of the past month, so that I can get an overview of my monthly costs*" can also be added. In addition, the existing functionality of the app needs to be extended in order to support these new features. In its current form, the app does not keep track of the user's previous routes, which is necessary in order to file the expenses. The following US can be added to remedy this: "*As a consultant, I want to file a route, so that I can calculate the costs of that route for my employer*". The three newly added features described by the previous ES and USs are illustrated as gray boxes in Fig. 2. Features are not only grouped based on their functionality, their naming can also be utilized to determine their position. For instance, the "file route" feature is part of the "determine route" composite feature within the "Route Planner" module. The keyword here is 'route'. On the other hand, the architectural elements related to the newly introduced ES all contain the word 'expense' or a variation thereof. Figure 2 illustrates the way ESs and USs can be transformed into functional architecture components and also serves as a simple means to discuss the position of the new functionality as well as the impact on the current implementation of the system.

C. Expected Benefits

RE4SA is intended to improve communication between product managers and software architects or product owners through (1) simple communication means, (2) clear structural guidelines, and (3) consistent domain terminology. The objective of the RE4SA model, however, is not limited to improving communication. Gayer *et al.* argued for the need of dynamic architecture creation. This architecture allows for traceability in order to make software more maintainable, changeable and sustainable [14]. By establishing a relationship between ESs and modules and USs and features respectively, traceability is supported, with little documentation and effort required. The conflicts between architects and requirements engineers has been the subject of research before, such as in the case of the RADAR tool [15]. RADAR supports requirements and architecture decision analysis in an attempt to reduce struggle and miscommunication among stakeholders. As opposed to designing a new modeling language and analysis approach, we apply knowledge and techniques that already have a high adoption in the RE4SA model, in order to minimize the need for change and training in industry. USs, for instance, were found to often be one of the requirements documents used in agile methods [16]. The validity and applicability of the RE4SA model are discussed based on three case studies, presented in Section III.

III. Case Studies: First Experiences with RE4SA

To support the envisioned relevance and benefits argued in the previous sections, we present an industrial multi-case study that illustrates how RE4SA can be applied and that helps us develop the model and formulate hypotheses for future research. The three cases encompass different apps; while the selection is based on industrial availability, all of them target business consumers, and each studies a different use case of RE4SA: (i) modeling requirements and architectures prior to developing software, (ii) extending an existing product, and (iii) recovering an architecture. Case study findings on RE4SA are numbered and presented in bold.

A. Modeling for a Start-up

The first case is concerned with a software start-up in the context of intelligent greenhouses. To support modeling activities for the start-up's software, the RE4SA model was applied, which resulted in the formulation of 31 ESs and 96 USs. Based on the formulated RE artifacts, a functional architecture was developed that consisted of 31 modules. During this activity, the start-up's founder observed that **Finding-A.1:** ES formulation leads to module identification. Prior to development, the artifacts were discussed with the system's stakeholder to determine their value. Most importantly, it became apparent that none of the artifacts alone provided sufficient information to about the system to be developed. However, when taken together, the artifacts sufficiently provide a comprehensive overview, which provides evidence on the synergies between RE and SA, leading to the finding **F-A.2:** RE and SA artifacts shall be used in conjunction in a synergistic fashion.

Furthermore, this case has produced additional insight into how RE4SA should be applied to a development process. Firstly, it is important to determine the level of abstraction in terms of ESs and USs. Information to fully define the abstraction level is lacking prior to development, so instead it is important to distinguish between the ES and US level. Moreover, an ES should categorize at least two USs, otherwise the formulation of the ES is superfluous. This implies that if an ES contains only one US, the ES is formulated on an incorrect level of abstraction and should be reformulated to fit the US template instead. This may seem trivial, but is crucial to the structure of RE4SA given the levels of abstraction. Once this distinction is established, ESs can be written, followed by USs. **F-A.3:** the level of abstraction should be established prior to developing the RE artifacts.

The ESs were found to be especially useful in the modeling and subsequent naming of modules in the functional architecture. Names for modules could often be derived from the nouns and verbs included in the ESs, which does not only simplify the modeling process, but also facilitates linking RE artifacts to the functional architecture, **F-A.4:** modules names can be derived from ESs. Likewise, the verbs and nouns used to formulate USs can be adopted to name features (as illustrated in Section II-B), **F-A.5:** feature names ought to be derived from USs. The formulation of ESs is often functional in nature, which lends itself to mapping them to a functional architecture, while still allowing the requirements engineers to work from a problem-oriented perspective. In addition to designing modules, ESs can support the specification of information flows between said modules too, as shown in Table I.

