From Idea to Product: Requirements Evolution within Software Projects

Master Thesis for the Master of Business Informatics at Utrecht University
by

Nikita van den Berg
(5825083)

First supervisor
Prof. Dr. Fabiano Dalpiaz
Second supervisor
Prof. Dr. Sjaak Brinkkemper
Daily supervisor
M.Sc. Tjerk Spijkman

May 2023

Abstract

Requirements evolution is a phenomenon that has been acknowledged and recognized, but not extensively researched yet. It has been widely known that requirements change is one of the most persistent challenges in the software industry. Having such a closely related field be seen as a persistent challenge, and the fact that researchers have been calling out for more research on the topic, emphasizes the necessity for research on requirements evolution. In this study, we delve deeper into requirements evolution through an exploratory case study, supported by a literature study.

In the literature study, we encountered various definitions and taxonomies on the topic, from which we developed a taxonomy of requirements evolution. In the case study, we closely followed the progress of an internal software project from its initiation until the creation of a minimum viable product. We collected all requirements and requirements-relevant information, tagged it, and create sequences of changes that represent the evolution of a requirement. Our analysis focused on identifying evolution patterns, combinations of associated tags, and the influence of timing, source location, and initiators on requirements evolution. Lastly, we investigated the similarities and differences of requirements evolution between the contents of conversations and those of project management systems.

We discovered some evidence that irreversible changes are less likely to occur, and we observed a correlation between stakeholders' roles and the requirements evolution steps they initiate. Furthermore, this correlation appeared to be linked to the company's working methodology, as revealed through stakeholder interviews conducted during the case study. Finally, we analyzed the recorded requirements conversations, and their connection to the documented requirements evolution from the project management system. We found that these conversations need to be analyzed on a different granularity level to be effectively utilized, and that there were differences in the occurrence of different tags between these discussions and the requirement management system.

Contents

Chapter 1

Introduction

> "*Evolution is a fact of life. Environments and the species that operate within them – living, artificial, or virtual – evolve. [1]*"

Evolution is everywhere, also in the software industry, or even more specifically in Requirements Engineering (RE). Although the presence of requirements evolution has been acknowledged [1], its nature and dynamics are not so evident [2]. The topic of evolution in RE is studied by a few researchers [2, 3, 4, 1, 5, 6], with others still calling for more attention to the subject [7]. Ambreen *et al.* [7] state that specifically empirical research on the topic is scarce. This research contributes by expanding the small pool of empirical research on the evolution of requirements.

A common technique in RE, especially for requirements elicitation, is the conduction of interviews and other types of conversations [8]. Although a few precursory studies exists in literature surrounding this technique [9, 10, 11, 12], only recently researchers have proposed Conversational RE as a systematic approach for conducting research around RE-related conversations [13]. This domain of Conversational RE is defined by Spijkman *et al.* [13] in "Back to the Roots: Linking User Stories to Requirements Elicitation Conversations" as:

> "*the analysis of requirements elicitation conversations (in short form, requirements conversations) aimed at identifying and extracting requirements-relevant information.*"

As this research is about the evolution of requirements, where the requirements are elicited through conversational techniques, this thesis takes place in the Conversational RE domain and contributes to this under-explored domain.

We explore this domain with a longitudinal case study where we extract data from a requirements specification document, a project management system, and requirements elicitation conversations. We extract all requirements and requirements-relevant information from those sources, then tag that information with the tags from a taxonomy. This taxonomy consolidates frameworks that were already existing in literature, with our own findings. After

tagging the collected data using our taxonomy, we analyze it on various perspectives. From analyzing patterns in requirements evolution, to analyzing the impact different initiators have on requirements evolution.

With this analysis of real-world data, we contribute to the limited pool of empirical research on the topic of requirements evolution.

1.1 The Gap

In this fairly new domain of Conversational RE, there is yet a lot to explore. The initial research in the field of conversational RE, by Spijkman *et al.* [14], has investigated the extraction of information from individual requirements conversations. However, additional information can be learned from analyzing subsequent conversations from the same project. Analyzing them is a time-consuming task which is also prone to error and forgetting things, as it is a human task [14]. Therefore, Spijkman *et al.* [14] worked on a prototype, named Trace2Conv, to support this important process of requirements elicitation. The tool is mostly tested with individual conversations and can be enhanced with information about subsequent conversations. Hence, a study focusing on subsequent requirements elicitation conversations not only holds potential for research advancement, particularly due to the lack of similar studies, but also offers opportunities for enhancing the usability of this tool in a business context. Thus, both academia and the business community can derive significant benefits from such a study.

Requirements evolution is a more explored domain than the Conversational RE domain, but even though some researchers talk about requirements evolution [2, 3, 4, 1, 5, 6], and others call for more attention to the subject [7], there is not one popular, often used definition. This brings the difficulty that there are no set boundaries as to where such evolution can start and where it ends. There has been research that talked about requirements evolution as a part of software evolution, thereby having a focus on the phases after a software artefact has been developed [15]. Others focus only on the early stages of RE for the requirements evolution [16]. Li *et al.* [2] also identified this problem, and therefore proposed the following definition based on the definition of the word evolution itself from the Webster Dictionary:

> "*Requirements evolution is a process of continuous change of requirements in a certain direction. It can happen during the entire software life cycle excluding the definition phase. The propagation of requirements evolution spans from requirements to maintenance of a software system.*"

The definition phase is the phase that ends when the requirements document as an artefact is completed [2]. However, this thus excludes the evolution that can take place in the initial elicitation process, or how Ferrari *et al.* [6] call it, the *early requirements* evolution.

As to what requirement evolution consists of, there is not a definitive answer either. One can say that a series of requirements change is requirements evolution, which was also

studied in the earlier mentioned SLR [2], and for this a clear definition was present. Li *et al.* [2], Mao *et al.* [17], Davis *et al.* [4] and Costello and Liu [18] state that requirements can be tagged as follows when there is a change:

- Addition
- Modification
- Deletion

However, Ferrari *et al.* [6] use different tags in their research about requirements evolution, where the tags are comparing the new set of requirements with an initial set of requirements. The tags used in that research are *Existing, Refinement* and *New*.

Anderson and Felici [19] use a more extensive tagging list which also includes the tags that define requirements change. However, in their research they had quite a specific data set with an avionics case study, therefore some tags such as *range modification* might not be as generalizable for other settings. The tags identified by Anderson and Felici [19] are: *Add, Delete and Modify requirements, Explanation, Rewording, Traceability, Non-compliance, Hardware modification, Range modification* and *Add, Delete and Rename parameters/variables*.

Only the work of Harker *et al.* [20], which is also referenced by Ernst *et al.* [21], has a more unique view on requirements evolution. They differentiate between stable and changing requirements, where the changing requirements can be of the following types: *Mutable, Emergent, Consequential, Adaptive* and *Migration*.

On the other hand, this distinction could be attributed to their exclusive emphasis on the evolution of requirements following the deployment of a software product [20, 21]. In Section 3.6 these tags are discussed more in depth.

Taking into consideration the preceding points, it is evident that there exists a notable research gap in the domain of Conversational RE, specifically regarding the investigation of requirement evolution within real-world case study settings. In this research, we aim to bridge this gap and contribute to the existing body of knowledge.

Chapter 2

Research Approach

2.1 Research Question

The gap described in the introduction makes it interesting to research the subsequent requirement elicitation conversations that are underrepresented in the Conversational RE domain, combined with the requirement evolution that can be identified in those conversations. Therefore, the main research question is be:

> ***RQ:*** *What types of requirement evolution can be identified through the course of a software project?*

In order to address this research question, we define a number of research sub-questions

The first sub-question is specifically about the current state of research on this topic. The goal of this question is to find out what is already known about the evolution of requirements, to see whether there is a framework or tagging system available for identifying requirement evolution. The literature research surrounding this question builds a strong foundation for the empirical research in the case study.

> ***SQ1:*** *What knowledge is available regarding requirements evolution in the literature?*

The second sub-question pertains to the potential observation of a pattern in the evolution of requirements during the execution of real-life projects. As already stated in the introduction, there is only limited research on the topic of evolution of requirements, especially empirical research is lacking. With this sub-question, the goal is to translate the empirical findings about the evolution to elements of a more general theory of requirements evolution.

> ***SQ2:*** *What is a typical evolution pattern for requirements?*

The third and last sub-question focuses on the causes of an evolution of a requirement. Similarly to SQ2, this question contributes to the empirical evidence in the requirement evolution domain. Besides the patterns that can be identified in the evolution, the origin is an interesting aspect as well. This question helps identify what triggers can result in what type of evolution of a requirement. With this identification, even a predictable character might be recognized.

SQ3: *What are triggers that can result in an evolution of a requirement?*

Besides these research questions, three hypotheses are tested. The first hypothesis, H1, relates to SQ2, where the research question is focused on what typical evolution patterns are, and the assumption in this hypothesis is about whether there are such patterns that are case independent.

H1: *It is possible to identify case independent evolution patterns.*

The second hypothesis relates to the last sub-question (SQ3). Here a closer look is given to the origins of evolution patterns, and whether different origins result in different patterns. This hypothesis is based on the intuition that an evolution that comes from a requirement initiated by a stakeholder will be different than one that is initiated by the requirement analyst, based on the analyst generally making assumptions and a stakeholder probably having less experience in the field of formulating requirements.

H2: *Requirements with a different origin will have a different evolution pattern.*

Additionally to H2, the third hypothesis talks about differences in evolution patterns, based on the phase of the project when the requirement was added. For example, whether the evolution of a requirement added in the development phase will be different than a requirement added in the specification phase.

H3: *Requirements that are added later in the project will have a different evolution than requirements that are added from the beginning.*

2.2 Research Method

Neither the domain of Conversational RE, nor the topic of requirements evolution have been studied extensively. Researchers already called out to do more empirical research on this topic [7, 2]. The most fitting method for this research is an exploratory case study, as it is quite an undiscovered part of research and we want to add to the empirical research [22]. The data collection will be of qualitatively nature. For the data analysis, the qualitative data will be coded. Coding is the identification of segments of meaning in data and labeling them with a code [23] or more specifically defined by Saldaña [24] as:

> "*a word or short phrase that symbolically assigns a summative, salient, essence-capturing, and/or evocative attribute for a portion of language-based or visual data*"

Coding is used as it enables interpretation and analysis in an artful and creative way of the data [23].

We started with a structured literature study, this is done to lay a strong theoretical basis for the empirical research. We identified the main topics as stated in Table 2.1, for each main topic we used the main search queries below them in google scholar to gather research on the topic. Some extra search queries were used with slight deviations, for example singular terms in plural or the other way around, Requirements Engineering as RE and the other way around, and Requirement as Requirement Engineering and the other way around.

Conversational Requirements Engineering	*Requirements Traceability*	*Changes in Requirements*	*Evolution in Requirements*
"Conversational Requirements Engineering"	"Requirements" and "Traceability"	"Change" and "Requirements"	"Evolution" and "Requirements"
"Conversational RE"	"Requirements" and "Trace"	"Change" and "Requirements Engineering"	"Evolution" and "Requirements Engineering"
"Conversation" and "Requirements Engineering"	"Requirements" and "Trace" and "Requirements Elicitation"	"Difference" and "Requirements"	"Evolution" and "Software"
"Conversational Techniques" and "Requirements Engineering"	"Requirements" and "Link" and "Elicitation"	"Development" and "Requirements"	"Developing" and "Requirements"
		"Modification" and "Requirements"	"Transformation" and "Requirements"

Table 2.1: Search queries used for the literature review

With these search terms we started the *literature review*. It is worth noting that in literature requirement change and requirement evolution are quite closely related and sometimes even used interchangeably, therefore search terms for one of the topics, also lead to some interesting research on the other topic. Results were selected based on language (only English works were used), them being published (no grey literature was used), and both perceived usefulness and quality.

Additionally, the backward snowballing technique is used to find more related works on the topic. For some researches, forward snowballing is used as well. For example works that

propose a definition or a method, to see if other researches have adapted such a proposed definition or method.

We aim to follow three cases for this research. We aim to attend most of the *requirements elicitation conversations* and record them. When we cannot be present, the conversations will be recorded by other participants of the meeting. A rough transcript will be automatically generated by Microsoft Teams, this is done live and we will be able to take some additional notes where needed.

Another piece of data that needs to be gathered before analysis can take place is the set of requirements. We collect both the *requirement specification document* that is provided by the requirements analyst, as well as an *extraction of the requirements from the requirements management system*.

Having these different origins for the data gathering creates triangulation, which helps to test the validity and create a better understanding of the phenomena happening.

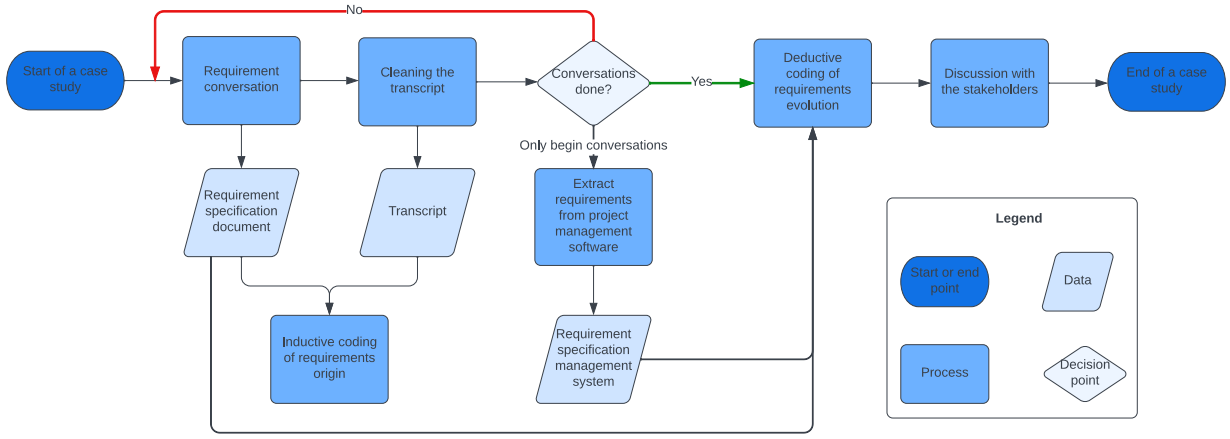

Figure 2.1: Flowchart representing the case study analysis

In Figure 2.1 the flow of the analysis of a case study is represented. We use two types of coding. For the traceability part of this research [25, 26], the analysis to link requirements to their origin, we use inductive coding. The inductive approach is when codes are based on the actual data and what it says [23]. So the roles the participants have in the case study, will be used as codes.

For the analysis in evolution, we will mostly use deductive coding, as the basis of the code will be from theory and another already existing framework or taxonomy [23]. There will be some inductive coding as well as the basis set of codes from theory can be extended when needed.

So the *requirement specification document* and the *transcript from the requirement conversations* are the main artefacts for the traceability part of the analysis. After the starting conversations are finished, the requirements can be extracted from the *project management software* that is used. This artefact, and the *requirement specification document* received from the requirement analyst, are used to analyze the actual evolution of the requirements.

Both analyses take place in an excel sheet where an origin and changes are recorded. For these steps we will code independently of each other and check the inter-rated agreement. Besides that, we will code the same data at different points in time to increase reliability, and to minimize possible bias in the labeling and analyzing of the data. Once all the data is analyzed, we will discuss the findings with the stakeholders of the cases, to fill in some possible gaps.

When we finished coding both for the origin and the evolution, we will do an overall analysis to find out whether there are common patterns in evolution for various types of requirements, and if there are links to be found between evolution patterns and requirement origins.

For the case studies, a case study protocol is created based on the guidelines as described in Designing a Case Study Protocol for Application in IS research [27]. This protocol is distributed to the participants of the case study. The complete protocol can be found in Appendix A.

Chapter 3

Literature Study

3.1 Requirements Engineering

Requirements Engineering (RE) is a field that has been researched extensively in the past couple of decades [28]. It went from being seen as a way to describe what a system should do and not how, to much more than that [29]. As Zave [30] defined RE:

> "*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.*"

RE relates to the elicitation of requirements, analysis, specification, documentation, validation, verification and maintenance of them [31, 10]. During all of these steps evolution can take place.

3.2 Requirements Elicitation

The elicitation of requirements is a critical process in the requirements engineering field, as it is in charge of the translation between the mind of stakeholders and the concrete work for software developers [10]. When there are discrepancies here in this translation, it can create difficulties along the entire requirement engineering and software engineering road [10]. To elicit requirements, different techniques can be used. The most established techniques are displayed in Table 3.1, a textual explanation of these techniques are described by Sharma and Pandey [11], and Yousuf and Asger [12].

An additional categorisation of the requirements elicitation techniques is often made in research [32, 33]. The main categories are listed below with an example of specific techniques below them, as identified by Sajid *et al.* [32], and Anwar and Razali [33].

Traditional Techniques	Interviews
	Meetings
	Document Analysis
	Questionnaires/Surveys
	Introspection
Contextual Techniques	Observation
	Ethnography
	Discourse Analysis
Collaborative/Group Techniques	Prototyping
	Joint Application Development
	Brainstorming
	Requirements Workshops
	Group Work
	User Scenarios
Cognitive Techniques	Knowledge Acquisition Techniques
	Class Responsibility Collaboration
	Protocol Analysis

Table 3.1: Requirement elicitation techniques [11, 12]

- *Conversational Techniques*
  E.g. interviews, meetings.
- *Observational Techniques*
  E.g. protocol analysis, ethnography.
- *Analytical Techniques*
  E.g. discourse analysis, requirements workshops.
- *Synthetic Techniques*
  E.g. prototyping, user scenarios.

Of these techniques, conversational techniques, especially interviewing, is one of the most matured and most used ones [10, 8]. Interviews itself can be further categorized into subcategories of structured (closed), semi-structured and unstructured (open) interviews [11, 12]. These subcategories are specifications for the interview, but the overall idea of the interviews stays the same. The reason that specifically these techniques are the most employed ones, is because their ability to acquire comprehensive information and knowledge [32, 8]. However, they are more time-consuming than some other techniques, so there are some cons to using interviews [32, 11, 8].

3.2.1 Requirement Reuse

Software or requirements reuse describes the practice of reusing existing software artefacts and their corresponding requirements for developing another product [34]. Besides the elicitation techniques discussed in Section 3.2, another form of acquiring requirements for a system is the reuse of requirements. Besides making it easier to understand a requirement, since it has been used before, and therefore boosting productivity, it has also been recognized as a way to improve the quality of the software [35]. Two main ways are identified in which requirement reuse can be beneficial. The first is that it reduces the analysis time of requirements, secondly, it helps to identify code and code tests to possibly be reused as it was developed with a similar requirement [34, 36]. The reuse of software requirements is possible at any time during the lifecycle of a software project, however, it leads to more benefits in earlier stages of the lifecycle [34, 37, 38]. This applies especially the second benefit mentioned earlier, with code reuse is more beneficial when identified in earlier stages of the project lifecycle. Requirements reuse is mostly seen in the non-functional requirements, however, there is not a well-defined reuse method, therefore it is not widely used in regular practice [39].

Irshad *et al.* [34] list multiple approaches to requirements reuse that they found through a systematic literature review. Table 3.2 contains a short description of all the requirements reuse approaches found by Irshad *et al.* [34]. They are sorted from being identified in most literature to the least. Depending on the cases, and what already existent requirement reuse techniques they work with, any of these approaches might be used and could be traced back as the origin of a requirement. The analogy approach is independent of already existing specific techniques of the companies in the case study. This is the only approach that does not depend on any specific framework or process that should be used by the company. The requirement reuse through the analogy approach can be as simple as a stakeholder using inference in the requirements conversations and the requirements analyst basing requirements of this inference [40, 34].