The (problematic) situation, motivation and expected outcome formulated in an ES can be utilized to determine the input flow, module name and output flow respectively. **F-A.6:** ESs provide naming suggestions for all elements in a functional architecture: modules, input flows, and output flows. An ES can be translated into functional architectural elements as illustrated in Fig. 3.

TABLE I
INFORMATION FLOWS AND MODULES FORMULATED BASED ON ESS.

Epic Story	Module	
	Input flow	**Output flow**
When using a neural network, I want to gather and format data continuously, so that the AI can interpret the data.	Data Manager	
	sensor output, weather report, growth model	climate data
When there is new prediction data available, I want to run it through a trained neural network, so that I can make yield predictions.	Harvest Predictor	
	climate data	prediction
When I have a yield prediction, I want to plan the right course of action, so that I can set the right climate conditions.	Action Planner	
	prediction	instructions: humidity, light, CO2
When receiving a humidity instruction, I want to determine a course of action, so that I can control humidity systems.	Humidity Action Management	
	humidity instruction	ventilation on timer, pump on timer

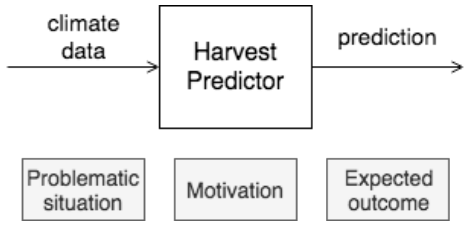

Fig. 3. ES to module translation in a functional architecture.

B. Extending a Software Product

The RE4SA model was also applied to a software company that determines the value of real estate and wanted to extend their software product with valuation analysis. The architectural artifacts were limited to tacit knowledge repositories, so it was unclear which existing modules and features were relevant for creating the software extension. Therefore, it was necessary to recover the functional architecture of the current system manually. The existing modules (28) have been modeled as feature diagrams, with 121 atomic features in total.

Extending the functional architecture with new modules resulted in several findings. First, the clarity of the visual representation was found to ease the communication between product manager and technical lead through the use of explicit architectural components. Second, the diagrams highlight which modules the extension depends on to implement new features. The development team confirmed that the functional architecture was helpful in discussions among stakeholders, leading to the finding **F-B.1:** the concepts in the RE4SA model are suitable for functional architecture recovery.

Then, the RE4SA model was applied to establish a functional architecture that satisfied the requirements of the software product extension. These requirements were elicited by the product manager, directly from customers of the software product. 23 USs were created and subsequently categorized

in eight ESs. The functional architecture of the current system, illustrated in Fig. 4 positions the extension on three different levels of abstraction. These models identify parts

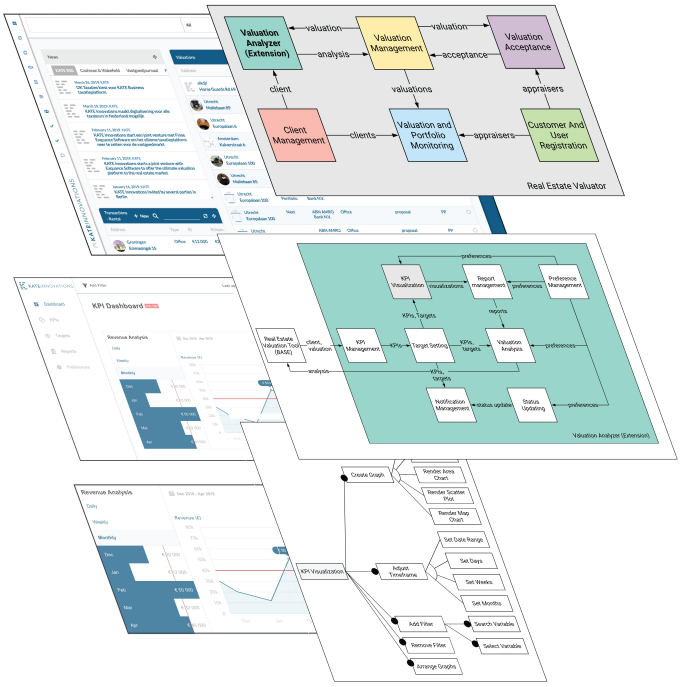

Fig. 4. Three-layered functional architecture of a real estate valuation tool.