3.3 Conversational Requirements Engineering

The conversational requirements elicitation techniques have been described often in research [10, 8, 11, 12, 32], however, a systematic approach for conducting research around RE-related conversations has only been proposed recently by Spijkman *et al.* [13]. They proposed the Conversational RE domain as follows:

> "*the analysis of requirements elicitation conversations (in short form, requirements conversations) aimed at identifying and extracting requirements-relevant information.*"

In their work, Spijkman *et al.* [13] talk about *Pre-RS* traceability and *Post-RS* traceability, which is based on the earlier work of Gotel and Finkelstein [41], where RS stands for Re-

Approach	Description
Structuring	The saving or storing of requirements in a specified organized structure so that they can be easily retrieved when required for reuse.
Matching	The comparing and matching of existing requirements using different abstraction levels and approaches.
Analogy	Using inference to reason on the similarity of different things.
Model-based Approach	Modeling the requirements to be able to understand reusable parts of requirements.
Ontology-based Approach	Usage of kowledge-based approaches to enhance the requirements reuse, as requirements engineering as a knowledge intensive process.
Parameterization	Creating requirements with a static and a variable part. The actual behavior of the system is described by the values of the parameters of the variable part.
Requirements Traceability	Providing a link between different requirements which helps identifying the reusable requirements.
Domain Analysis	Used in product line context to develop requirements that can be used in similar applications.
Feature Interaction Management	Uses interaction management in the context of requirements reuse to facilitate reusability.
Machine Learning Assistant	Using a maching learning tool to suggest the most relevant use-cases for reuse.

Table 3.2: Requirement reuse approaches identified by Irshad *et al.* [34]

quirement Specification. The *Pre-RS* phase is between the source of a requirement and it being documented in a specification, *Post-RS* is the phase after a requirement is included in the specification [13]. The *Pre-RS* phase is where the initial conversations happen when a conversational technique was chosen in a project, and therefore also the main focus area in the Conversational RE domain. This phase, therefore this domain, are not that in depth researched yet, possibly because of the lack of available data from this phase [13, 42]. Whereas the *Post-RS* phase for example has code as a data artefact, *Pre-RS* has conversations. These conversations are generally more difficult for researchers to access than code is [13].

Conversational RE sets those requirements elicitation conversations as central RE artefacts [13]. There have been a couple of studies, identified by Spijkman *et al.* [13], that work with requirements conversations as central artefacts, both in real-world conversations as in simulated ones [43, 44, 45, 46]. However, this is only a small number of studies whilst the importance has been clear for a long time [41], and researchers have called out that more research has to be done on this topic [7]. A reason for this scarcity of research on the topic can be linked to the lack of availability and accessibility of the main artefacts for this kind of research. A case study that contributes to this field is therefore important and the right choice for this research.

3.4 Requirements Traceability

To connect evolution to its origin, there needs to be a trace. Requirement Traceability (RT) is about specifying a logical connection between deliverables or artefacts of the software development process [26, 25]. In other words, as Lin and Chen [47] describe it:

> "*the ability to describe and follow the life of a requirement*"

Traceability is seen as an essential part for successful software systems, and even as a measure for the reliability for software [25]. It is also seen as an effective way to manage requirement change [48, 47]. Kotonya and Sommerville [49] classified four requirements traceability types, which can be found in Table 3.3 with a short description [49, 47]. These four types of traceability expose different angles. For this research, we mostly use the type *Backward from Traceability*, as we link the requirements back to its origin or source.

Traceability Type	**Description**
Backward from Traceability	Links the requirements to the document source or the person who created it.
Forward from Traceability	Links the requirements to design and implementation.
Backward to Traceability	Links design and implementation back to the requirements.
Forward to Traceability	Links documents preceding the requirements to the requirements.

Table 3.3: Traceability types identified by Kotonya and Sommerville [49]

Even though there has been quite some research on the topic, and even many models, tools and frameworks have been proposed [47], they are not widely used in practice. Some do not see the need for traceability [48, 47] or the commercial available tools do not support specific practices [50, 51, 47], often the cost outweighs the benefits for the practitioners to conduct traceability [47, 51].

3.5 Changes in Requirements

Requirements always change. One reason for this is that they are situated in an environment, and dependant on stakeholders' needs that continuously change [52]. This change in requirements is seen as one of the most persistent problems in the software industry [53], as it is one of the most prominent reasons for project failure [52]. As also stated in the systematic literature review by Bano *et al.* [54], changing requirements are challenging and play a vital role in the success or failure of an project. These changes in requirements happen during the whole process of the software development life cycle, and various causes have been identified in literature, which can also occur during any phase of the software development life cycle [54]. Because the causes for change, and the requirement changes itself can happen during any phase of the process, being able to identify those changes and anticipate on them is difficult but crucial [54]. Working with agile methods increases the developers trust in the

ability to anticipate those changes [54]. The positive effects early change anticipation could have on software projects are as follows [55, 54]:

- Reducing the cost
- Reducing overall development time
- Increasing the success rate

On the other hand, failing to anticipate early on requirement change could have the following effects on software projects [56, 54, 57]

- Delays
- Problems in configuration
- Defects
- Requirement inadequacy and ambiguity
- Difficulties in traceability
- Overall customer dissatisfaction

Thus, it is known that changes in requirements happen often and at any given point in the software development life cycle, and that lacking to anticipate early on them can result in negative effects on a software project. However, the causes of requirement change are not that evident. In 2009 a taxonomy of the sources of requirements change was proposed by McGee and Greer [58], however, as noted by Bano *et al.* [54], and Naseer and Shoaib Farooq [59], this taxonomy and research are based on non-empirical evidence. Therefore, these systematic literature reviews [54, 59] added empirical evidence to their research, as it is important to understand the causes of requirement change to better manage the impacts on software projects [54, 59].

The main causes identified in both of those systematic literature reviews [54, 59] are displayed with a short description in Table 3.4. In the researches, itself a more in depth analysis and sub categorisation can be found.

This is not an exhaustive list as there are numerous causes identified in literature [54], however, this gives an overview of some of the more popular causes of requirement change that are identified from empirical evidence.

The next important question is, when such a cause invokes change, what is that change. Requirements can be seen as two types, stable and volatile requirements. Of the volatile requirements, the following types of change are identified [2, 17, 60, 61, 4]:

- *Addition*
  The inclusion of a new requirement to the requirement specification.

Main Cause of Requirements Change	Description
Customer Needs/Market Demands	A change in customer trend and market demand or supply can cause requirement change.
Changing Environment	A change in the technology, environment and third party hardware or software can cause requirement change.
Organizational Consideration	A change in organizational policies, laws, strategies and goals can cause requirement change.
Functionality Enhancement due to External Demands	A change initiated by the customer as a functionality enhancement request, or changing test scenarios and error correction can cause requirement change.
Increased Understanding of the System	A change in the knowledge of stakeholders/developers of the system can cause requirement change.
Functionality Enhancement due to Internal Demands	A change initiated by the developer as a functionality enhancement request, or changing test scenarios and error correction can cause requirement change.
Requirement Uncertainty	A change due to ambiguous vision, partially evaluated business case, communication issues. conflicting requirements or the dropping of a feature can cause requirement change.
Completeness and Correctness of Requirement Specification	A change due to incorrect, inaccurate or incomplete requirements specification can cause requirement change.
Inadequate Training or Manual	A change due to omitted or partial training/manual can cause requirement change.

Table 3.4: Main causes of requirements change and a description as identified by Bano *et al.* [54], and Naseer and Shoaib Farooq [59]

- *Modification*
  The alteration of an existing requirement from the requirement specification.

- *Deletion*
  The removal of an existing requirement from the requirement specification.

Requirements can change at any time, changing requirements are inevitable and failing to respond accordingly can have many negative effects on the overall software project. There are three types of requirement change, addition, modification and deletion, and these can be caused by numerous activities. Being aware of this and working agile have been found as helpful methods to reduce the negative impact it can have.

3.6 Evolution in Requirements

In literature, we see that requirement change and requirement evolution are often used interchangeably, despite others stating there is a definite difference between the two [2]. This sparks the question of what actually is the difference between requirement change and requirement evolution. When looking at the literature on both topics we can see that there is a shared understanding as to what requirement change is, namely the addition, modification or deletion of requirements [2, 17, 60, 61, 4]. However, such a shared understanding is not present for requirement evolution [2], despite requirements evolution being a topic in quite some researches [2, 3, 4, 1, 5, 6, 19, 20]. Therefore, no set boundaries as of what is part of requirement evolution are present. This also resulted in some researches using requirement change and requirement evolution interchangeably [2]. However, when purely looking at the definitions of change and evolution in dictionaries, we do see a difference between the two.

> *Change* - to make or become different [62]
>
> *Evolution* - a gradual process of change and development [62]

In these two definitions we see that evolution always comes with change, as it exists of change, but change itself does not necessarily mean evolution.

Based on the definition of evolution from the Webster Dictionary, combined with their systematic literature review, Li *et al.* [2] proposed the following definition:

> "*Requirements evolution is a process of continuous change of requirements in a certain direction. It can happen during the entire software life cycle excluding the definition phase. The propagation of requirements evolution spans from requirements to maintenance of a software system.*"

Ernst *et al.* [15] take it even a step further, and they state that requirement evolution is part of software evolution, and focuses only on phases after a software artefact has been developed.

Ferrari *et al.* [6] have the most holistic view of requirement evolution, they do separate two different phases in their research. First, the *Early Requirements Evolution*, which is the requirement evolution that can start at the initial idea. Secondly, the *Late Requirements Evolution*, which is the evolution once the system is deployed. Both these phases are analyzed in their work.

In Figure 3.1 the timeline can be found for when requirements evolution takes place according to Ernst *et al.* [15], Li *et al.* [2] and Ferrari *et al.* [6]. For Ernst *et al.* [15], the requirements evolution can only start when there is a first software product present, Li *et al.* [2] state that it can start after the definition phase, which is after a first requirements specification is present, and Ferrari *et al.* [6] state that the evolution of requirements can start right at the beginning of a project, when an initial idea is present.

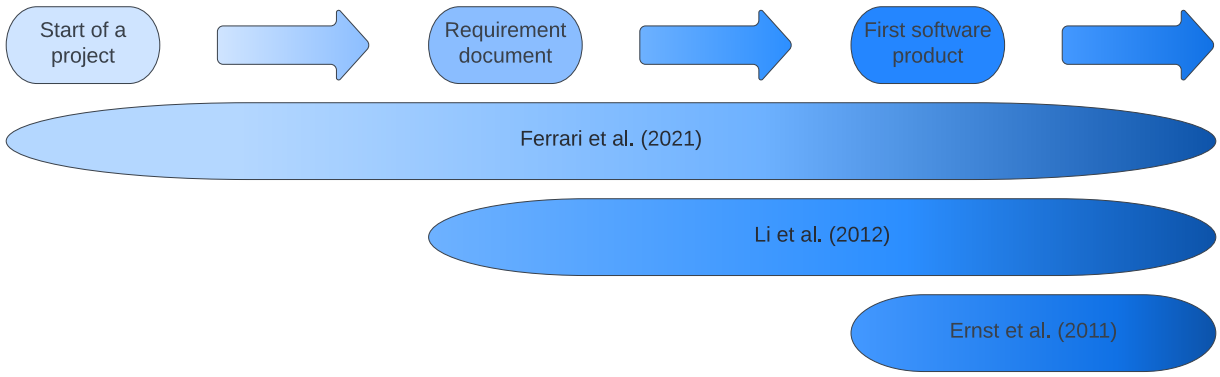

Figure 3.1: Timeline of a project and when requirements evolution can happen according to Ernst *et al.* [15], Li *et al.* [2], and Ferrari *et al.* [6]

To have an as complete as possible overview for this thesis, we will look at requirements evolution similarly as Ferrari *et al.* [6], where we will observe them over the whole timeline. As they make clear in their research, requirements evolution can take place from the very start where a stakeholder has an idea.

The *initial idea* from Ferrari *et al.* [6], from where evolution can start, is based on the concept *pre-requirements* of Hayes *et al.* [63], which is the information available prior to requirement specification. So the part of Figure 3.1 where Ferrari *et al.* [6] and Li *et al.*[2] do not overlap.

We were not able to find research that identified typical evolution patterns, or what such requirement evolution can look like. However, there are a couple of researchers that identified a set of steps that together in a sequence display evolution. Or when looking at the definition for evolution (Section 3.6), changes and development that together can form evolution.

In the work of Ferrari *et al.* [6], the following tags are used in their taxonomy:

- *Existing*, **E**

User stories that express content that was already entirely present in the initial set of user stories.

- *Refinement*, **R**
  User stories that express content that is novel with respect to the initial set of user stories, but that belongs to one of the existing high-level categories of the initial set.

- *New*, **N**
  User stories that express content that is novel, and belonging to a novel category not initially present.

Besides these actual changes in the requirements, the following items are identified in that research:

- The name of novel categories of user stories introduced.
- Recurrent themes in **R** and **N** stories.
- Roles that were used also in the original stories.
- Roles that represented a refinement of roles used in the original stories.
- Roles that were novel and never considered in the original stories.

The referred *initial set of user stories* from the tags above, comes from the controlled experiment setting. In the work of Ferrari *et al.* [6], they do a controlled experiment where someone plays the role of customer. This customer has a list of requirements drafted so that the person playing the customer knows what kind of product he wants, and has to talk about with the interviewers. This initial list is referred to with the phrase *initial set of user stories*.

The work of Harker *et al.* [20] has a whole different set of tags that are used to describe changes and developments that together form requirement evolution according to them. They first differentiate between stable and changing requirements, where the stable ones do not go through an evolution, and the changing ones do. The following types of change are identified by Harker *et al.* [20]:

- *Mutable*
  Arises in response to demands outside the system and its development process.

- *Emergent*
  Arises when there is a need to clarify and broaden the target and also to enhance the sense of ownership and commitment to the eventual solution amongst the stakeholders.

- *Consequential*
  Changes that arise after experiencing the system, the technology itself acts in such a way as to cause the requirements or users to evolve.

- *Adaptive*
  Emerges when there is a need for change to be an inherent and on-going capability of the delivered system.

- *Migration*
  Changes that arise when one wishes to migrate a system from one state to another, these are often constraints an existing system creates when needing to transition from one state to another.

In the work of Harker *et al.* [20], requirements evolution is seen as something that happens after a software product has been deployed, similarly like Ernst *et al.* [15]. This also reflects on the tags as some refer to an existing system.

Anderson and Felici [19] did a case study on the topic of requirements evolution as well. They proposed the following taxonomy for changes that they identify as part of requirement evolution:

- *Add, Delete and Modify requirements*
  Requirements are changed due to the specification process maturity and knowledge.

- *Explanation*
  The paragraphs that refer to a specific requirement are changed for clarity.

- *Rewording*
  The requirements itself does not change, but it is rephrased for clarity.

- *Traceability*
  The traceability links to other deliverables are changed.

- *Non-compliance*
  A requirement that is not applicable for a new software package. This is the case when the requirements specification is based on that one of a previous project.

- *Partial compliance*
  A requirement that is applicable partially for a new software package. This is the case when the requirements specification is based on that one of a previous project.

- *Hardware modification*
  Several changes are due to hardware modifications. This type of change applies usually to hardware dependent software requirements.

- *Range modification*
  The range of the variables within the scope of a specific requirements is modified.

- *Add, Delete and Rename parameters/variables*
  The variables/parameters to which a specific requirement refers can change.

This taxonomy also takes the *Early Requirement Evolution* into account.

There are also other researchers that do not propose unique change types as part of evolution, but work with the types of requirement change from Section 3.5, so addition, modification and deletion [2, 64]. This does have some overlap with both the tags from Ferrari *et al.* [6] and from Anderson and Felici [19], where both addition and a form of modification (refinement) are used as well. We have made comparison of the three earlier discussed proposed evolution tags, shown in Table 3.5. This comparison is based on the *objective*, so whether the evolution change is about the core of the requirement, or non requirement, which can be a description additional to the requirement or other system features not documented as requirements. The other comparison is made on the change type, where a change can be an *addition, mutation (or modification)* or *deletion*. Some evolution changes can encompass multiple options, for example a hardware modification can initiate an addition, mutation but also deletion. The viewpoint is the last aspect of the comparison. Here the viewpoint of the name given to the evolution change as well as the description and additional explanation from the research they were first proposed in is used. We differentiate between three viewpoints; *event, status* and *change*. Event is for the evolution changes that focus on the event that initiates the change, for example hardware modification where the event of modifying the hardware initiates a change. Status is for evolution changes that look at the state of a requirement, for example existing. The last viewpoint is change, where an actual change is described, so not the event of hardware modification, but the change of add requirements.

In the objective and viewpoint categories a checkmark (✓) is used to check which type is applicable. For readability, the change type makes use of different checkmarks, *addition* is checked with '+', *mutation* is checked with '∼' and *deletion* is checked with '-'. The middle columns now do not have to be traced to the upper row to find out what the check is applicable to.

Additionally, in Figure 3.2 the evolution changes with the requirements objective are displayed independent of viewpoint. So the evolution changes with the same change type are grouped together. This is an interesting overview as the names of the evolution changes might differ a lot, as well as the viewpoint, but what they actually describe can be quite similar.

This shows again that there is not one agreed upon taxonomy as to what requirements evolution exists of. When comparing the four options described above, the requirement change [2, 64] and the one as proposed by Ferrari *et al.* [6], might seem too shallow. As they only have fairly high-level actions as possibly part of evolution. However, Harker *et al.* [20] only focus on the requirement evolution after a software product is deployed, which might make the tags less fitting for requirements before a product has been deployed.

This makes the taxonomy of Anderson and Felici [19] as the most fitting one for our research. However, some slight alterations will be made, to have a clearer overview, and to

Literature Origin	**Evolution Changes**	**Objective**		**Change type**			**Viewpoint**		
		Requirements	*Non Requirements*	*Addition (+)*	*Mutation (∼)*	*Deletion (-)*	*Event*	*Status*	*Change*
Ferrari *et al.* [6]	Existing	✓						✓	
	Refinement	✓			∼				✓
	New	✓		+				✓	
Harker *et al.* [20]	Mutable	✓			∼		✓		
	Emergent	✓		+			✓		
	Consequential	✓		+	∼	-	✓		
	Adaptive	✓		+	∼		✓		
	Migration		✓	+			✓		
Anderson and Felici [19]	Add requirements	✓		+					✓
	Delete requirements	✓				-			✓
	Modify requirements	✓			∼				✓
	Explanation		✓	+	∼				✓
	Rewording	✓			∼				✓
	Traceability		✓		∼		✓		
	Non-compliance	✓				-	✓		
	Partial compliance		✓		∼		✓		
	Hardware modification	✓		+	∼	-	✓		
	Range modification		✓		∼		✓		
	Add parameters/variables		✓	+					✓
	Delete parameters/variables		✓			-			✓
	Rename parameters/variables		✓		∼				✓

Table 3.5: Comparison of evolution as described by Ferrari *et al.* [6], Harker *et al.* [20], and Anderson and Felici [19], with different checkmarks used for the middle *Change Type* columns for readability purposes

incorporate the requirements change and the proposed work of Ferrari *et al.* [6] more into it, the *Add, Delete and Modify requirements* will be split up into *Add Requirements*, *Delete Requirements*, and *Modify Requirements*.

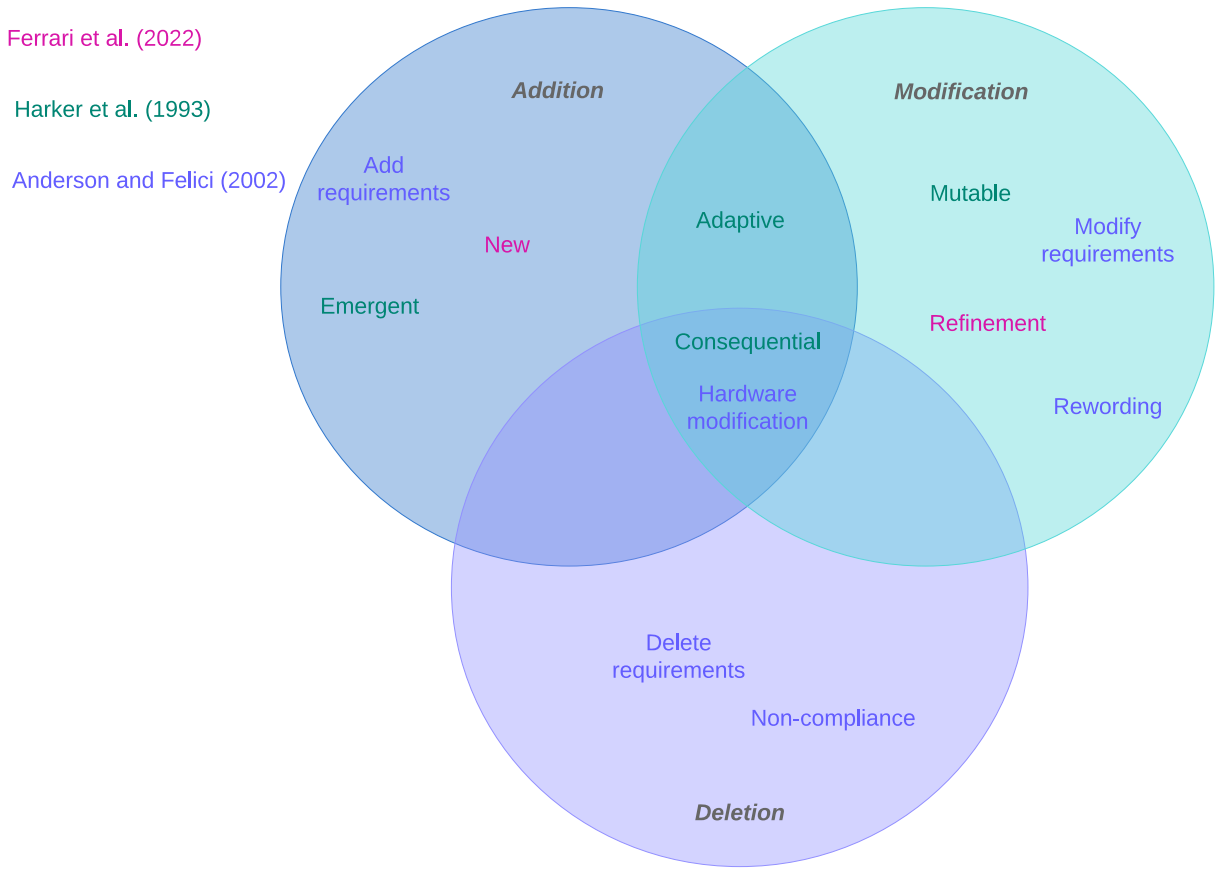

Figure 3.2: Comparison of the change types with the requirements objective, independent of viewpoint

Chapter 4

Case Study

4.1 Preparation

To prepare for the case study, we have created a case study protocol, which is presented in detail in Appendix A. In this case study, we aim to follow a software projects from start until a finished minimum viable product (MVP). As explained before, the difficulty with researching in the Conversational RE domain is in the lack of access to the data. Requirements elicitation conversations, one of the main artefacts of this thesis, are not always documented by recording or transcribing them. Therefore, it is important for the case study in our research to start together with the start of a software project, so that all conversations deemed relevant can be recorded and transcribed proactively. This way, we enable the gathering of data that is generally not saved.

4.2 Description

Although we initially began with three case studies, two of them were abandoned as they did not progress through all the phases of the protocol. However, they still served as valuable references and contributed to our overall understanding. As a result, we have shifted our focus to a single case study that encompasses the complete protocol, allowing us to concentrate our efforts and gain in-depth insights.

The cases that were dropped were both at the same company, a full-service IT company. Customers can come here for anything IT related, from online workplaces to software solutions. The company is medium sized, with around 25 employees. The clients of this company are from all different industries. The two projects at this company are both internal projects, the first is to create a system where calling plans for customers can be noted down. This project had one requirements analyst, one product owner, two developers and three key stakeholders, who will also be the end-user of the system. The second project is to create a system for the company administration. This project had one requirements analyst, one

	Duration (Months)	Number of Recorded Conversations	Hours of Recorded Conversations	Number of Employees	Number of Involved Stakeholders	Project Management System	Number of Requirements in Requirements Specification Document
First case that did not finish	3	2	4	25	7	Trello	31
Second case that did not finish	4	3	6	25	8	Trello	0
Case that was followed until the MVP development	10	7	7	20	10	Jira	20

Table 4.1: Overview of some properties of the different cases of the case study

product owner, three developers and three key stakeholders. Due to some changes at this company, the two cases did not progress after the second phase. The first project had had two meetings of around 2 hours each by then, and a requirement specification document of 31 requirements. The second one had had three meetings of around 2 hours each by then, and no requirement specification document, as the scope of the project shifted at every meeting. The focus of this thesis is therefore on the third case at a different company.

The case study we followed from beginning to the development of the MVP takes place at a software development company that automates analog processes for different business fields, offering flexibility and scalability [65]. The company of the case study works mostly with low-code development platforms. Low-code development platforms works with predefined blocks of code that developers can drag-and-drop to create applications of different forms [66]. It operates model-driven, which makes it possible for developers to create a set of models which a low-code platform interprets to generate executable code. The company is medium sized, with around 20 employees. They mostly work with clients in the wholesale, corporate services, manufacturing and logistic industries [65]. However, for this case study, the company will not work with an external customer, but rather on an internal system. This means that the whole process will be in-house, all the stakeholders work at the company, the development is done by the company it self and the final system is for internal purposes.

The goal of the company is to create a product that supports project management and helps to create an overview of all current projects within the company. The idea from the product started with a project management viewpoint, but it should be extensible so that it can be used widely within the company. Due to the familiarity with the technology according to the stakeholders, it will be developed in the low-code platform Betty Blocks.

Being an internal project, the company considers it an opportunity to coach less experienced employees in new roles, as there is no pressure of an external customer. Within this project, there were one requirement analyst, one product owner, three developers, two testers and three key stakeholders, whom are all project managers and end users of the system. Mainly the requirement analyst and a developer were lesser experienced and they were

assigned this project as an opportunity to improve their skill. They do however get coached by other, more experienced stakeholders within the project.

For this project, the company works with a Jira board, as they do with any other project. The issues in this Jira board can be labeled of type Epic, Story, Task, Subtask or Bug. Stories are mostly functional requirements in the user story format, and most often added by the requirement analyst; however, any member of the team could add and edit issues in Jira. They work in sprints, however, as there is no external customer involved, the priority of this project might not be as high as other projects. Therefore, the sprints will not be as strictly planned as they are in other projects; rather, the aim is to have regular meetings with all stakeholders to talk about the progress that is made.

For the duration of this case study, the main focus will be on creating a tool that supports project management, but the aim after that is to extend it so that it can support and be used by everyone within the company. However, to set a clear end date for this case study, we will only be following the project until the first version (MVP) with focus on supporting project management is done according to the stakeholders. This can however lead to the fact that requirements will be added to the data set, but will not be implemented in this first version yet. This is something to keep in mind when analyzing the data.

4.3 Data Extraction

We rely on two data sources for the main data set that is analyzed first. There is a third data source, the requirements conversations, this data source is analyzed in the later part of the research. We start with the first two data sources. The first one is the requirement specification document and the second the Jira board used in the project. The requirement specification document is in this case a Microsoft Word document where the requirements analyst documented a first set of epics and user stories after the kick-off meeting. This document was then discussed in the second meeting, after which the requirements were documented in the Jira board. The requirements from this document could easily be extracted manually, for the Jira board we first did some research on specialized tools that extract information from Jira. However, none of those tools seemed fitting for the information needed for this research. They were either incomplete or very intrusive for the other company data in Jira. Therefore, we extracted all the information manually. First we describe what information is gathered, and why that information is important, and after that we explain the tags that are used for the tagging of the data.

4.3.1 Data Retrieval

From the two main sources the data is extracted, the requirement specification document is used only at the start of the project, between the first and second meeting. This document consists of *epics* and *user stories*, which are then handled as issues in the project management system, Jira, that was used after the second meeting.

A visual representation of the data that is available in Jira can be found in Figure 4.1. The attributes with an asterisk in front (*) are not extracted as they do not hold information used in the analysis.

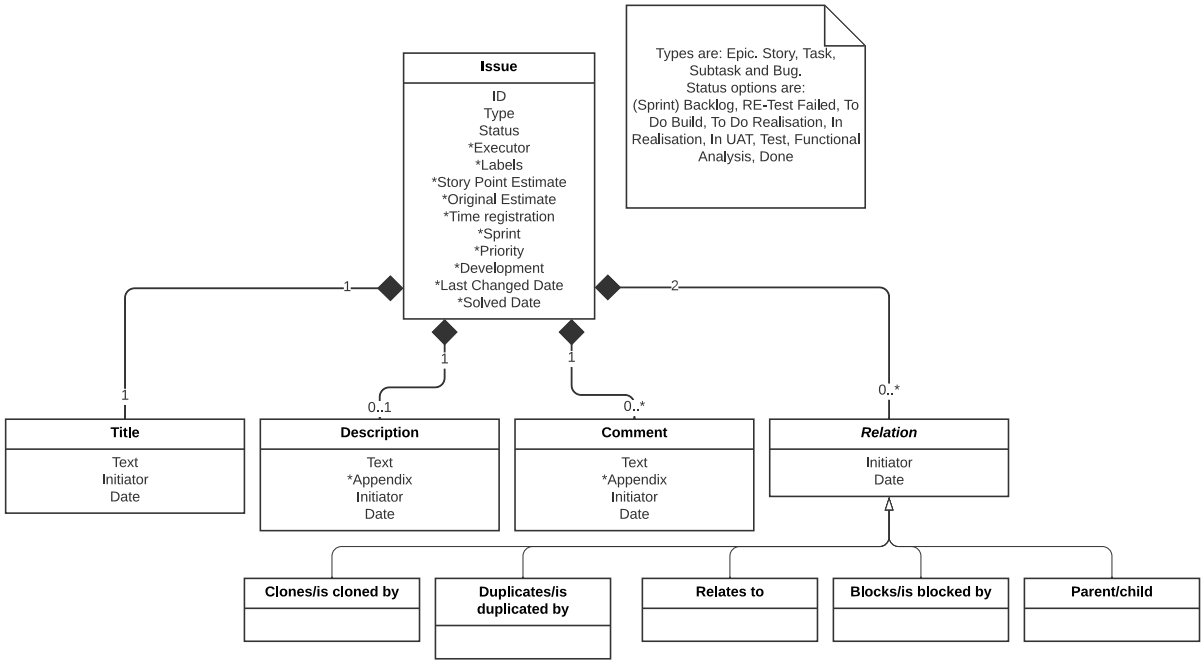

Figure 4.1: Class diagram of the available data in Jira

Each *issue* can be of type *epic, story, task, subtask* or *bug*, and has at least a *title*. Within this company, the *title* is often used to insert the main text of a possible requirement, often in *user story* format. Additionally but not mandatory, each *issue* can have a *description*, a free input field where anything possibly relevant can be added. There is also a *comment* section, where relevant information could be added, but also other discussions or remarks could take place. An *issue* can have multiple *comments*, which follow each other sequentially. Users can create relationships, chosen from pre-defined link types, between two *issues*. Then this *relation* will be added by both *issues*. The link types are *clones/is cloned by, duplicates/is duplicated by, relates to, blocks/is blocked by* and lastly it is possible to have a *parent/child* link between *issues*. An *issue* can have multiple *comments* and *relations*, unlike the *title* and *description*, of which an *issue* only can have one. When changes are made in the *title* and *description* part of an *issue*, the old version is overwritten, and only the version after the change still exists.

Above we described the data that is available in the original sources, but not all available data is useful for this case study. Therefore, we extract the data that is suitable for this research, which is collected in our *data set*. This data set is an excel document with the extracted data, complemented with additional useful information, and is used for the analysis

Title	Description
ID	The ID that is used in the original source.
Text	The text that is extracted from the original source. Containing the requirement or requirement relevant information..
Date	The date on which the change happened.
Type	The type of data entry. Could either be an epic, story, task, subtask or a bug.
Status	The status of the data entry at the moment of change.
Item	The item that held the text of the change. Could be the title, description, comment or relation.
Initiator	The person that made the change.
Relation	The relation between the data entry and another entry.
Sequence	A sequential ID to create a timeline for items with the same ID.
Evolution	The tag most fitting for the data entry.

Table 4.2: Properties of a data entry

of this research. In Table 4.2 we present the information each *data entry* consists of, these are all the items from the class diagram in Figure 4.1, except for the attributes with an asterik (*) in front, these attributes are available in Jira but not needed for our analysis. First, we have the *ID*, this is the same *ID* as the one an *issue* has in the data source. This *ID* is automatically added by the project management software, and is not changeable. After that we have the *text*, which is the literal text from the *title, description, comment* or *relation*. The *date* is the date that the change happened on. The *type* is the type of issue used in the system, and *status* is the status that the issue had at the moment of the change. The options for the *type* and *status* are the same as the ones described in the note in Figure 4.1. The *item* is whether the text came from the *title, description, comment* or *relation*. *Initiator* is the person that actually made the change, so that is mapped from the *initiator* fields from Figure 4.1. Lastly for the mapped values, *relation* is the relation between that entry and another one, so the inherited options for *relation* in Figure 4.1.

Besides these literal extractions, our data set also contains an item where an index number is used, the *sequence*. This is mostly important for issues that have multiple items, so for example a title, description and multiple comments. All these entries will have the same *ID* because they are linked to the same issue, however, it is useful to know the exact order in which they were added or changed. There is also a *date* field that could be used for this, however, this does not work with changes that were made on the same date. Therefore we introduce the *sequence*, which is simply an ascending number series starting at 1 for every new ID. Lastly the *evolution* step is added to the data set, which is the tag fitting for the evolution in that data entry. The tags are explained more in depth in Section 4.3.2. Table 4.3 shows a subset of the *data set* to showcase the possible values of the properties.

For our own data set we will use some specific terminology. To start off, the *data set* is the excel document in which all the data being used for the analysis is collected. This *data*

ID	Sequence	Text	Evolution	Date	Type	Status	Item	Initiator	Relation
62	1	As a user, I want to be able to see all relevant contacts on the project details page, so that can find contacts I need easily.	Add	08-09-2022	Story	Backlog	Title	Requirements Analyst	
62	2	We have a list of contacts that relate to the customer, but also contacts that are directly related to the project. The contacts of the customer should be displayed separately.	Prescriptive Explanation	08-09-2022	Story	Backlog	Description	Requirements Analyst	
62	3	[This issue is blocked by ID 58]	Traceability	08-09-2022	Story	In Realisation	Relation	Requirements Analyst	Is blocked by ID 58
62	4	[Link to system] I have created a list with the contact that are linked to the customer, and a list of the other contacts.	Traceability	14-09-2022	Story	In Realisation	Comment	Developer	
62	5	Maybe an additional requirement could be to be able to filter on this on the overview page.	No Evolution	21-09-2022	Story	Test	Comment	Stakeholder	

Table 4.3: Example of a subset of data entries

set consists of *data entries*, which are all the separate items from the original sources, so every title, description, comment or relation change is a new entry, each entry contains the properties from Table 4.2.

Besides that we also work with *sequences*, where a *sequence* a set of data entries that follow one another with evolution tags from one state to the next. A *sequence* can be seen as the evolution flow of a requirement. For the example subset of data in Table 4.3, we end up with a sequence of: *Add, Prescriptive Explanation, Traceability, Traceability.* The tag *No Evolution* is not part of a sequence.

4.3.2 Tagging

After extracting the data, all the data entries are tagged. In Section 3.6 in the literature study, we already discussed different taxonomies used in literature, and as discussed, we use a combination of the taxonomies of Anderson and Felici [19] and Ferrari *et al.* [6] as basis, but made some alterations where we felt they were necessary. The tagging was done in a couple of rounds, where in each iteration the tags and their definitions were reviewed. One author tagged all the data at different points in time, and a second author tagged subsets of the data to support inter-related reliability. Once an agreement was made on the tags and their definitions, and both intra-related and inter-related agreement was formed, we moved to the analysis. The final tags we work with can be found in Table 4.4 with a short description. A more elaborate description can be found below, divided into some categories.

Core Actions

Within this category fall the tags that make up for some core changes in the set of *sequences*, such as expanding and reducing it.

The **add** tag is used when a new requirement is introduced with its own unique identifier. This tag is the indicator of the start of every new requirement its evolution, each *sequence*

Tag	**Short Description**
Add	Introducing a new requirement with its own unique identifier.
Delete	The removal of a requirement from the project, with no possibility of it returning with the same unique identifier.
Refinement	Introducing an extension to an existing requirement in the form of a new requirement, and such, depends on the original one. This refinement can, but does not has to have its own unique identifier.
Modify	A change in a requirement that alters its meaning. The old version does not exist anymore.
Rewording	A change in a requirement that does not alter its meaning, only the textual formulation. The old version does not exist anymore.
Traceability	The addition of a clickable link to another artefact that relates to the requirement.
Indirect Traceability	The addition of a non-clickable link in the form of an image or text to another artefact that relates to the requirement.
Prescriptive Explanation	The addition of an explanation to clarify the requirement, with conditions the requirement should comply to.
Descriptive Explanation	The addition of an explanation to clarify the requirement, which does not add extra conditions or constraints.
Typo	Where the person adding or changing text made a typo, which is fixed later.
No Evolution	The tag to use when there is no other tag above applicable. This may happen when remarks are made in comment sections not related to the requirement, or when there are bugs instead of requirements.

Table 4.4: Tags used in the data set with a short description

in the data set starts with an **add** tag; there is no further evolution if there has not been an **add**. This tag can only be identified by entries that have their own unique identifier in the project management document or Jira, and an **add** can only occur in a *sequence* once.

The **refinement** tag is used when an extension to an existing requirement is introduced. This can be either in a new or in the same issue as the existing requirement. It is often structured as a more granular requirement than the existing one, extending it with more detail, and typically mentioned in the issue of the original requirement. The following two data entries from the data set showcase where the **refinement** tag is used. The first entry is the *title* of a requirement, which has been given the **add** tag, and the second entry is an excerpt of a *comment* added to that *issue*. The *comment* is labeled a refinement as it is an extension that heavily depends on the original requirement.

> **Requirement:** "*As a user, I want to see text for the buttons in the project overview, so that the action of the button is more clear.*"
>
> **Comment:** "*... Add this for the tags as well. ...*"

The **delete** tag is the opposite of an **add**, as it removes requirements. This tag should only be linked to actions that are irreversible, and where the requirement is actually expected to be deleted from the project. Archiving them for later use should not be tagged as a **delete**. When a requirement is deleted, there is no possibility of another evolution tag to occur in that *sequence*.

Irreversible Changes

This category holds all the changes that are irreversible, where previous versions of the information get lost, therefore only the new version is used. Due to the nature of the project management system, only the *title* and *description* can be tagged with the tags in this category. The reason for this is that those are the only items of an *issue* that are overwritten.

The **modify** tag is the first tag in this category. This tag is used when a change is made to a requirement, which alters the meaning of that requirement. As it is an irreversible change, the previous version of the requirement gets lost, and this modification is the requirement that will be worked with in the project from that point onwards.

The **rewording** tag is the other tag in the irreversible changes category. Here a change is also made to the requirement, but it does not alter the meaning of the requirement. It only changes the textual formulation for clarification purposes.

Linking

Here in this category the tags that can be identified by links between artefacts or requirements can be found. Linking is mostly present in requirement relevant information, and less in the actual core of a requirement text.

The **traceability** tag is used when there is a clickable link to another requirement or another artefact of the project identified. The other requirement or artefact that is linked to should be in some form related to the original requirement.

The **indirect traceability** tag is similar to the **traceability** tag, however, this tag is used for indirect links, so not clickable. This is enough information in a text or in an image so that all the stakeholders of the project can find the part that is linked, but they have to reach the target artefact themselves. This can be a screenshot of the product where the requirement is implemented, or an in-text reference to another requirement where the unique identifier is named, but no clickable link is created. Every stakeholder in the project will have the knowledge needed to find the right place to look at the linked item live, but they do have to find it themselves.

Clarification

The clarification category is for the tags that are used to label data that is added to clarify a requirement. It is only an addition to the main requirement, as that main requirement could still exist without the clarification, but adding it has the goal of making the requirement better understandable. A clarification cannot exist without a main requirement text. Therefore, clarifications can never exist in *titles*.

The **prescriptive explanation** tag is used for information that elaborates on conditions that the requirement should comply with. This explanation does not introduce a new requirement itself, it only adds additional context or demands for the requirement. An example from the data set can be found in Table 4.3, and another fictional example can be found below. With this description we have a clarification to the requirement with conditions it should comply to.

Requirement: "*As a user, I want to be able to upload files.*"

Description: "*Accepted file formats should be PDF and JPG.*"

The **descriptive explanation** tag is similar to the **prescriptive explanation**, except that this one does not introduce conditions the requirement should comply to. This tag is used for explanatory information of a requirement which describes suggestions for the implementation or design of the requirement. In the example below we see a requirement

and description of the data set where the description adds a clarification with an example for the requirement, but does not set extra conditions. The description is purely to add context and an example.

> **Requirement:** "*As a project manager, I want to be led through the process step-by-step, so that I can follow the process consistently with a standard working method.*"
>
> **Description:** "*Example of the process can be found in the following link: [link removed for anonymity reasons].*"

Other

This category is for the tags that are not main evolution tags, but to be as concise and complete as possible we want to be able to tag each data entry. For those we differentiate between two specific tags, however, both of these tags will not be used in the *sequences*, they are not seen as part of the evolution flow of a requirement.