the implemented system will be affected by the extension. Furthermore, the required interactions (modeled as information flows) with other modules are captured in the top level. The functional architecture enables a clear focus for sprint planning on developing well-defined components of the system and stakeholder validation of the architecture. **F-B.2:** the functional architecture allows sprint planning to focus on specific components. Moreover, thanks to the separation of concerns that RE4SA promotes, **F-B.3:** units of the software can be tested individually and thereby promotes re-usability. Furthermore, the functional architecture could be used to determine which parts of the system are (not) affected by the software product extension, **F-B.4:** the functional architecture has an appropriate level of abstraction that enables predicting the impact of new requirements on the existing system. Based on this evaluation of the RE4SA model in practice, there appears to be **F-B.5:** a 1-to-1 mapping between ESs and modules, and between USs and features.

C. Architecture Recovery of a Web Application

In a recent study, Tamburri and Kazman affirmed that "*both theory and practice suggests that maintaining good quality software architectures is non-trivial*" [17]. For web applications the problem may be even worse: proper documentation is rare, because well-known software engineering practices are seldom adopted by web developers and there is a high employee turnover rate [18]. To improve the understanding of these applications or systems, reverse engineering and system visualization techniques have been proposed [18].

The RE4SA model was applied to recover a functional architecture from the Graphical User Interface (GUI) of a web application. The application, a tournament planner with nearly 25,000 lines of code, was modeled in a feature diagram using the GUI as input for architecture recovery. Then, the website hierarchy and the principles of the RE4SA model were used to group the 199 atomic features into eight modules. Each module embodies a manageable and well-defined functionality that can be developed relatively independently from other modules [19]. Sub modules (21) were added for six modules to further differentiate between features and to facilitate the interpretation of the model by different stakeholders. The created interactive visualization of the model, shown in Fig. 5, was found by an interviewee at the company to help improve the communication between the stakeholders by allowing them to discuss specific components of the architecture, instead of a list with discussion points. Moreover, there is no need for all the stakeholders to understand the code. Therefore, we could conclude that **F-C.1:** the layered architecture recovered using the RE4SA model facilitates communication between stakeholders.

Fig. 5. Three-layered functional architecture of a web application, recovered based on the GUI.

During this case study, a crowdsourcing platform was used for the elicitation of new requirements in the form of USs. One user requested: "*As an organizer I want to set a unique start time of a playing field for each match day*". The current "set start time of playing field" feature does not allow this. If this feature would be implemented, it would require two new sub features ("set start time per day" and "set overall

start time") with an alternative relationship to the feature above: exactly one of the sub features must be selected. **F-C.2:** the relationship between USs and features facilitates the positioning of new features in the functional architecture.

Mapping USs to features makes it easier to analyze which parts of the architecture are affected by evolving requirements. Furthermore, once the requirements have been mapped to the architecture, their location in the architecture and relation to the type of architectural component can support development estimations of USs. For instance, USs that are linked to atomic features were found to be relatively easy to implement by all stakeholders, while USs that require a new sub module are more complex and require more time to develop and implement. **F-C.3:** applying the RE4SA model facilitates impact forecasting in the context of changing requirements. Although no automation for creating and maintaining the traceability between the requirements, architecture and code was used, the case study shows how the RE4SA model can serve as a basis for communication and for the further analysis on the linguistic relationship between USs and features.

IV. Hypotheses

We have proposed a model that relates RE to SA in order to improve communication between stakeholders and support software development. The RE4SA model was proven to be promising based on a multi-case study, however, coincidence is still a factor and the relationships have not yet been sufficiently investigated. For instance, the strength of the relationships and their cardinalities are still unclear. Furthermore, the multi-case study that was presented previously was conducted on a small scale over a short period of time. To be able to accurately refine the links, large scale and long term research are required. In case of the latter, this could involve a study that examines the design and maintenance phase of a software development project, as opposed to one or the other. Moreover, we envision additional purposes for the model as well.

Based on the insight gained during the multi-case study, we have formulated nine hypotheses for future research in categories structure, requirements, architecture, and development process, presented in Table II. While the existence of relationships between the artifacts was confirmed, the exact nature of these relationships require further refinement as well as their cardinalities. The hypothesized cardinalities on the artifact structuring in the RE4SA model are shown in a meta-model in Fig. 6.