The **typo** tag is used when the person entering the data entry made a typo which is later corrected. The data entry where the typo is introduced can be tagged as **typo** where the next data entry that corrects the typo can be tagged regularly again as if the typo was never there.

The **no evolution** tag is used for all other data entries where no earlier mentioned tag can be identified. Here we can say that there is no evolution that took place with that data entry. This is often seen for bugs, as bugs are not requirements, or in comments where someone leaves a comment agreeing with another comment without actually saying anything about the requirement itself.

Chapter 5

Analysis

We analyse the collected data from multiple viewpoints. First, we discuss some basics about the data, including the number of entries and the sequence length. After that, we look into the frequency of various tags within the data set, and explore the likelihood of different tag combinations occurring together. Then, the focus is on the timing of the tags, followed by an analysis of both the sources and the initiators of the changes. Lastly, we jump to a more granular level and analyse the requirements conversations.

5.1 Analysis Overview

In Section 4.3, we provided an explanation of all the terminology and the extracted data, which consequently comprises our data set. A small recap is that we mostly reference the project management system where every entry is an *issue*, which can be of type *epic, story, task, subtask* or *bug*. Every *issue* has at least a *title*, but can also have one *description* and multiple *comments* or *relations*. With our data set, we have each of these *titles, descriptions, comments* and *relations* as data entries. Besides that, we work with sequences, which is an evolution flow from one main requirement, so a sequence of evolution tags which always starts with an `add`. In this first part of the analysis, most focus is on this data set, with mostly information gathered from the project management system. So the granularity is set on changes that were actually documented, not changes that were only discussed outside of the project management system. With the last part of the analysis we do a conversational analysis where we work on a more granular level, there we also take into account what happens before the actual change is documented.

5.1.1 Data Set

Our data set consists of 413 *data entries*, several of which could not be tagged as being part of an evolution and received either the tag `no evolution` or `typo`. These were mostly remarks in the comment section that were not requirement-relevant, bugs, or changes that occurred to rectify spelling errors. In the end, 228 of the *data entries* could be tagged as an

ID	Sequence	Text	Evolution	Date	Type	Status	Item	Initiator	Relation
1	1	As a user, I want to be guided through a checklist, so that I will not forget any steps.	Add	01-08-2022	Story	Backlog	Title	Requirements Analyst	
1	2.1	The checklist from here [link to system to display processes] is implemented here [image of the system]	Traceability	20-08-2022	Story	In Realisation	Comment	Developer	
1	2.2	The checklist from here [link to system to display processes] is implemented here [image of the system]	Indirect Tracebility	20-08-2022	Story	In Realisation	Comment	Developer	
1	3	The checklist is now implemented here [link to the system], but I want it to be implemented here [link to the system]	Traceability	25-08-2022	Story	In UAT	Comment	Key Stakeholder	

Table 5.1: Mock example to illustrate how multiple tags for one entry were handled

actual evolution step. Some of these *data entries* had multiple tags within the same entry, therefore we could tag the evolution tags 248 times. For several entries, we could not assign a single tag, and we therefore decided to opt for a multi-label annotation, where a single entry can take more than one tag. These tags were then ordered by occurrence order. However, we did only tag an entry with multiple tags if those tags are unique in that entry.

For example, the mock data in Table 5.1 illustrates this. Here we see an entry with both a clickable link, as well as an image. This would have been **ID** 1 and **Sequence** 2, however, as this entry can be tagged twice, we add it twice as well, once with **Sequence** 2.1 and once with **Sequence** 2.2. The clickable link comes first in the entry, so 2.1 is tagged `traceability`, and 2.2 `indirect traceability`.

Then in Table 5.1, **ID** 1 and **Sequence** 3, we see an entry with two clickable links in it. For this entry, no duplicate is added as both links would be tagged as `traceability`, and entries are only duplicated when multiple different tags are tagged.

Figure 5.1 displays the distribution of the tag categories that are labeled in the data set. The largest part is the *other* category, where the tags `no evolution` and `typo` fall in to. Of the actual evolution tags, the category *linking* is most common in the data set, and the category *irreversible changes* least common.

Figure 5.2 displays the distribution of tags per category, having the *Core Actions, Irreversible Changes, Linking, Clarification* and lastly the *Other* category. The colors correspond with the colors of the categories in Figure 5.1.

91 *Issues* were collected from the project management system, and an additional 2 from the requirement specification document, that were never entered in the system. Thus there are 93 *issues* in the data set. Of these 93 issues the stakeholders labeled them as follows:

- 6 Epics
- 46 Stories
- 9 Tasks

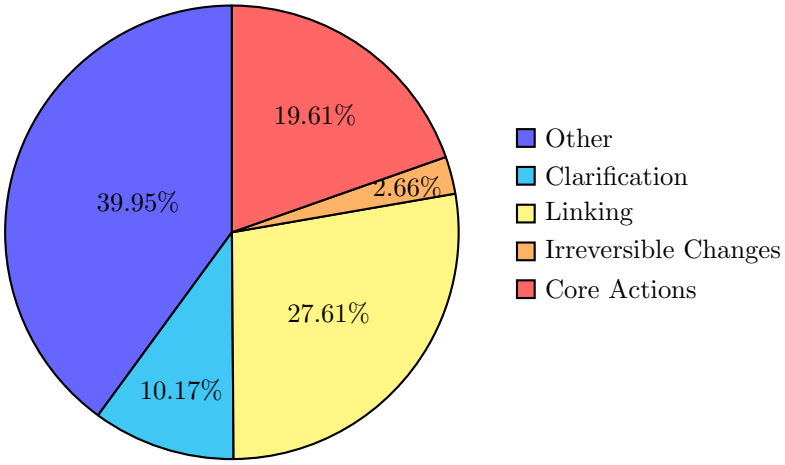

Figure 5.1: Distribution of tag categories labeled in the data set

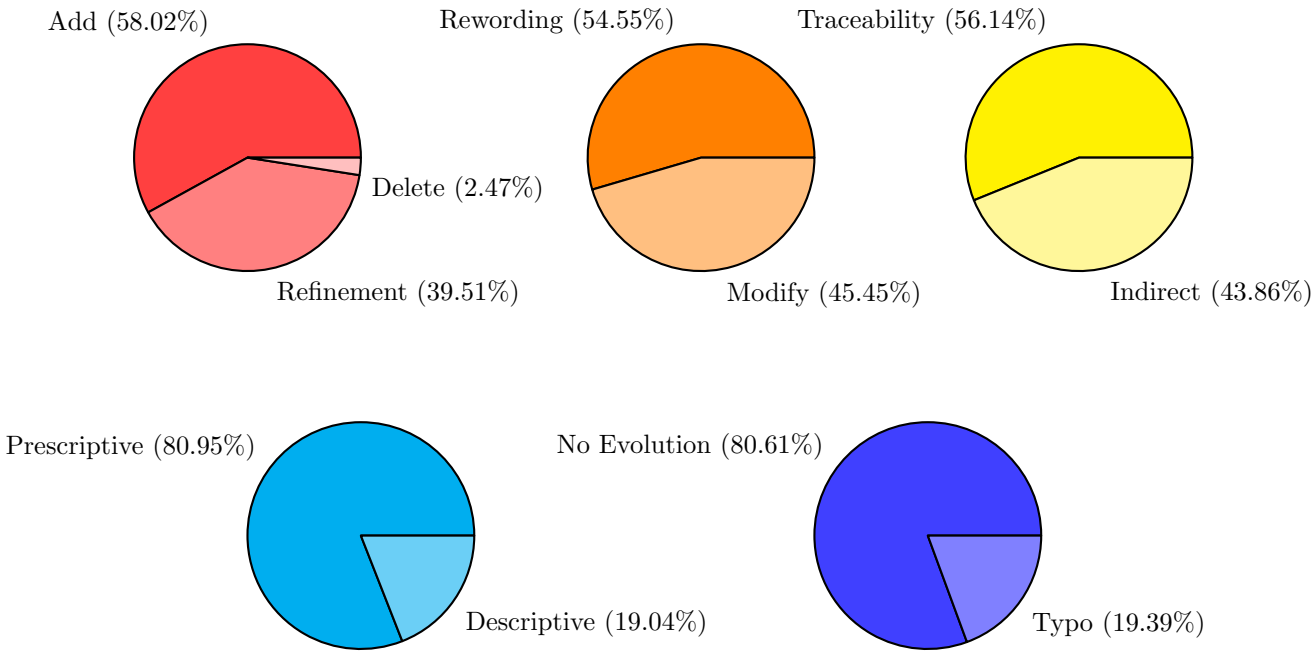

Figure 5.2: Distribution of tags labeled in the data set per category as displayed in Figure 5.1

- 13 Subtasks
- 19 Bugs

From those issues and all their data, we could identify 47 *sequences*, each being an evolution sequence that starts with an `add` tag. These sequences are mostly what we focus on for the remainder of this analysis and thesis, as we are interested in the evolution flow per requirement sequence.

5.1.2 Sequence Length

When talking about the length of those 47 *sequences*, we mean the number of evolution tags within the sequence, so the number of evolution steps that requirement went through. These tags are the ones that actually are part of the evolution, the tags within the *other* category are not taken into account here. Each sequence always has a length of at least one, as a sequence has the property of always starting with an `add` tag, so there is at least one tag. As can be seen in Figure 5.3, there are five sequences of length one, so only consisting of an `add` tag, and the longest sequence has a length of 19 tags. From the figure we can see that shorter sequence lengths are more common than longer ones, 74.46% of the sequences has a length of six or less. The average sequence length in this data set is 5.3, with a median of length four.

When we zoom in on the sequences, we can see that of the 12 sequences with a length of seven or longer, 75% of these sequences has at least one time a `refinement` tag within the sequence. This could be an explanation as to why the sequence is longer, as the `refinement` of a requirement creates new specifications and therefore could start evolving again as well, adding to the sequence length.

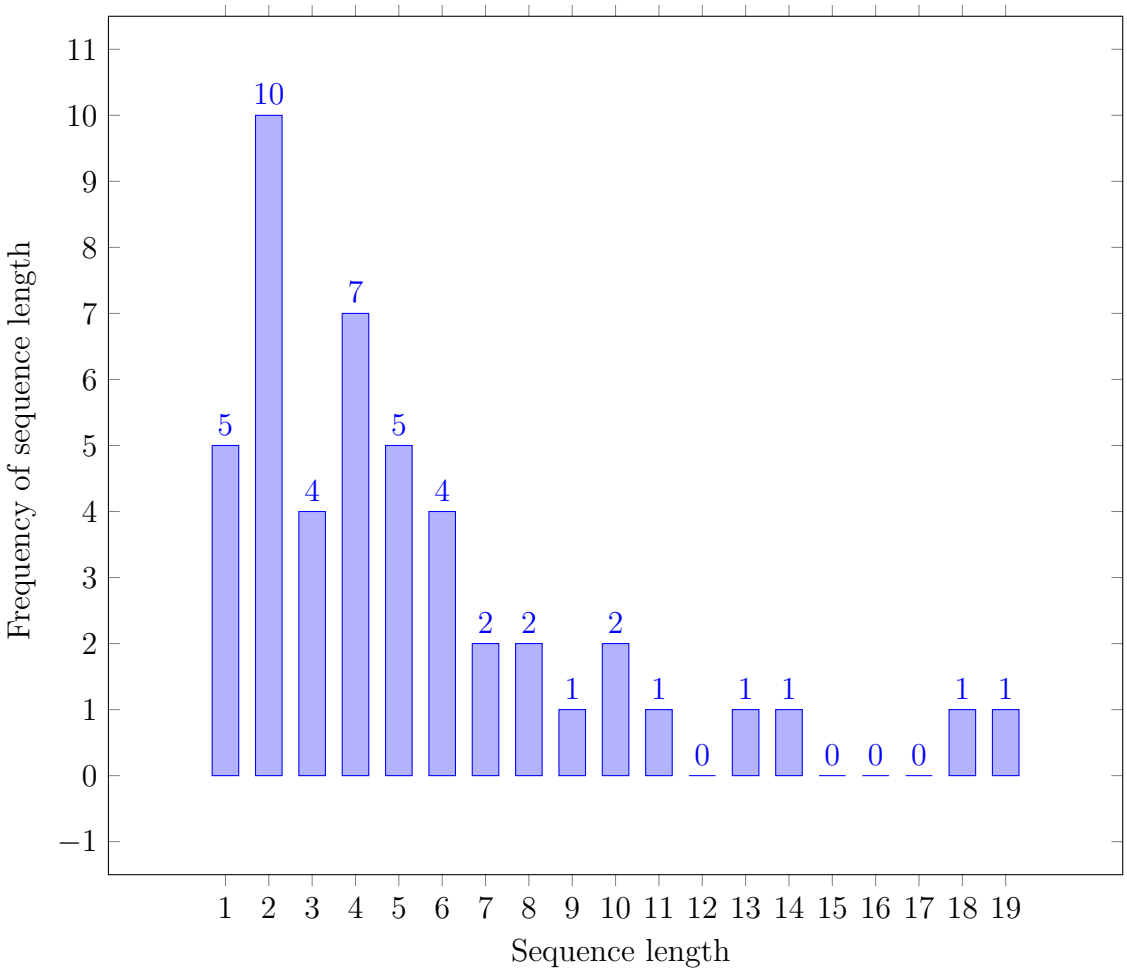

Figure 5.3: Frequencies of sequence lengths

5.2 Evolution Tags Occurrence

One of the first noticeable results is the amount of occurrences of the evolution tags. In Table 5.2, the first column shows the evolution tag discussed, the second column is the total number of occurrences where this tag could be identified. The last two columns are the number of sequences that were tagged at least once with that tag, and the percentage for that as well.

When looking at the percentages, we see five tags that appear in sequences far more than the other tags. The 100% of the `add` tag is to be expected, as we defined a sequence as one that starts with an `add`, so every sequence will contain at least one tag of this kind. The other noticeable tags are `refinement`, `(indirect) traceability` and `prescriptive explanation`. All of these tags are identified in around half of the sequences, and with the other tags only being identified in less than 15% of the sequences, there is quite a gap here. For `(indirect) traceability`, this could be explained by the working method of

Evolution	Total number of occurrences	Sequences containing tag at least once	Sequences (%)
Add	47	47	100.00%
Delete	2	2	4.26%
Refinement	32	20	42.55%
Modify	5	4	8.51%
Rewording	6	4	8.51%
Traceability	64	23	48.94%
Indirect Traceability	50	26	55.32%
Prescriptive Explanation	34	25	53.19%
Descriptive Explanation	8	7	14.89%

Table 5.2: Amounts of occurrences of evolution tags

the company of the case study, as it could be prescribed to link to other elements. For some more information, it is necessary to dive a bit deeper into the specifics of the `(indirect) traceability` tags, and what they trace to.

5.2.1 (Indirect) Traceability

`Traceability` is used to tag all the clickable direct trace links between the requirement and any other requirement relevant information, a more extensive explanation can be found in Section 4.3.2. For the 64 `traceability` tags, the distribution can be found in Figure 5.4, on the left-hand side. In this figure, we show how much of the `traceability` tags come from tracing to other issues within the project management system, to the system that is being developed or to a process. An example of a process is when there is a link to a system the company makes models in to show a process overview. In Figure 5.4, we see that the greater majority of the `traceability` tags come from trace links to other issues within the project management system. The reason for this being a greater majority could be because of the nature of the project management system, Jira. Here, relationships can be added between *issues* from a predefined list of relationship types, where the relationship shows up for both *issues*. So, the `traceability` tag can be identified at both *issues* as well. A list with these predefined relations can be found below. However, these relationships as provided by Jira are not the only way to link *issues*, one could also simply add a hyperlink to another *issue*.

- Clones / is cloned by
- Duplicates / is duplicated by
- Relates to
- Blocks / is blocked by
- Parent / child

Direct links to the system that is being build were added in 31.25% of the `traceability` tags, where it often occurred together with comments telling that this link lead to where the implementation of the requirement could be found. Only a small percentage (3.12%) of the `traceability` tags originate from links to other artefacts used to support processes. These were mostly process flows that had been made visual in another system.

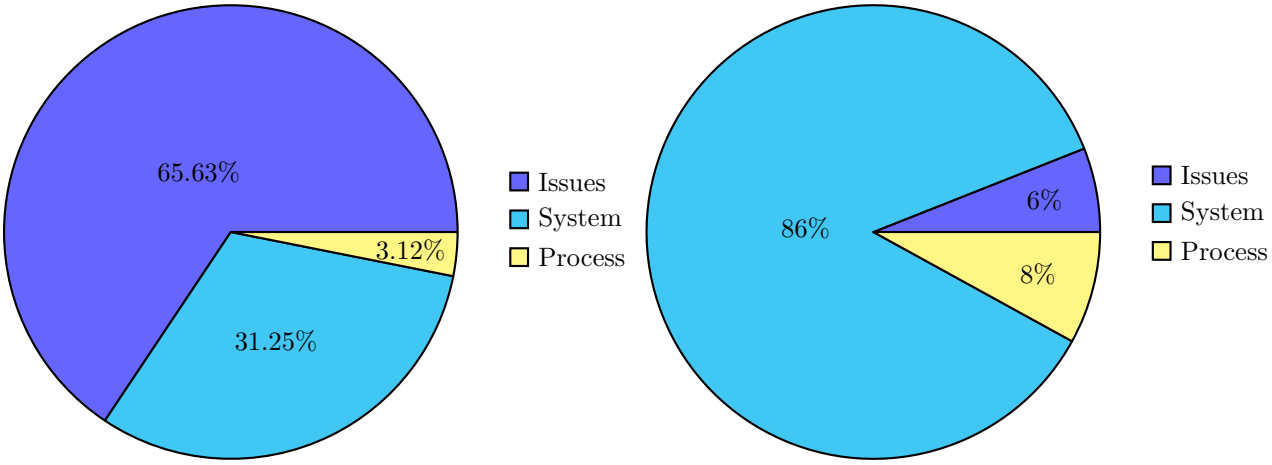

Figure 5.4: Distribution of the traceability tags (left-hand side) and the indirect traceability tags (right-hand side)

`Indirect Traceability` has a different distribution. This tag is used for all indirect trace links, meaning links that are not clickable, but do give stakeholders enough context to navigate to the linked item themselves. Most of the data entries that could be tagged as `indirect traceability` were entries where images to other artefacts were added. However, in some cases there was also a textual trace link, where the ID of another issue was mentioned, instead of using a hyperlink. This way, there was enough information for stakeholders within the project to find the issue. More information about this tag specifically can be found in Section 4.3.2. The distribution of the `indirect traceability` tag is visualized in Figure 5.4 on the right-hand side. Most of these tags, 86% of the 50 tags, originate from images of the system, mostly to display where, or confirm that, the requirement is implemented. The other two groups are way smaller, only four of the 50 `indirect traceability` tags were concerned with the processes or the code, where images of those were used as a reference. Lastly, the other issues were only linked in 6% of the `indirect traceability` tags. These were the ones that did not have an actual clickable link to another issue, but only referred to it in text by naming its ID. This could be because most of the references to other issues were actual clickable links, thus tagged as `traceability`.

With these distributions, we can see that most tags from the *linking* category originate from either relations and links with other issues, or links to the implemented part of the

requirement in the system. In the results section further in this thesis we will discuss these outcomes with the participants of the case study to see whether this is related to the working method of the company.

5.2.2 Prescriptive Explanation

`Prescriptive explanations` are tagged at least once in 53.19% of the sequences. These tags are used when there are the explanations to clarify the requirement by adding conditions that the requirement should comply to (Section 4.3.2). Of these explanations, 73.53% took place in the description of issues. Additionally, many of these tags are added right after a main requirement text, 58.82% of these tags are identified right after a `add` or `refinement` tag. This indicates that most `prescriptive explanations` are added in early stages of a requirement life-cycle. This might be explainable by the nature of the `prescriptive explanation`, which adds conditions to the requirement. Adding these conditions once development has already been started is less desirable as new conditions might mean having to change the implementation.

5.2.3 Refinement

The `refinement` tags are identified in 42.55% of the sequences. This tag is used, as described in Section 4.3.2, when there is an extension to an existing requirement. The `refinement` can be in the form of a new issue, but can also happen within the *issue* of an original requirement. This is quite evenly distributed; 56.25% of the `refinement` tags are refinements of a requirement in a different issue than the original requirement. In 43.75% of the `refinements` tags, we have the extension and the original requirement within the same issue. Of these `refinement` tags within the same issue as the original requirement, the vast majority (78.57%) is situated in the *comments* of an issue, and only 21.43% can be found in *descriptions*. This shows that *comments* do not only have a supporting role for the discussion about requirements or tracing them, it also can introduce extensions of requirements.

5.2.4 Descriptive Explanation

The `descriptive explanation` is the first tag that we discuss that has been tagged in far fewer sequences than the tags discussed above. This tag is used in cases when there is a clarification added to a requirement without conditions the requirement should comply to (Section 4.3.2). This tag has only been identified in 14.89% of the sequences, which is less than half of the sequences fewer than the other tag in the clarification category, the `prescriptive explanation`. Of all the tags that we tagged from this category, clarification, we see in Figure 5.2, in the lower left pie chart that 80.95% of those tags is tagged as `prescriptive`, and only 19.04% of `descriptive`. From this we can see, if clarifications are added, most of them hold extra conditions the requirement should comply to.

5.2.5 Modify and Rewording

Both `modify` and `rewording` fall within the same category, the irreversible changes. Both of these tags are used in the same amount of sequences, only four, which is 8.51% of the sequences. Both `modify` and `rewording` are irreversible changes, which means that they change the requirement in a way that the old version is practically impossible to retrieve, unless one exports the database, just as we did for this analysis. `Modify` is used when the the meaning of the requirement is altered, and `rewording` is used when the meaning stays the same, only the textual formulation is changed. Section 4.3.2 explains these differences more in depth. Both of these tags are only identified in 8.51% of the sequences, and have not occurred within the same sequence. For the `rewording` tag, five out of six of these tags are identified right after the `add` tag or one after that. For the `modify` tag this is the case for four out of five tags, where the fifth tag is right after a `refinement`. We can see that when these tags are used, they are often identified early within the requirement life-cycle. The whole sequences can be found in Appendix B.1. We believe that this is desirable, especially for the `modify` tag, as this tag is used when the meaning of a requirement is altered, and when this happens later in the requirement life-cycle chances are greater that the previous work done for the requirement has to be altered as well. This then results in more work than when the changes were taken into account right from the start.

5.2.6 Delete

The remaining tag (`delete`) has a remarkably low number of occurrences. This tag is only identified in two sequences (4.17%). `Delete` is the only certainty for the end of an evolution, as it means that that sequence comes to an end, with no way of continuing with the sequence at a later point in time (Section 4.3.2). The reason for the low amount of `delete` tags could be linked to the working method, as deleting issues might not be widely accepted for traceability purposes. In the results chapter of this thesis, this hypothesis will be checked with case study participants. As for the data, the only occurrences of `delete` are when a requirement was first described in the initial requirement specification document, outside from the project management system, but then never entered in the project management system. With this data set we could not identify a `delete` tag for any requirement that has been entered into a project management system.

5.3 Combinations of Tags

The occurrences of tags within sequences give an interesting first overview, however, the goal of this thesis is to see if there are patterns that we can identify. Looking at combinations of tags seen together is needed to answer SQ2 (2.1) as well. Therefore, in this section we look at the combinations of tags that can be seen together in a sequence. First, we look at the frequency of occurrence of couples of tags, irrespective of their position. Then we also take the position into account with respect to each other, so if they are present in the same

sequence before, or after the other tag. Lastly, we use Association Rule Mining to calculate likelihoods of different tags occurring together.

5.3.1 Co-occurrence

We first start by analysing all combinations of two tags and calculate how often those are present in the same sequence. These results are presented in Table 5.3, here all cells with a percentage of 50 or higher are highlighted green as they stand out as it could indicate a strong correlation. This correlation will be statistically looked into in Section 5.3.3 by means of association rule mining. The table can be read as follows: the number of sequences that contained both the row value as the column value, divided by the number of sequences that contained at least the column value. For example, the column `traceability` and the row `modify` can be read as: the number of sequences that contained both at least one `traceability` tag, and at least one `modify` tag, related by the number of sequences that contains at least one `traceability` tag, is 8.70%. In other words, of all sequences with at least one `traceability` tag, 8.70% also contains at least one `modify` tag.

For the diagonal, where the row and column titles are the same tag, the percentage of sequences with that tag in it at least twice is noted.

	Add	Delete	Modify	Rewording	Traceability	Indirect Traceability	Prescriptive Explanation	Descriptive Explanation	Refinement
Add	0.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Delete	4.26%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Modify	8.51%	0.00%	25.00%	0.00%	8.70%	3.85%	12.00%	42.86%	15.00%
Rewording	8.51%	0.00%	0.00%	50.00%	13.04%	11.54%	8.00%	0.00%	10.00%
Traceability	48.94%	0.00%	50.00%	75.00%	65.22%	69.23%	64.00%	71.43%	70.00%
Indirect Traceability	55.32%	0.00%	25.00%	75.00%	78.26%	57.69%	64.00%	57.14%	60.00%
Prescriptive Explanation	53.19%	0.00%	75.00%	50.00%	69.57%	61.54%	20.00%	71.43%	70.00%
Descriptive Explanation	14.89%	0.00%	75.00%	0.00%	21.74%	23.08%	20.00%	14.29%	20.00%
Refinement	42.55%	0.00%	75.00%	50.00%	60.87%	46.15%	56.00%	57.14%	30.00%

Table 5.3: Combinations of evolution tags, how often the row value is identified in the same sequence as the column value, for all the sequences where the column value is present

In the previous section, we described the amounts of occurrences of tags, here we see that there are a couple of tags that occurred way more often in a sequence than the others. These are the `add, refinement, (indirect) traceability` and `prescriptive explanation` tags. This is something to keep in mind when looking at Table 5.3, specifically when looking at the rows of these tags, as these are more likely to have higher percentages. This comes from the fact that for example an `indirect traceability` tag is present in more than half of the sequences, so chances of it being present in the hand full of sequences containing a `descriptive explanation` is higher as well.

The first result, which is useful for a validation of the tagging, is of the `add` tag. Every value in the `add` row has a result of 100%, except for *Add* × *Add*. Every sequence starts with an `add` tag, so every sequence with a `delete` will also have an `add`. This is the same for every other tag as well, except for the diagonal, so where `add` is present at least twice. This result is 0%, as expected, as an `add` implies a new sequence, so there will never be two of this tag in the same sequence.

When looking at the diagonal, we see how often a tag is identified at least twice in the same sequence. The `rewording` and `(indirect) traceability` tags have a percentage of 50 or higher, which means that in at least half of the sequences where such a tag was identified, at least one other tag of the same type is identified.

The percentage for *Descriptive Explanation* × *Modify*, is noteworthy. Here we see that in 75% of the sequences that contain a modify, there is also a descriptive explanation. In Section 5.3.3 we look at this result with support levels and confidence rules to see if this is a compelling correlation.

5.3.2 Co-occurrence Considering the Sequencing

After looking at the co-occurrence of tags in the same sequence, it is interesting to also look at this co-occurrence with a specific ordering of the co-occurring tags. Table 5.4 shows the percentages of tags occurring before each other. It shows the percentage that the tag in the row is in the same sequence and positioned before the tag in the column, related to all the sequences containing the tag in the column. For example, out of all the sequences containing at least one `traceability` tag, 60.87% also contains a `prescriptive explanation` tag, which is positioned *before* the `traceability` tag.

	Add	Delete	Modify	Rewording	Traceability	Indirect Traceability	Prescriptive Explanation	Descriptive Explanation	Refinement
Add	0.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Delete	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Modify	0.00%	0.00%	25.00%	0.00%	8.70%	3.85%	12.00%	14.29%	10.00%
Rewording	0.00%	0.00%	0.00%	50.00%	13.04%	11.54%	8.00%	0.00%	10.00%
Traceability	0.00%	0.00%	0.00%	25.00%	65.22%	57.69%	36.00%	14.29%	65.00%
Indirect Traceability	0.00%	0.00%	0.00%	0.00%	39.13%	57.69%	24.00%	14.29%	45.00%
Prescriptive Explanation	0.00%	0.00%	25.00%	25.00%	60.87%	50.00%	20.00%	14.29%	60.00%
Descriptive Explanation	0.00%	0.00%	50.00%	0.00%	21.74%	15.38%	20.00%	14.29%	20.00%
Refinement	0.00%	0.00%	25.00%	25.00%	39.13%	34.62%	24.00%	28.57%	30.00%

Table 5.4: Combinations of evolution tags where the percentage is how often the row value appeared before the column value in all sequences where the column value is present

The first row and the first column, for the `add` tag can be explained, just as with Table 5.3, by the nature of the `add` tag. This tag is the first tag in any sequence, so for the row it is as

expected that in 100% of the sequences with the other tags, the `add` tag will be present before it. As for the column, no other tag will ever be present before an `add` tag in a sequence, therefore all those percentages are zero.

The diagonal is the same as in Table 5.3, as position does not have an impact when working with two tags that are the same.

It is interesting to look at the most identified tags, that occur together often, and to see in what order those tags have in a sequence. These tags that often occur in the same sequence are `(indirect) traceability, prescriptive explanation` and `refinement`. In Figure 5.5 we have six pie charts for all the combinations of these tags. In the legend, we use the first letters of the tag to keep it shorter. The red pie chart is about the combination of the tags `traceability` and `indirect traceability`. Here we see that in 33.33% of the sequences where both tags are present, the tags alternate, which means that both `traceability` comes before `indirect traceability` as the other way around within the same sequence. The majority of sequences that has both of these tags in them, have the tag `traceability` before `indirect traceability`.

The orange pie chart in Figure 5.5 is for the tag combination of `traceability` and `prescriptive explanation`. Here we see that a greater majority of the sequences either has both tags alternating, or has the `prescriptive explanation` before the `traceability`. This could be a result of the observation that `prescriptive explanations` often are added in early stages of the requirement life-cycle, whereas `traceability` is often added later in the life-cycle.

The yellow pie chart displays the `traceability` and `refinement` combination. In this case, the majority favors the alternation of the tags, which could be explained by the fact that `refinements` are extensions of an original requirement. These original requirements have their own `traceability` tags when they are being worked on, and once a `refinement` is added, some more `traceability` links are added to trace the extension that is being implemented.

The lighter blue pie chart shows the distribution for the `indirect traceability` and `prescriptive explanation` tags. Here the majority comes from `prescriptive explanation` before `indirect traceability`. This could be explained by the distribution of the `indirect traceability` tags as displayed on the right-hand side of Figure 5.4. Here we showed that 86% of the `indirect traceability` tags come forth out of images of the system, mostly to show where or confirm that the requirement is implemented. This together with the fact that `prescriptive explanations` are mostly added in the early stages of a requirement life-cycle, make the order of first finding a `prescriptive explanation`, and later in the sequence the `indirect traceability`, as expected.

The bottom left pie chart in Figure 5.5 shows the relation of `indirect traceability`

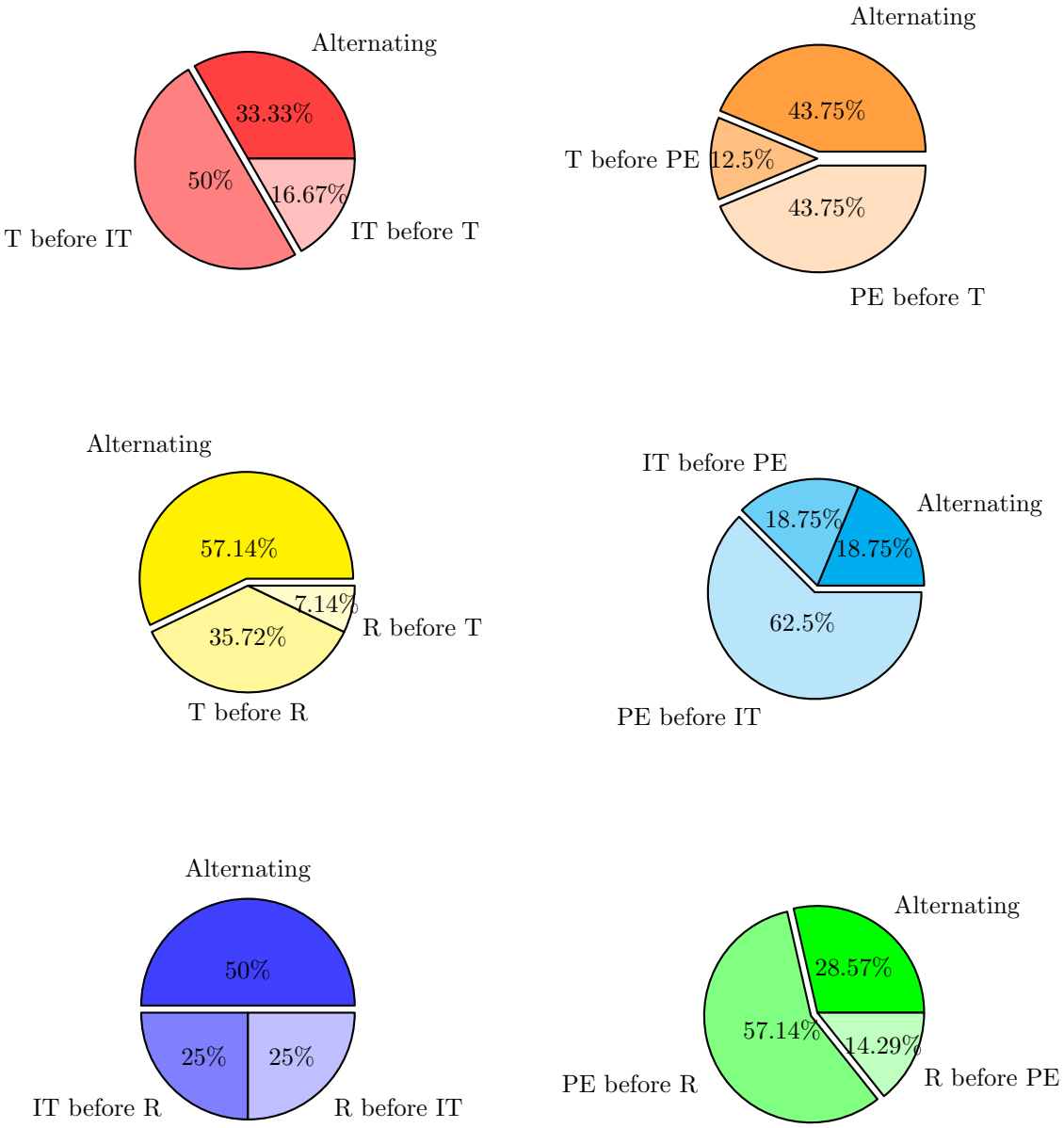

Figure 5.5: Most occurring tags and their position towards each other

and `refinement`. This relation is quite evenly distributed. With a majority of 50% for both tags alternating, and the other options uniformly split into two equal parts of 25%.

The last pie chart displays the combination of the tags `prescriptive explanation` and `refinement`. For this combination, the majority lays with the `prescriptive explanation` before the `refinement`. Here the `prescriptive explanation` being present in the early phases of a requirement life-cycle comes to light again, and could be an explanation as to why this order is seen most for this combination of tags.

5.3.3 Association Rule Mining

To make the above calculations and observations more rigorous, we worked with the Apriori Algorithm to conduct Association Rule Mining (ARM). ARM is a research field on its own, but we will discuss some basics to explain how we used it. ARM is a data mining technique to identify relations between items, and the Apriori algorithm is a popular approach to identify frequent itemsets and generate association rules based on the frequency of items their co-occurrence [67]. Here, we do not take into account the position of items towards each other. We only look at the presence of what items can indicate the presence of another item in the same set. With ARM, one works with a minimum support and a minimum confidence level. The minimum support level is a threshold value used to determine which itemsets are considered frequent and which are not [68]. For example, when the minimum support level is set to 0.4, an itemset is considered as frequent if it appears in at least 40% of the transactions. The minimum confidence level is used to determine which rules can be considered interesting or not, it stands for the probability of the consequent being present, given the antecedent [68].

Originally these are values determined by the user, however, in 2021, Hikmawati *et al.* [69] came with a critique on the user-determined minimum support levels. They note the importance of finding the right minimum support level to generate useful results, and that finding this value being purely dependent on the intuitiveness of the user, often leads to problems in the results [69]. Hikmawati *et al.* [69] thus presented a method for generating the minimum support value by incorporating additional criteria, such as utility, in addition to considering the frequency of occurrence alone. With this proposed formula by Hikmawati *et al.* [69], we calculated a minimum support level of 0.02. For the confidence level, we worked with 75%, this generated 387 rules, however, one should note that the specific properties of the `add` tag are not taken into account. However, as we know, the `add` is present in every itemset, therefore all rules are added three times, once with the tags the rule is actually useful for, once with the tag `add` added to the antecedent, and once with the tag `add` added to the consequent. Besides that, these rules also contain all possible combinations with only an `add` as either the antecedent or consequent, even though these rules are not useful as having an `add` in every sequence is a given. Filtering these values out of the results, we are left with 105 rules, of these 105 rules, 77 have a confidence of 100%. This percentage mostly comes from rules where either or both the antecedent and consequent are quite large, which makes those rules more specific. When removing these rules from the results, we end up with 28 rules. Lastly, we sort by the lift value. This value measures how more often the antecedent and consequent appear together than we would expect [70]. For example, a lift value of 4 means that when the antecedent appears in an itemset, it is four times more likely to also have the consequent appear in that same itemset, than when the antecedent would not be there. Most of the 28 results have a lift value between one and two, which is a positive correlation, but a rather low one. In Table 5.5, the first quarter of the rules sorted by lift value, are displayed with their support, confidence and lift values.

In this table, we see that with a confidence of 75% we can say that when a `modify` tag

Antecedent	Consequent	Support	Confidence	Lift
Modify	Descriptive Explanation	0.064	75%	5.036
Modify	Refinement, Prescriptive Explanation	0.064	75%	2.518
Descriptive Explanation, Refinement	Traceability, Prescriptive Explanation	0.064	75%	2.203
Indirect Traceability, Descriptive Explanation	Traceability, Prescriptive Explanation	0.064	75%	2.203
Rewording	Indirect Traceability, Traceability	0.063	75%	1.958
Descriptive Explanation, Prescriptive Explanation	Refinement	0.085	80%	1.88
Indirect Traceability, Refinement, Prescriptive Explanation	Traceability	0.17	88.89%	1.816

Table 5.5: Results of ARM with minimum support value 0.02, confidence 75% or higher, but not 100% and sorted by highest lift values

is present in a sequence, it is 5.036 times more likely that there will also be a `descriptive explanation` tag in the same sequence, than when there would not be a `modify` tag. This corresponds with our earlier noted results of the `modify` and `descriptive explanation` tags in Section 5.3.1. The last rule in the table is also noteworthy. This rule has the lowest lift value in the table, but the highest confidence and support values. This rule states that when in a sequence the following tags are present: `indirect traceability, refinement` and `prescriptive explanation`, then with a 88.89% confidence, it is 1.816 times more likely to also have the `traceability` tag present in that same sequence, than when those antecedents would not be present. As stated before, ARM does not take an order of the items into account, it only looks at the occurrence in an itemset and the likelihood of the presence of items given there are other items present.

5.4 Timing of Tags

When researching the impact of timing on the evolution of a requirement, we can talk about two different meanings of timing. The first is the timing of requirements evolution tags related to the actual duration of the project as a whole. For example, 2 weeks versus 2 months after the start of the project, and the project was followed from July 2022 up until March 2023. The second is the timing of requirements evolution tags related to the phases a requirement can live in, so related to the requirement life-cycle. The company of the case study works with a set of statuses that demonstrate the workflow. These statuses are displayed in Figure 5.6. Below we have a list with the statuses that occur in our data set,

and a short description for each status. The grey statuses in the figure mean that they are pending or in a status where rework on the requirement is needed. The blue statuses in the figure are when a requirement is in progress, either someone started building or it is being tested. The green statuses in the figure are the finalized statuses a requirement can have in this working method.