We hypothesize that ESs and modules should contain two or more USs and features respectively. If this rule is violated it is likely that an incorrect level of abstraction is used. Secondly, we expect that a US and a feature both belong to one ES or module. Based on the quality framework designed for USs, it is reasonable to assume that a US describes one feature, since a US of sufficient quality should express a requirement for exactly one feature [20]. Similarly, we expect an ES to describe only one module. On the other hand, we hypothesize that features and modules are described by one RE artifact, as we expect that the abstraction level may be inaccurate if multiple RE artifacts are required to describe one module or feature. The hypothesized cardinalities (*H.1-H.3*) will be tested by conducting empirical research. Multiple case studies will analyze real-world artifacts in order to precisely define the links between the concepts.

A natural progression of this research is to analyze how change impact can be supported, especially in the context of impact forecasting and automated requirements traceability. *H.4* focuses on the maintenance and evolution phase of a software product and will therefore require a fully implemented system with existing development artifacts as a case study. Likewise, we aim to investigate how to generate (partial) artifacts to facilitate software development. We hypothesize (*H.5*) that both these objectives can be achieved by developing a complete picture of the artifact structures included in the model, as well as by conducting research on linguistic analyses of the artifacts and the potential discovery of linguistic patterns and links. These structures will be investigated using linguistic analysis techniques such as Part-of-Speech tagging and NLP.

TABLE II
Hypotheses for future research.

ID	**Hypothesis**	**Findings**
	Structure	
H.1	ESs and modules contain at least two USs or features.	A.3, C.1
H.2	There is a 1..1 relationship between ESs and modules and USs and features.	A.1, A.2, A.6, B.5
H.3	USs and features belong to exactly one ES or module respectively.	A.3, C.1
	Requirements	
H.4	The application of the RE4SA model effectively supports impact analysis of new requirements.	B.4, C.3
H.5	There exists a linguistic relationship between names of RE and SA artifacts.	A.2, A.4, A.5
	Architecture	
H.6	The RE4SA model supports architecture recovery activities.	B.1
H.7	The application of the RE4SA model effectively supports positioning of new features.	C.2
	Development process	
H.8	The RE4SA model can be utilized to guide and support testing activities.	B.3
H.9	The RE4SA model uses appropriate levels of abstraction so that it can be embedded in software product management activities, such as release planning.	B.2

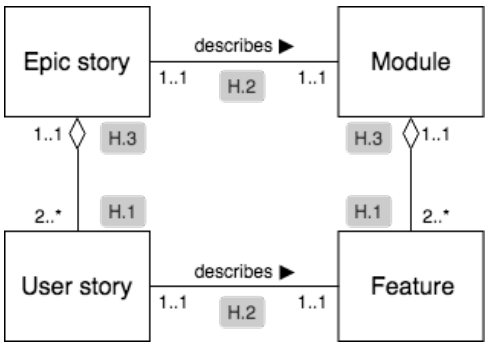

Fig. 6. Meta-model of the relationships between the RE4SA concepts, illustrating the hypotheses.

Regarding architectures, functional architecture recovery was performed twice during the multi-case study presented earlier. However, this recovery was not the objective of the study and was also not performed in a structured manner. In order to properly test *H.6*, architecture recovery activities need to be performed (following a rigorous method) on a real-life case study and subsequently replicated using other cases. The RE4SA model can also be applied to design a method for software product design and development. This could be effective for keeping the software architecture up to date, and deciding where to position new features in the software. *H.7* will be tested by examining existing systems that need to be extended or updated through means of a case study. It is expected that during all of these studies, artifacts will need to be developed. Finally, case B and C have hinted at the RE4SA model's usefulness for release planning and guiding the testing process, stating that parts of the software can be tested somewhat independently and can provide separate functionality. *H.8* requires case studies to assess the applicability and feasibility of utilizing the RE4SA model for testing activities, as well as expert interviews or surveys to evaluate the usability and reliability. Finally, *H.9* will be tested based on additional literature research to determine how the model can be utilized and whether it uses the appropriate levels of abstraction, as well as case studies to validate these applications.

V. Conclusion

In this study on the links between requirements and architectures we propose a model with the objective to solve communication issues as well as supporting the software development process, best illustrated by the RE4SA model. A multi-case study was performed to verify the accuracy and applicability of the model in various contexts. The cases have also shown that the use of the model, and its underlying principles, supports multiple activities, such as: determining the level of abstraction for modeling the system, name derivations, identifying information flows, recovery of functional architectures, modeling extensions of an existing system, traceability between artifacts and impact forecasting. The most important results are the hypotheses for future research we formulated through the use of the multi-case study.