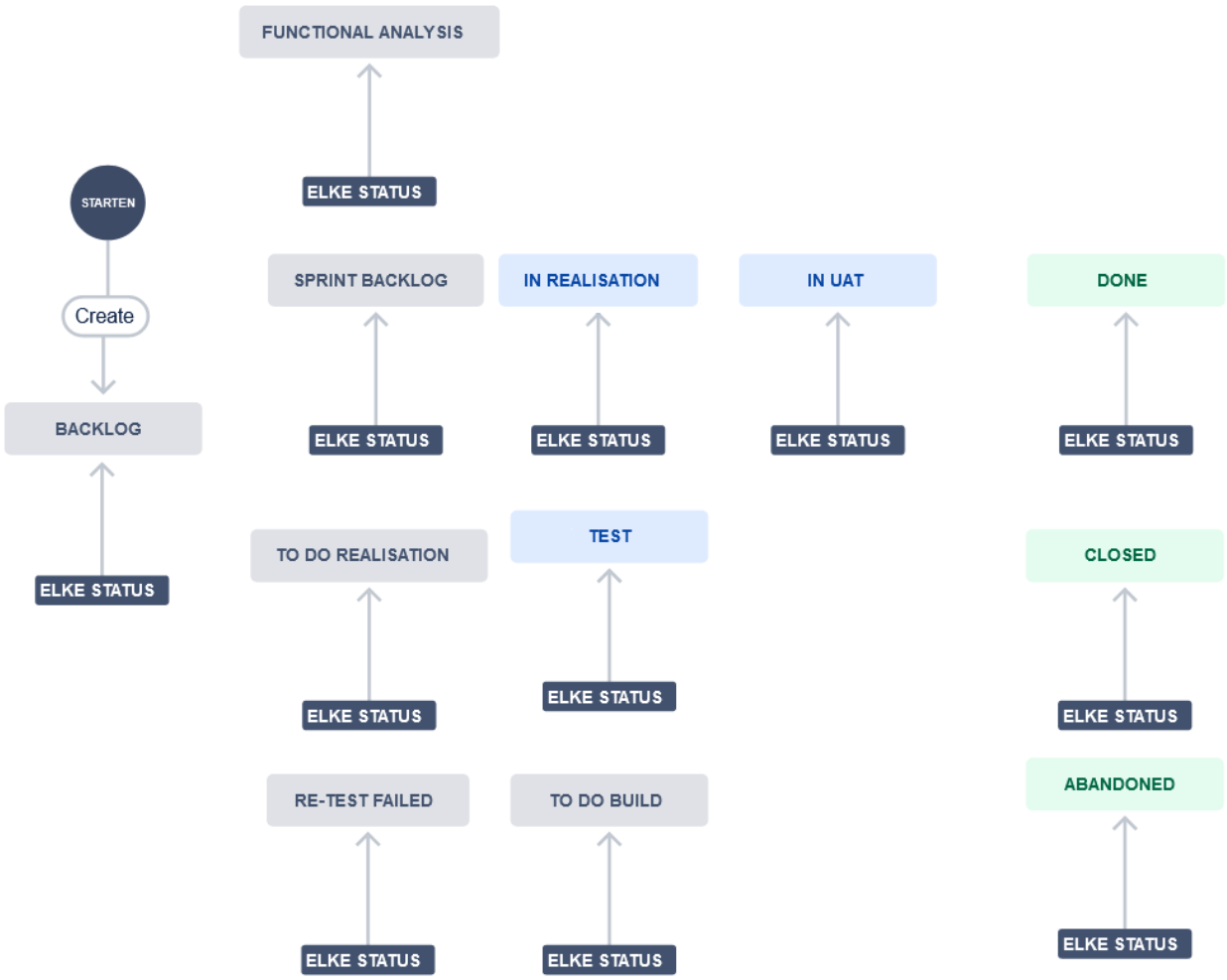

Figure 5.6: Requirements statuses and their workflow according to the project management system

- *(Sprint) Backlog*
  This is the status for requirements that still or again are in the backlog.
- *Functional Analysis*
  The status for requirements for which a functional analysis is still required.

- *In Realisation*
  This requirement status is used when an requirement is being implemented.

- *Test*
  This status is used when the requirement is implemented and that implementation is being tested by internal testers, or in this specific case also often the product owner.

- *In UAT*
  UAT stands for user acceptance test. This is usually the status after the test status, here it is not tested by specific testers, but it is tested by the actual end-users of the system. In this case these were the key stakeholders.

- *RE-Test Failed*
  This status is when a test is failed, either because the functionality is not working, or because it is not what the end-users had in mind. In that case, often the developers have to iterate back over their implementation again.

For this section we first analyze the timing related to the actual duration, and after that we analyse the timing looked at the statuses.

5.4.1 Timing in the Duration of the Project

In a software project, it is expected that most of the requirements are added in early stages of the project. This is because the first backlog has to be filled and every feature is mapped in a requirement, later in the project there can be additions of extra features. Figure 5.7 shows how many of the requirements that are added within the project, are added within which month. The project started mid July 2022, the majority of the total requirements added in the project were directly added in July, with 25 requirements which is 53.19% of the total requirements. When looking at the last half of July and the first full month, August, we see that in total 37, which is 78.72%, requirements have been added. As explained before, the majority of the requirements being added is as expected in the beginning of the project. However, in September, December and March, a couple of requirements are added as well.

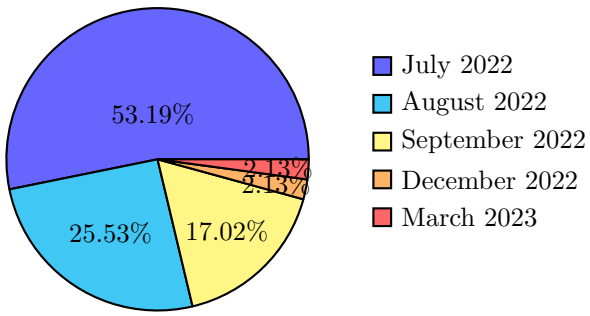

Figure 5.7: Distribution of how many requirements are added when in the project

Figure 5.8 shows the distribution of the frequency of a requirement evolution length, depending on the month the requirement was added. As more than half of the requirements were added in the month of July, the chances are higher that greater sequence lengths are also from requirements added in that month. There is no clear distinction that can be seen from this figure about differences in sequence lengths for requirements added early or later in the project.

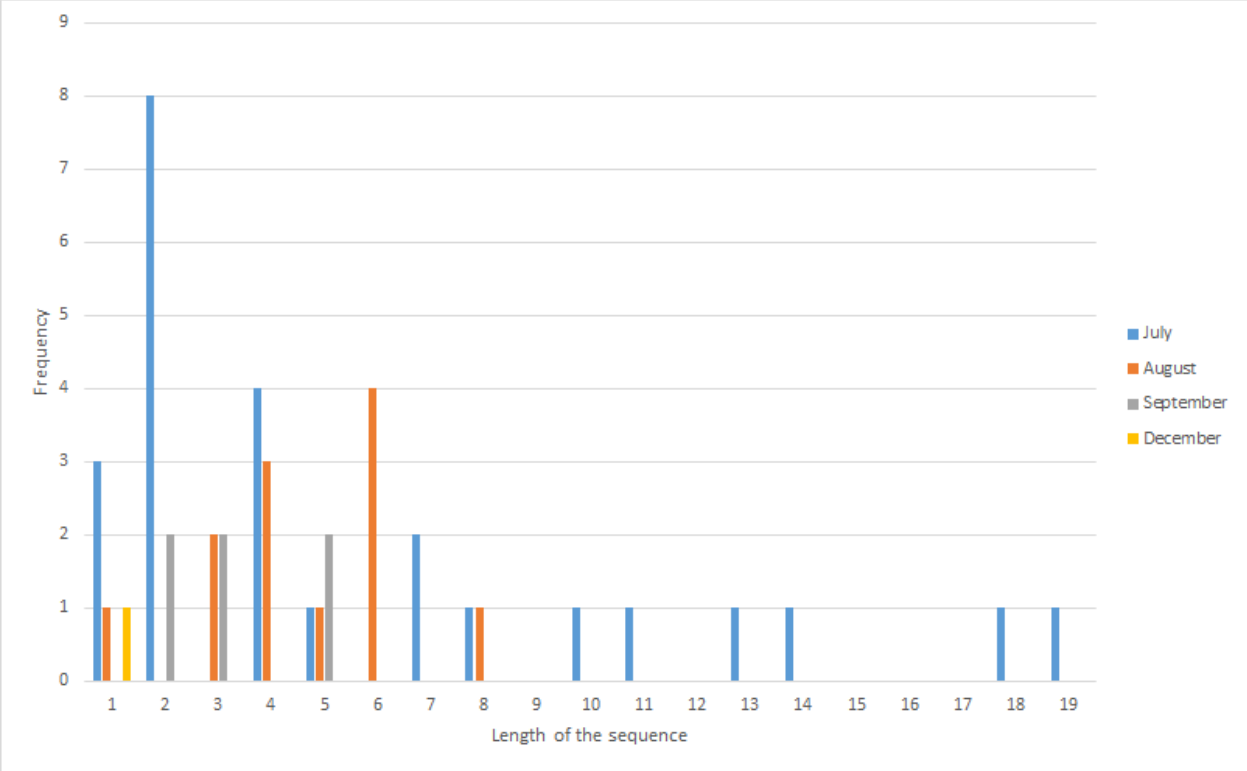

Figure 5.8: Bar chart of the frequency of a sequence length, per month the requirement is added

Additionally, as discussed earlier, `refinement` is a form of adding a requirement (Section 4.3.2), however, this is is an extension of an already existing requirement. Therefore, the expectation is that this kind of addition is more likely to be added later in the project, as there has to be an original requirement which is extended. Figure 5.9 shows the distribution of the 32 tags of `refinement`, and when they were added in the project. We indeed see that for this tag the first one and a half months only provide for 28.13% of the refinement tags. The largest part of these tags have been added in the months September and October. This could be related to most stakeholders being back in office after holidays, working on this project again.

Finally, when looking at the dates of the `add` and `refinement` tags and comparing them

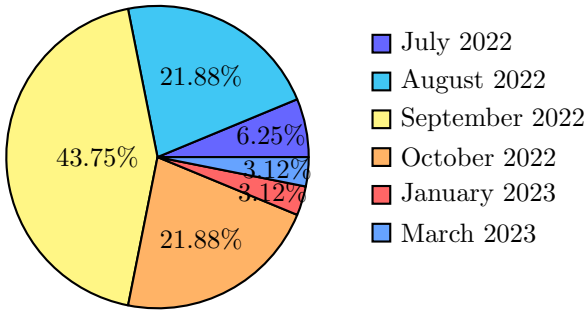

Figure 5.9: Distribution of how many refinements are added when in the project

to the dates of the meetings between the stakeholders about the project, we see that a small majority (53.16%) of the `add` and `refinement` tags are from the same, or within two working days after such meeting. 36 `Adds` and `refinements` are tagged on dates that are not directly linked to the meetings. Notably, of those 36, 77.78% are added with at least one other `add` or `refinement` tag on the same date. This could indicate that there was another trigger besides the meetings which sparked ideas for additional requirements.

5.4.2 Timing of the Status of a Requirement

Besides looking at the absolute timing of requirements in the project, we also look at the timing related to the statuses that are being used in the project. These statuses have been discussed earlier, and they are the statuses that stakeholders in the project can label to a requirement in the project management system. For the two requirements that were never added to Jira, the start status of *backlog* has been used. The stakeholders indicate with this status what the next expected, or current, steps for the requirement are. Figure 5.10 shows a scatter plot of the frequency of evolution tags occurring in the different statuses. In this figure, we see that the *backlog* status has a high frequency of almost every evolution tag. One noticeable outcome here, is that most of the `traceability` and `indirect traceability` tags are added when an issue had the *in realisation* status. This could be related to the earlier discussed working method of the company, where developers might be expected to add traces to the system when they implement a requirement. This hypothesis will be discussed with the stakeholders in the results chapter of this thesis.

The group of statuses that are connected with a form of testing, so *in UAT* (user acceptance test), *RE-test failed* and *test*, all have a high frequency of the `(indirect) traceability`, where we see a connection with stakeholders clarifying what they tested, what worked and what did not work, by adding traces to the other artefacts.

For `refinements`, we see that a lot are added in the *backlog* status, which is explainable as a majority of the `refinements` are added in new issues in the project management sys-

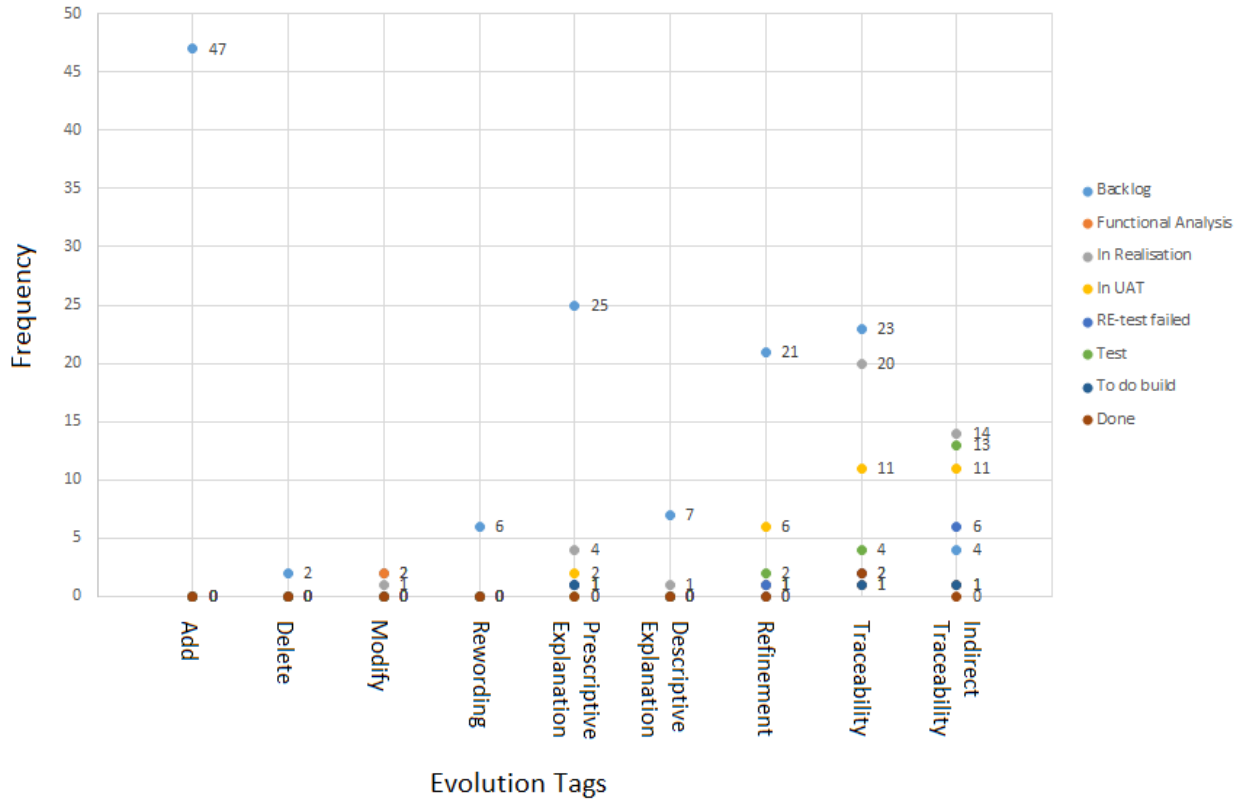

Figure 5.10: Scatter plot showing the frequency of certain tags occurring in different statuses

tem, and when new issues are added, they almost always start in the *backlog* status. The next greatest part of the `refinement` tags are added in the *in UAT* status. An explanation for this could be that a requirement was in the user acceptance testing phase, and that requirement was accepted, but the user did think of an extension of that requirement, thus added a `refinement`. The source analysis at the end of this chapter could give some more context for this explanation.

The last noticeable result from this analysis is the relative high frequency of the `modify` tag in the *functional analysis* status. This could be a result from unclear requirements being identified in the *functional analysis* phase, where then the requirement is changed, therefore modified.

5.5 Source Analysis

In this section we investigate the impact of the source of a requirement has on the evolution of that requirement. We analyse where the origin of a requirement comes from, so whether the addition of a requirement is based on a conversation in a formal meeting, if it could be

led back to comments made in the project management system or if we are not able to locate the origin of the requirement. We look at the impact these origins have on the requirement and its evolution.

So the three different source options we have are as follows:

- Requirements Conversations
- Project Management System
- Unknown

The *requirements conversations* are the formal meetings that are recorded throughout the case study. The *project management system* is only focused on `add` tags that have been added based on another item in the project management system that holds the reasoning. Lastly, we have the *unknown* group, this group could hold requirements that have been discussed by stakeholders, just not in places we have access to, or requirements whose origin have not been discussed but were a thought process of a stakeholder that we do not have access to. Simply said, sources that are not documented, or that we do not have access to.

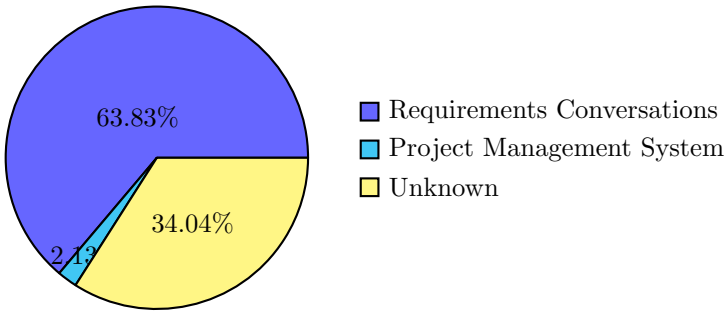

Figure 5.11: Distribution of sources for `add` tags

Figure 5.11 shows the distribution of the sources of the added requirements. Of the 47 `add` tags, 63.83% could be led back to a *requirements conversation* where the need for the requirement was discussed. This has to be in quite literal form, if the requirement has not been discussed with the same terms in the meeting as it is written down, then the source will not be tagged as such meeting. Only for one `add` the source can be found in the *project management system*. Here a stakeholder expressed their thought process and reasoning for the necessity of that requirement through a comment. Lastly, we have the *unknown* group, which holds 34.04% of the `add` tags. For 31.25% of all the unknown sources, an indirect relation could be identified with other requirements that originated from a conversation. For example, one conversation had a discussion about displaying specific project details, and what those details were. This discussion led to a set of requirements focusing on specific details that could be directly tracked back to the conversation. However, there are also a few requirements concerning a broader overview. Although they were likely conceived during the same discussion, they were not explicitly mentioned in the conversations. As a result, these

requirements are linked as *unknown*.

On the left-hand side of Figure 5.12, we visualize how many of all of the tags could be tracked back to the *requirements conversations*, to the *project management system* and how many could not be tracked back to one of those sources, which were then labeled as *unknown*. Where for the `add` tags, the majority could be tracked back to *requirements conversations*, when we look at the total of all tags, the majority has an unknown source. We can zoom in on those *unknown* sources a bit more, which is visualized on the right-hand side of Figure 5.12. The tags that fall in this category of *unknown* sources, half of them can be labeled as *original*. This label is used for tags that cannot be tracked back to a source such as a different item in the *project management system* or to a *requirements conversation*, but where that is expected, as the data entry that is tagged reads as original content. For example, these original unknown sources are mostly used for entries that are tagged as `traceability` or `indirect traceability`. This is because these links often originate from developers linking what is implemented, or testers confirming that the requirement is indeed implemented and working. As these are the kind of data entries that are not expected to be discussed outside of that developer or tester their work, we labeled them as *original*. Summarizing, *original* sources are used for data entries that cannot be linked to a different source, but where this is as expected.

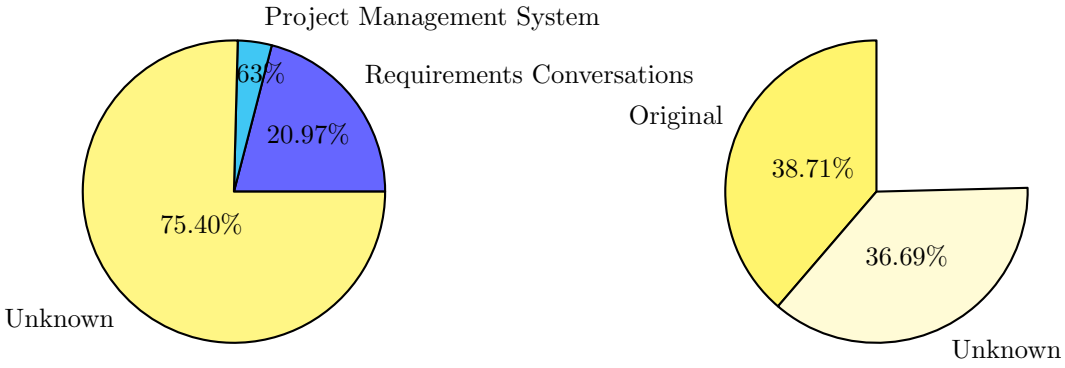

Figure 5.12: Distribution source location of all tags (left-hand side) and unknown source detailed (right-hand side)

5.6 Initiator Analysis

In this section, we are looking at the initiator of the requirement or change. We look at the impact of the role of the person initiating a requirement, on the evolution of that requirement. As discussed before, in this project there was one requirements analyst, one product owner who was also supporting the requirements analyst, two testes, three developers and three key stakeholders, whom all are project managers and the main end users of the system. For this part of the analysis we are looking at the different roles and their possible impact

on the requirement and its evolution.

Here, we have multiple options for who the initiator of the tagged data entry is. First we can look at the initiator as the person realizing the change, this is the person whose name is added to the data entry in the project management system. However, we can also dive a bit deeper for all the requirements that have a known source location, and look for the person that actually first suggested the change. First we look at how many requirements that have the `add` tag are initiated by what role. For this analysis we look at the initiator as the one first suggesting the requirement, when the source location of the requirement is unknown, the initiator in this distribution is also labeled as unknown. For the `add` tag we work with the person suggesting it first, as looking at the person that actually added it to the project management system will be less interesting. It is the job of the *requirements analyst*, especially in early stages of the project, to add requirements in the project management system. When looking at those values, 87.23% of the `add` tags are from data entries entered by the requirements analyst. When diving into the requirements conversations and tracing back to when the need for a requirement was first mentioned and by whom, we get the distribution as seen in the left-hand side of Figure 5.13.

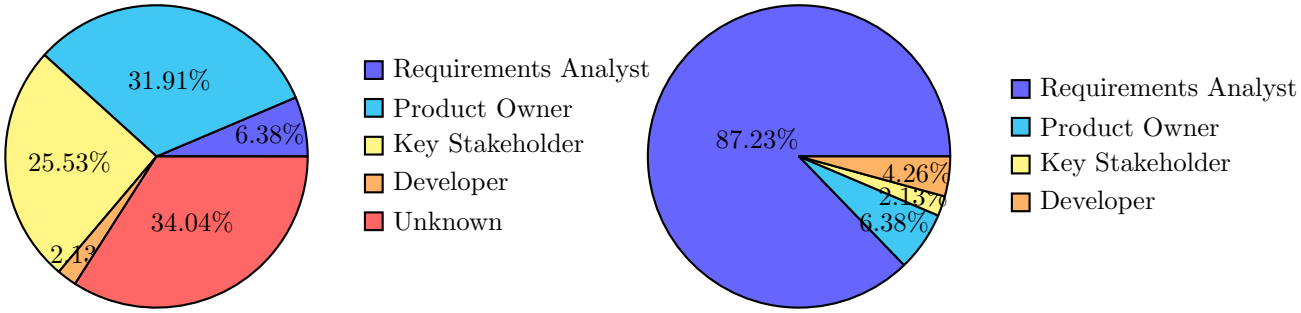

Figure 5.13: Distribution of initiators for added requirements based on source location (left-hand side) and based on entered in project management system (right-hand side)

In this figure we see that only a small amount of needs for requirements are definitively initiated by the *requirements analyst* or *developers*. Most of the needs are expressed by the *product owner* and *key stakeholders*, this is as to be expected as these will be the end users of the system, therefore they will have most ideas as to what should be added to the system. The *unknown* initiators come from the *unknown* source locations for some of the `add` tags. When looking at the person actually adding those specific requirements to the project management system, we get the right-hand side of the figure, and we see that most of them are added, as expected, by the *requirements analyst* (68.75%). Both the *product owner* and the group of *developers* enter just as many requirements to the system, which is 12.5%, lastly, only one requirement (6.25%) is entered by the group of *key stakeholders*.

When looking at all the tags besides the `add`, we work with the initiator in a hybrid form.

If there is knowledge of a source and a person first suggesting the change, we work with that person as the initiator. This happens in 11.94% of the cases. When there is an unknown source location, we work with the person entering the change as the initiator. This results in the distribution for all tags besides `add` as shown in Figure 5.14.

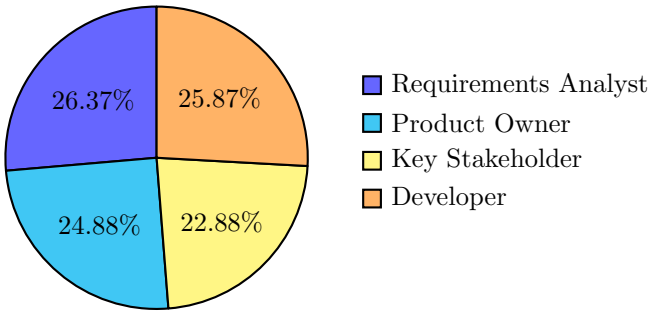

Figure 5.14: Distribution of initiators for all evolution tags except add

We can see that every role initiates around as many changes that resulted in an evolution tag, except for the testers. They did not change anything to the requirements resulting in evolution tags. It is important to keep in mind that these are roles, not people, for example, the requirements analyst role was fulfilled by only one person, whereas there were three developers. Similarly, there is only one person categorized in the product owner role, where there are three in the key stakeholders category.

	Add	Delete	Refinement	Modify	Rewording	Prescriptive Explanation	Descriptive Explanation	Traceability	Indirect Traceability
Requirements Analyst	20.90%	0.00%	14.93%	2.99%	2.99%	23.88%	5.97%	26.87%	1.49%
Product Owner	25.37%	0.00%	10.45%	1.49%	1.49%	14.93%	1.49%	20.90%	23.88%
Key Stakeholder	22.03%	3.39%	22.03%	0.00%	5.08%	8.47%	3.39%	18.64%	16.95%
Developer	5.45%	0.00%	3.64%	3.64%	0.00%	5.45%	1.82%	38.18%	41.82%

Table 5.6: Table supporting 5.15 showing distribution percentages per role

When examining the initiation of evolution tags by different roles, several notable results emerge. The results are visualized in Figure 5.15, two supporting tables are added that display the percentages of this distributions on two ways. The first table, Table 5.6, displays the distribution of the initiators of a tag by role. Here we can see that of all tags initiated by the *requirements analyst*, 20.90% are `add` tags. Table 5.7 takes the perspective from the tag point of view, instead of the role point of view. In this table we see the distribution of the initiator per tag. In this table one can see that of all `refinement` tags, 31.25% was initiated by the *requirements analyst*. When looking at Figure 5.15, Table 5.6 and Table 5.7, especially the distribution for the *developer* role is quite noteworthy. Here we see that there are two tags that stand out, the `traceability` and `indirect traceability` tags. Of all the changes initiated by the *developers* resulted in evolution tags, 80% resulted in either `traceability` or `indirect traceability` tags. This could be explained by the working

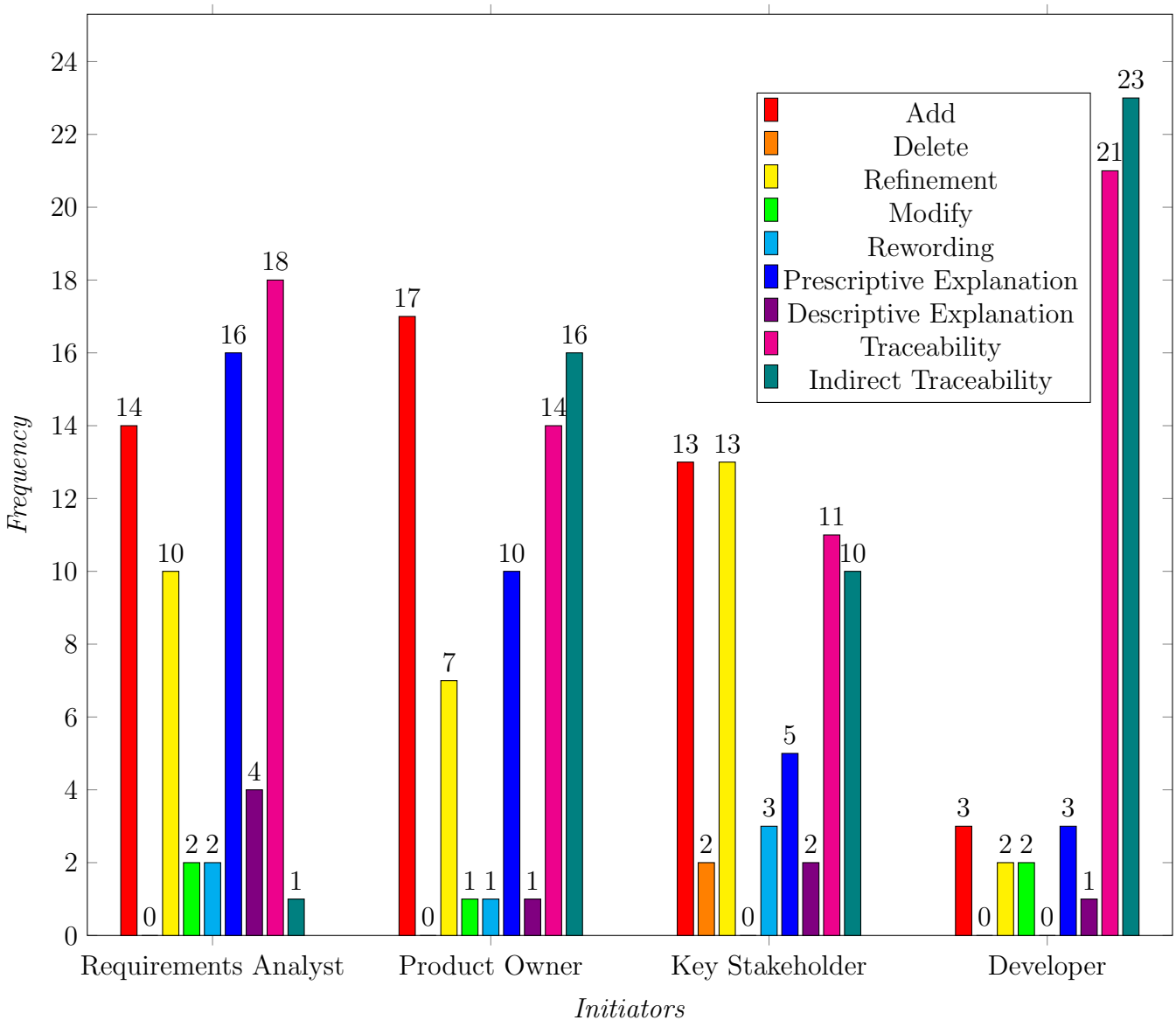

Figure 5.15: Evolution tag distribution grouped by initiator

method of the company, where *developers* link to the system once they finish implementing the requirement. This is discussed with the stakeholders in the results chapter of this thesis.

In the *requirements analyst* their evolution tag distribution (Figure 5.15), we see that the largest amount of changes are labeled as `traceability`. When zooming in on that, we can see that, contrary of the *developers* group, here most of the `traceability` tags do not come from the link to a different artefact, but they originate from relations between requirements. Of all the *requirements analyst* their `traceability` tags, 88.89% originate from relations. So this shows that there is a clear distinction to be seen, between the high

	Requirements Analyst	Product Owner	Key Stakeholder	Developer
Add	29.79%	36.17%	27.66%	6.38%
Delete	0.00%	0.00%	100.00%	0.00%
Refinement	31.25%	21.88%	40.63%	6.25%
Modify	40.00%	20.00%	0.00%	40.00%
Rewording	33.33%	16.67%	50.00%	0.00%
Prescriptive Explanation	47.06%	29.41%	14.71%	8.82%
Descriptive Explanation	50.00%	12.50%	25.00%	12.50%
Traceability	28.13%	21.88%	17.19%	32.81%
Indirect Traceability	2.00%	32.00%	20.00%	46.00%

Table 5.7: Table supporting 5.15 showing distribution percentages per evolution tag

amount of `traceability` tags for the *requirements analyst* and the *developer*. Where one role primarily adds traces to other requirements to clarify relations, the other focuses more on adding traces to the system to showcase the implementation of the requirement.

For this role of *requirements analyst*, we also see the highest amount of `prescriptive explanations`, compared with all the other roles. This could be interpreted by the high discrepancy between the number of adds initiated by *requirements analyst* based on them suggesting it in a *requirements conversation*, and the number of adds where the *requirements analyst* was the person entering the requirement in the *project management system*. The latter is way higher, this means that only a small amount of the requirements added in the *project management system* by the *requirements analyst*, were actually first suggested by the *requirements analyst*. Of all the `add` tags, 57.45% were first suggested by another person than the *requirements analyst*, but were added to the *project management system* by the *requirements analyst*. This could result in the high `prescriptive explanation` tag amount for the *requirements analyst*, as they might need more context to explain what the other person told them their wishes were for a requirement.

5.7 Conversational Analysis

Lastly, we analyzed the requirements conversations more in depth. In the analysis above we had the main focus on the granularity level as all registered changes in the *project management system*. We did already look into the initiators and sources of tags that appeared in the *project management system*, but in this section we will be looking in to the discussions that were held during the conversations and what the impact of those was. Finally we will also analyze what kind of evolution can be seen within a discussion.

5.7.1 Discussions Resulting in Changes

For this analysis we chose the focus on the second type of conversation as defined in the Case Study Protocol (Appendix A), this is the recap meeting. The characteristics are that there had already been a kick off meeting, where all stakeholders got onto the same page, and in between the kick off and the recap meeting a first *requirements specification document* has been created. In this recap meeting, this document is discussed and, where needed, additions, changes or removals of those requirements are discussed. For this analysis, we chose this type of meeting to investigate further, as we saw in the timing analysis that many changes were made after meetings, and in the early phases of the project.

We analyzed the conversation and first identified the discussions in the conversation. We mark an exchange in the conversation as a discussion if it fits the following two criteria:

- At least two people are involved in the exchange
- There is a main topic related to a requirement or other requirements relevant information being discussed

With these criteria, we found 21 unique discussions in the conversation, these are all discussions about different requirements. One discussion was held at two different points in time, but it was about the same topic so it is counted as one discussion. Two additional times were identified where remarks were made about requirements, but only one person made the remark so those are not counted as discussions.

There are three options we looked at for those 21 discussions, and displayed this in Figure 5.16. The first option is when a discussion led to a change in the requirements in the *project management system*, the second option is when this is not the case, but also not expected as, for example, the conclusion of the discussion was that no change was needed. Lastly, we have the option when there is a discussion, where one would expect to see a change in the *project management system*, but that change did not take place.

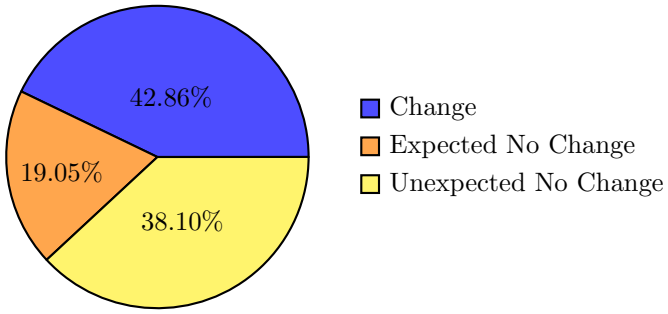

Figure 5.16: Distribution of what happens after a discussion

The figure shows that the majority of the discussions leads to no change, however, for 19.05% this is expected. These are discussions where the conclusion was that there was no

need for any change. 38.10% Of the discussions also did not lead to a change in the *project management system*, even though one might expect to see such change. One example was of a discussion that took place where the *requirements analyst* asked if there was a need for adding a requirement about automation, both *product owner* and *key stakeholders* discussed, and agreed that such requirements should be added, however they were never actually added, possibly because they were forgotten.

Another example is when there were discussions where the conclusion was that in the near future some additional information should be gathered on a specific topic. However, for this research we gathered data for seven to eight additional months and that additional information was never entered in the *project management system*. Here a reason could be that later outside of this meeting that additional information was deemed irrelevant, or for some other reason intentionally left out, or it was forgotten.

Lastly, another possible reason for not seeing a change in the *project management system* was for discussions that did not reach a definitive conclusion. These were discussions where we observed that there is a conclusion missing but there had been some back and forth by the discussion participants on the topic. After a couple of times going back and forth it is interpreted as if there is a common understanding established on the requirement, and it is left with that. However, this leaves a lot of room for interpretation and therefore might not result in a change as the participants in the discussion might look back on it feeling unsure what the next steps should be. Another reason might be that it feels that there is no need to document the change, as most stakeholders were present in the meeting and a feeling of a shared understanding could have been present, which makes adding it to the *project management system* feel redundant.

5.7.2 Requirement Evolution within a Conversation

Besides comparing what discussions led to an evolution tag in the *project management system*, it is interesting to see what evolution takes place during the discussions. To do this we worked with the same tags as for the earlier discussed evolution, however, we are now working on a different granularity level. In the main data set, there were clear data entries that could be tagged where every time a stakeholder saved something in the *project management system*, this was registered. Within a discussion there are not these clear borders present, where we can easily differentiate one state of a requirement to the next.

First we looked at the possibility of working with end points of discussions as a state of a requirement, however, as stated before there was only one discussion about the same topic that occurred at two different points in the conversation. All the other discussions did not re-continue at different points in time, so for these the evolution would then only be one change instead of a series. Therefore, we then looked within a discussion, and we decided to tag all the relevant speaker turns with evolution tags if there was any. For this to work we changed the tags a little bit, because in our original set we worked with entries in the *project management system*, where each item that could be tagged had to be saved by the stakeholder first. This makes us assume that in general, stakeholders will take a

little bit more time to think about their change before saving it, as entering it in the *project management system* directly impacts the work of coworkers. Moreover, during a discussion a stakeholder will take less time thinking about what to say, then they do thinking about changes they will make in the *project management system*. Additionally, in the original set, we only labeled actual accepted changes in the requirement, for this part we label all the suggestions separately. Even when a suggestion to `modify` a requirement is not accepted, in this part we will tag it as a `modify`. Most of these adjustments are made purely to have more insight in the difference in an evolution when working with it in a place where there is more time to reevaluate what change is entered, opposed to in discussions where direct thoughts get spoken out.

The concept of sequence differs from the previous tagging as well. Previously, a sequence was a set of evolution tags that started with an `add` and every tag within a sequence surrounded one main requirement. In the discussions each discussion is represented by its own sequence, and they do not necessarily start with an `add`. Additionally, here an `add` could take place at any time in a discussion sequence.

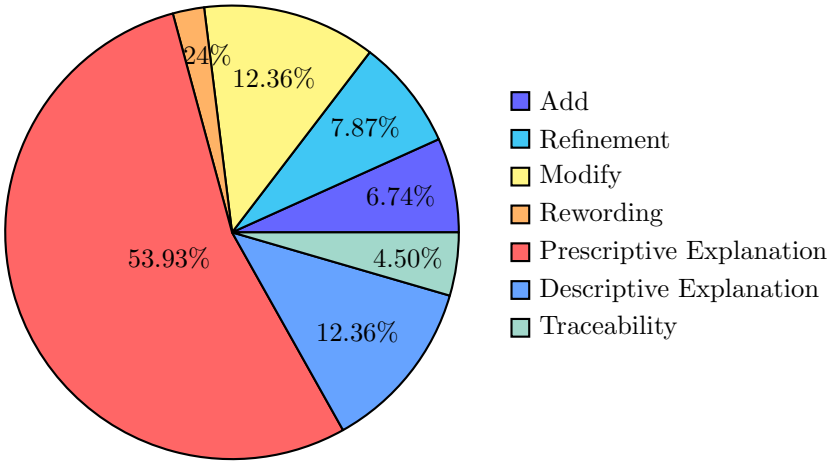

Figure 5.17: Distribution of evolution tags in discussions

This tagging scheme resulted in 21 sequences and a total of 89 tags. The distribution of these tags is displayed in Figure 5.17, where we clearly see that the majority of tags in the discussions are the `prescriptive explanation` tags. Reason for this can mostly be found in the nature of how the discussions took place, where one requirement was being discussed at the time. All the participants of the conversation could then comment on that requirement, which was mostly in the form of explaining what they thought the requirement should adhere to, which is tagged as a `prescriptive explanation`.