A few challenges related to the RE4SA model need to be addressed. Firstly, we need to study broader usage in large projects, since it was only applied in smaller cases up to this point. One of the main contributors to the expected hesitance of practitioners is the availability of a software architecture, more specifically, a functional architecture. On the other hand, the principle does rely on existing concepts that already have a wide adoption. As of yet we are not familiar with how the model can or should be applied in different development contexts. In similar fashion, there are no guidelines on how to apply the model. By this we mean that the starting point can vary and that the order of subsequent activities should not be fixed, since the development team should decide on the appropriate sequence of development activities. Finally, the software development process is not finished after eliciting requirements and designing the architecture. Future work should also focus on whether the RE4SA model can and should be extended in order to support more software product management activities, such as the design of technical architectures, feature programming, or release planning.

References

[1] D. Méndez Fernández *et al.*, "Naming the pain in requirements engineering," *Empirical software engineering*, vol. 22, no. 5, pp. 2298–2338, 2017.

[2] A. Smith, D. Bieg, and T. Cabrey, "PMI's pulse of the profession® in-depth report: Requirements management–a core competency for project and program success," *Project Management Institute, Newtown Square, PA*, 2014.

[3] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Communications of the ACM*, vol. 31, no. 11, pp. 1268–1288, 1988.

[4] D. Zowghi and N. Nurmuliani, "A study of the impact of requirements volatility on software project performance," in *Ninth Asia-Pacific Software Engineering Conference, 2002.* IEEE, 2002, pp. 3–11.

[5] T. Gilb and S. Finzi, *Principles of software engineering management*. Addison-wesley Reading, MA, 1988, vol. 11.

[6] D. L. Hughes, Y. K. Dwivedi, N. P. Rana, and A. C. Simintiras, "Information systems project failure–analysis of causal links using interpretive structural modelling," *Production Planning & Control*, vol. 27, no. 16, pp. 1313–1333, 2016.

[7] C. C. Venters, R. Capilla, S. Betz, B. Penzenstadler, T. Crick, S. Crouch, E. Y. Nakagawa, C. Becker, and C. Carrillo, "Software sustainability: Research and practice from a software architecture viewpoint," *Journal of Systems and Software*, vol. 138, pp. 174–188, 2018.

[8] B. Nuseibeh, "Weaving together requirements and architectures," *Computer*, vol. 34, no. 3, pp. 115–119, 2001.

[9] G. Lucassen, F. Dalpiaz, J. M. Van Der Werf, and S. Brinkkemper, "Bridging the twin peaks: the case of the software industry," in *Proceedings of the Fifth International Workshop on Twin Peaks of Requirements and Architecture*. IEEE Press, 2015, pp. 24–28.

[10] G. Lucassen, M. van de Keuken, F. Dalpiaz, S. Brinkkemper, G. W. Sloof, and J. Schlingmann, "Jobs-to-be-done oriented requirements engineering: a method for defining job stories," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2018, pp. 227–243.

[11] M. Cohn, *User Stories Applied: for Agile Software Development*. Redwood City, CA, USA: Addison Wesley Professional, 2004.

[12] S. Brinkkemper and S. Pachidi, "Functional architecture modeling for the software product industry," *In: European Conference on Software Architecture*, pp. 198–213, 2010.

[13] S. Apel and C. Kästner, "An overview of feature-oriented software development." *Journal of Object Technology*, vol. 8, no. 5, pp. 49–84, 2009.

[14] S. Gayer, A. Herrmann, T. Keuler, M. Riebisch, and P. O. Antonino, "Lightweight traceability for the agile architect," *Computer*, vol. 49, no. 5, pp. 64–71, 2016.

[15] S. A. Busari and E. Letier, "Radar: A lightweight tool for requirements and architecture decision analysis," in *Proceedings of the 39th International Conference on Software Engineering*. IEEE Press, 2017, pp. 552–562.

[16] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, "A systematic literature review on agile requirements engineering practices and challenges," *Computers in human behavior*, vol. 51, pp. 915–929, 2015.

[17] D. A. Tamburri and R. Kazman, "General methods for software architecture recovery: a potential approach and its evaluation," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1457–1489, 2018.

[18] A. E. Hassan and R. C. Holt, "Architecture recovery of web applications," in *Proceedings of the 24th International Conference on Software Engineering*. ACM, 2002, pp. 349–359.

[19] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[20] G. Lucassen, F. Dalpiaz, J. M. E. van der Werf, and S. Brinkkemper, "Improving agile requirements: the quality user story framework and tool," *Requirements Engineering*, vol. 21, no. 3, pp. 383–403, 2016.