The relatively high amount of `modify` and `rewording` tags, compared to the previous tagging, is interesting as well. This could come from the fact that in the previous tagging, we worked with only saved instances of a requirement. Here modifications were made less often as it seems that the requirements were being formulated well before entering them

	Project Management System	**Conversation**
# Sequences	47	21
# Evolution Tags	251	89
Average Sequence Length	5.3	4.2
Longest Sequence Length	19	11
Median Sequence Length	4	3
Most Occurring Tag	Traceability	Prescriptive Explanation
Least Occurring Tag	Delete	Delete, Indirect Traceability

Table 5.8: Comparison of results project management system and conversation analysis

to the system. In these discussions, we see moments where a stakeholder would propose a modification of the requirement, then another participant proposes a different modification again, or modifies it back to the original, and the requirement in the *project management system* is never modified.

Lastly, we have the noticeable results for the `(indirect) traceability` tags. Where those had the highest distributions in the previous tags, they now are at the bottom of the distribution, with `indirect traceability` even not being tagged once. The reason for this can be the relative early stage of the project when this conversation took place, so there were not many different sources the participants in the conversation could trace to.

In Table 5.8 we put a side-by-side for some of the properties of the analysis done on the *project management system*, and on the *requirements conversation*. We see a clear difference between the two, where conversations their evolution are shorter than those of the project management system. Both the average, and the median of the sequence length, is one tag shorter for *requirements conversations* opposed to the *project management system*. Besides that, we also see that for the *project management system* the most occurring tag was the `traceability` tag, whereas we saw in Figure 5.17 that this tag is at the bottom of number of occurrences. Additionally, the second most occurring tag in the *project management system* was the `indirect traceability` tag, which is never identified in the conversations.

Table 5.9 showcases the same information as Table 5.2 in the beginning of this chapter. However, this table focuses on the information form the discussion. In order to easily be able to compare it with the results in the table at the start of this chapter, we added the last column, which are the results from that table about the *project management system*. In this table, we now have the last two columns that work with a percentage to present in how many sequences at least one of the evolution tag of that row is present.

The `add` tag in the sequences created from information of the *project management system* is 100%, whereas that of the information from the discussions is only 23.81%. This can be explained by the difference in nature between these two types of sequences. As explained

	Total number of occurrences	Sequences containing tag at least once	Sequences (%)	Sequences (%) Project Management System
Add	6	5	23.81	100.00
Delete	0	0	0	4.26
Refinement	7	6	28.57	42.55
Modify	11	7	33.33	8.51
Rewording	2	2	9.52	8.51
Traceability	4	3	14.29	48.94
Indirect Traceability	0	0	0	55.32
Prescriptive Explanation	48	15	71.43	53.19
Descriptive Explanation	11	10	47.62	14.89

Table 5.9: Amounts of occurrences of evolution tags in the conversations compared with the project management system

earlier, the sequences created from the *project management system* always start with an `add`, because that is the start of the requirement in the physical space, it is always added to the *project management system* before it can evolve further. In the *conversations* this is not the case. Especially in the *conversation* chosen for this analysis, whose goals was to reevaluate the *requirement specification document*, so most of the adds are found in the previous *conversations*. Here each unique discussion counts as its own sequence, only when the need for a new requirement is discussed in this meeting, an `add` tag will occur. Which is happening way less often.

The `delete` tag is occurring in only two of the previous sequences, and zero of the new sequences. The reason for this has been discussed extensively at the beginning of this chapter. However, an interesting note to add is that there were two discussions in the *conversation* that led up to the two `delete` tags in the *project management system*. The reason for those two `deletes` to not show up in this overview is because the actual deletion of the requirement had not been talked about by anyone, but the conclusion of the *requirements analyst* was to `delete` the requirements anyway.

For the `refinement` tag, the results become more interesting when they are put into perspective. Because for these results it is important to keep in mind that the second to last column is only about one *conversation*. So in this one *conversation*, 28.57% of the discussions contain at least one `refinement` idea. This, compared with the fact that in the *project management system* in the duration of the whole case study only 42.55% of the sequences contained at least one `refinement` is noticeable. It appears as if there have been many more

`refinement` ideas in *conversations*, than actual `refinements` added in the *project management system*.

The `modify` tag is way higher in the discussions than in the system. The reason for this can be similar to the reasons for the `prescriptive explanation` and `descriptive explanation` tags occurring in more percent of the discussions than sequences. This could be because of the nature of discussions, where it is the goal that any idea or unclarity can be discussed about a requirement. In these discussions, a lot of explanations are suggested by stakeholders to clarify what is meant with the requirement, or how they think the requirement should be handled. In this discussion, also a lot of modifications can be suggested as there might not be a shared understanding about the requirement yet. Many of these explanations or modifications are disregarded in the continuation of the discussion once a shared understanding is created. Therefore, not all explanations and modifications will make it into the *project management system*. Also, as discussed earlier about the discrepancy between the number of discussions with expected changes to be entered, and the actual changes entered in the *project management system*, stakeholders might not always feel the need to enter the discussed change in the system. They might feel that a shared understanding is agreed upon and that no further documentation about this is needed.

Chapter 6

Results

6.1 Findings

In this chapter, we discuss the results from this case study. We start by listing the findings of the analysis from Chapter 5. To give more context to the results, we had a discussion with two stakeholders of the case study. The findings that came out of this discussion are listed in a separate list. Then we will discuss these findings more in depth structured by the main topics of the research questions. After that, we discuss the threats to validity of this research and propose topics for future research. Lastly, we finalize this thesis by summarizing the research.

6.1.1 Analysis Findings

The findings of the research analysis are summarized below, after each finding a link is placed to the section, figure or table where the evidence of the finding has been described in depth.

1. 75% Of the sequences of length seven or longer, contain at least one `refinement` tag (Section 5.1.2)
2. The *clarification* category of tags is the most unevenly distributed of all evolution tag categories (Figure 5.2)
3. Tags from the *irreversible changes* category are least likely to occur in a sequence (Figure 5.1)
4. Where most direct links (`traceability`) link to issues, most indirect links (`indirect traceability`) link to the system, even though those are both targets that could be referred to with either a direct or indirect link (Figure 5.4)
5. If there is a `modify` tag in a sequence, there is a significant increase of 5.036 in the likelihood of `descriptive explanation` being present in that same sequence (Table 5.5)
6. The status of a requirement can give an indication of the likelihood of a specific evolution tag occurring (Section 5.4.2)
7. Tags from the *clarification* or *irreversible changes* category are more likely to occur within the first two steps of the evolution sequence of a requirement than later in the

sequence (Section 5.2.5)
8. The role of the person initiating an evolution tag can give an indication of the likelihood of a specific evolution tag occurring (Section 5.6)
9. Conversations have their own evolution of requirements on a different granularity level as the evolution seen in a project management system (Section 5.7)
10. Not every evolution in a conversation is being translated to an evolution in the project management system (Figure 5.16)
11. Evolution sequences in conversations are on average shorter than evolution sequences in a project management system (Table 5.8)

6.1.2 Interview Findings

To give more context to the results and possibly fill in some gaps, we interviewed the requirements analyst and one of the key stakeholders from the case study. In this discussion with two stakeholders, we first presented what we analyzed, and then we discussed those results.

Some context for the findings above was given, and three additional findings were discovered based on this conversation, these findings are listed below.

In the conversation we found that specifically findings 6 and 8 from the list above can be linked to finding (ii), and the working method of the company can explain some results. For example, we saw in Figure 5.15 that most of the tags initiated by the developers are from the *linking* category, and in Figure 5.10 that from the *linking* category most were added in the *in realisation* phase. These results can be tracked back to the working method, the developers mostly work with requirements once they have the *in realisation* status, and developers are expected to link to the system to show what is implemented for that requirement. These links were then tagged as `traceability` or `indirect traceability` tags. This could therefore be directly linked back to the working method of the company.

(i) Despite stakeholders expecting many steps in the evolution of a requirement due to it being an internal project, sequence lengths under 7 are 74.46% more common than longer lengths (Section 5.1.2)
(ii) Results are dependent on the working method of a company
(iii) Results are dependent on the project being internal or external

Another finding that came from the conversation with the stakeholders was finding (iii). This is a finding is based on the feedback the stakeholders gave where they discussed differences and similarities between this internal project that we followed with the case study, and their regular external projects. A few of the key points that are different and would most likely change the results according to the stakeholders are listed below:

- There was no strict scope as there would be in an external project with limited resources. This made that requirements could be added at any point, whereas otherwise that would not be the case.
- There was less priority for the project, so it took longer to finish, especially with longer breaks in between. According to the stakeholders, this led to circling back to requirements more often than they feel they do in external projects.
- There was more informal communication about the project as all stakeholders were direct coworkers. Therefore not all communication was documented as well as it would have been in an external project, especially when a requirement was not completely clear, someone would just talk to someone else and ask for a clarification, which was then not always documented.

6.2 Existing Knowledge

The first main topic comes from the first sub-question (SQ1, 2.1), which is: *"What knowledge is available regarding requirements evolution in current research?"* This is mostly researched in Chapter 3, Literature Study, where we saw that there is not much research done on the topic of requirements evolution yet. Currently, it seems as if there is awareness on the topic, as researchers called for more research on the topic, but there is little knowledge available. The three studies done more in depth on the topic differ quite a lot which implies that there is no shared understanding as to what requirements evolution entails. For this research, we combined the taxonomies of two of these three studies. The third study and its taxonomy was disregarded for this study as it worked with a completely different viewpoint from what we aimed to research. That study worked with requirements in a phase we did not look at, the phase after deployment of a system, this made the taxonomy not fitting for our data set. Lastly, we combined the knowledge from these few studies on requirements evolution with knowledge available on requirements change, as this is a more intensively studied topic with a close relationship to requirements evolution.

So, there is not much knowledge available on requirements evolution from current research. However, the knowledge that is available shows that there are multiple viewpoints and taxonomies to work with when researching this topic. Moreover, the requirements change research field could also be used as it is closely related to requirements evolution.

6.3 Evolution Patterns

The second focus point is the patterns in requirements evolution, the second sub-question (SQ2, 2.1), *"What is a typical evolution pattern for requirements?"*, and the first hypothesis (H1, 2.1), *"It is possible to identify case independent evolution patterns."*, are related to this topic. The hypothesis could not be tested in this research as two of the three cases did not move past the second phase, making it impossible to analyze evolution patterns from it.

Moreover, only having one case with evolution patterns does not give enough data to test whether or not these patterns are case independent.

As for the typical evolution patterns found in the case that did move forwards, we found that we cannot identify a clear pattern in the evolution from the current data, such as the pattern of children first losing their front teeth, then new teeth will come in their place and later lose their back teeth [71]. There was no strong evidence for such a pattern, however, we did find that tags from either the *clarification* or *irreversible changes* category are more likely to occur either right after, or one after that, the `add` or `refinement` tags. Besides that, we were able to identified some likelihoods of tags occurring together in the same evolution. We found the following rules from ARM:

- If there is a `modify` in an evolution, with a confidence of 75%, it is 5.036 times more likely to also have a `descriptive explanation` in that evolution than for it to occur in that evolution if `modify` would not be present
- If there is a `modify` in an evolution, with a confidence of 75%, it is 2.518 times more likely to also have a `refinement` and a `prescriptive explanation` in that evolution than for them to occur in that evolution if `modify` would not be present
- If there are a `descriptive explanation` and a `refinement` in an evolution, with a confidence of 75%, it is 2.203 times more likely to also have a `traceability` and a `prescriptive explanation` in that evolution than for them to occur in that evolution if `descriptive explanation` and `refinement` would not be present
- If there are an `indirect traceability` and a `descriptive explanation` in an evolution, with a confidence of 75%, it is 2.203 times more likely to also have a `traceability` and a `prescriptive explanation` in that evolution than for them to occur in that evolution if `indirect traceability` and `descriptive explanation` would not be present

6.4 Evolution Origin

The last topic is where we focus on the origin of evolution steps or as its whole. The remaining sub-question (SQ3, 2.1), *"What are triggers that can result in an evolution of a requirement?"*, and hypotheses (H2, 2.1), *"Requirements with a different origin will have a different evolution pattern."*, and *"Requirements that are added later in the project will have a different evolution than requirements that are added from the beginning."*, (H3, 2.1).

The sub-question is about triggers, from the analysis we found some triggers, but the conversation with the stakeholders resulted in the complete list below, with things that were identified which could lead to an evolution.

- Discussion with peers (in meetings, through comments in the project management system, in a private chat both online and offline, in informal conversations)

- Mandatory part of ones job (e.g. developers adding links to the system once they implemented a requirement)
- Interacting with the system (could spark ideas for additional feature requests)

Hypothesis 2 states that there will be a difference in requirements that have different origins, in the analysis we looked at origin from two different viewpoints, first the source, which could be the formal meetings, project management system or unknown, and second the initiator, which could be a role within the project, either actually making the change tracked by the project management system or the first person suggesting it in a formal meeting. Within this data we found that the earlier discussed evolution patterns and other properties such as length, stay consistent for all requirements independent of their origin source or origin initiator.

The timing mostly is important for hypothesis 3, where we hypothesized that the evolution will differ for requirements added at different times in the project. In the analysis we found that there is no strong evidence that there is a difference. 78.72% of the requirements have been added within the first few weeks of the projects, and we see that the evolution patterns as discussed above, and other properties such as length, stay consistent for all requirements independent of when they were added to the project.

6.5 Discussion

In this section we will discuss the validity of this research, with the guidelines of Runeson and Höst [22]. They argue to use the validity classification scheme categories similar to the one typically employed in controlled experiments, for case studies as well [22]. We follow their guidelines, therefore discuss the construct validity, internal validity, external validity and reliability. All categories and an extensive explanation of them can be found in the work of Runeson and Höst [22].

Construct validity is the first category to discuss. In this category, no clear threats can be identified as we used different measures to prevent this. To begin with, we work with data triangulation, as we utilized data from multiple sources. We gathered data from the project management system, from the requirements elicitation document, from the meetings recordings and transcripts, and lastly, from the interview with stakeholders. We also used multiple methods of data collection, for example, we both did interviews as well as observations. Lastly, to minimize the threat to construct validity, we described the case study as detailed as possible and we had peer reviews of the taxonomy and labeling.

The internal validity threat is one that can be identified in our research. As we already identified earlier, the working method of the company of the case study has impacted the results. As described before, the developers of the company mostly contributed to the tags

in the linking category, and as described this could be explained by the working method of the company, where developers are expected to link their implementation of a requirement. Additional research needs to be done to research whether those results are actually role-bound, or whether they are dependent on the working method of this specific company.

External validity is one of the threats that is a little different for case studies, as one of the inherent limitations of a case study is that there is no statistical generalizability [72]. For case studies, the aim is to facilitate analytical generalization, allowing the findings to be extended to cases sharing common characteristics and, thus, making the results applicable and relevant. This is done by carefully selecting a case to follow that fitted the case study selection criteria from the case study protocol.

Finally, we have the reliability of the research. At the beginning of this research we had lower inter-coder and intra-coder reliability. One author of this research tagged the data multiple times at different points in time, at the early phases there were many differences in two sets of tags. Another author also tagged parts of the data, here there were also many differences at the early phases. After some iterations and tweaking the definitions in the taxonomy, one author kept tagging consistently, and the two authors also tagged consistent 90% of the time. This made for both better inter-coder reliability and better intra-coder reliability.

As of the benefits of this research, we do see this research as an important addition to the scarce set of research on the topic of requirements evolution. With this research we contribute to this set in the unique way of working with a real software project, whereas other research only work with data created for the purpose of the study, not for the actual need of a new product. In this study we did follow a real case where a product has been created, that would have been created similarly if there was no case study. Besides this, we also gathered a large amount of data in the conversational RE field and could add some analyses on this data as well as the project management system data, where the latter has mostly been research before, and the conversations have not been researched earlier. This way our research contributes to the scarce field and had exploratory value.

6.6 Future Research

As stated before, the research on this topic is still scarce, which leaves a lot of room for future research. First and foremost, more cases should be done on this topic to validate the findings in this research. Ideally, first more cases where an internal system is created, to keep that variable the same, should be researched. Later this should be expanded to external projects, to find similarities and differences in these types of projects.

Besides validating the results, research with a different point of view would be interesting future research as well. One of the viewpoints that would be interesting for business is mea-

suring success of a requirement, and seeing if there is a connection between evolution and the success rate. Whether the value or success of a requirement can be indicated beforehand by looking at the role of the person introducing that requirement. Interesting would be if a recommendation could be given if a connection is found, for example that more awareness should go to requirements initiated by a certain role, as they might be less successful so they can be changed or elevated right from the start.

Another interesting topic for future work is additional research on the link between the evolution that happens during discussions in meetings, and the evolution documented in the project management system. We already did an analysis on this, but we only focused on the recap meeting. For future works different phases of meetings should be analyzed to find patterns in those different types of conversations.

Additionally, we think in future research, a set of tags for its own should be looked in to for the conversations evolution tagging. In our research we worked with the same set of tags, however, we saw that the linking category of tags is less fitting when discussing the requirements in a meeting, than they are when the requirements are documented. We also saw that the line between some tags of the same category got way thinner when being used in a discussion opposed to in a project management system. Reason for this could also be the granularity differences, and the fact that in the discussion every in between state, which can be separated by speaker turn, gets a tag, whereas in the project management system only the accepted changes and additions are tagged. For this research using the same tags for both parts worked, but it would be interesting to do some more research on generating the most fitting tags for conversations and linking those tags to the tags used for the project management system.

6.7 Summary

In this research, we attempted to do an exploratory case study on three cases, following an internal software project from start to the completion of an MVP, going through at least four pre-defined phases, to explore the topic of requirements evolution. We aimed to tag and analyze the data from the project management system, the requirements specifications document and the formal meetings during the project, to investigate for interesting links, patterns or other connections. In the end, due to circumstances within the case companies, only one project was completed following the necessary phases for our research. We followed that internal project for nine months, from the start of project with its kick-off meeting and first requirement specification document, to the MVP being developed and the requirements from the second version of the product being implemented. In this time span, we gathered 391 data entries from the project management system, distributed over 93 issues.

We did a literature research (Chapter 3) to find out if there were already existing taxonomies for the tagging of the requirements evolution, and found out that in the scarce amount of research on the topic, multiple different viewpoints were used. Three taxonomies

were identified, and for our research we decided to combine two of those three, as the third one used a viewpoint not fitting for our research. Not all the tags had a clear definition and explanation, and some could be split up to create a more uniform taxonomy ourselves. We defined these tags in multiple rounds, after which we started tagging the data set. Here we encountered some difficulties with the definitions of the tags and took some more revision rounds on those tags. Then two of the authors tagged parts of the data set again, discussing the differences with a third author and making changes where needed to the definitions. After two of those rounds, we came to an agreement and used the tags as defined in this thesis in Chapter 4.

After tagging the data set one last time we started the analysis of the data from the project management system first in Chapter 5. Here we started with some basics about the data set, looking at the distribution of the categories of the tags, and the tags itself and the sequence lengths. We then looked at occurrence of tags, and tags occurring together, how often we could observe those in a sequence. We ended with association rule mining for this part, to give it more meaningful context as we worked with a large data set and big differences in the amount of occurrences of tags towards each other. We then analyzed the impact of timing, source and initiator on the requirements and their evolution. Lastly, for the analysis we looked at the conversations as well, where we looked at the evolution seen in such a conversation, as well as the differences and similarities with this evolution opposed to the evolution found from the project management system data.

In the analysis, we found some results of which we hypothesized that they could be linked to the working method of the company. To test that hypothesis we set up an interview with two stakeholders from the company, the requirements analyst and a key stakeholder. In this interview we started by displaying our results and then we had a discussion about those, here we found that indeed some results could be linked to the working method used by the company.

In Chapter 6, we presented our findings and answered our research questions, the questions about findings being case independent could not been answered as we were not able to collect data from the three cases we planned beforehand. Despite this we wrote in our discussion about this research still being an interesting addition to the scarce amount of research done on the topic, especially as it fills the gap on research in the requirements evolution domain that works with a real project instead of simulated data or data created for the purpose of that research.

To conclude, in this research we were not able to find patterns in evolution that are case independent, but we did find indications of certain evolution tags often occurring together. We also contributed to the research field with our taxonomy of evolution tags with their definitions, so that future research could use our taxonomy as well, so that we can create a uniformity in this unknown field.

Acknowledgements

I would like to thank my supervisor, Prof. Dr. Fabiano Dalpiaz, and daily supervisor, Tjerk Spijkman, for their guidance, support, time, and encouragement throughout the journey of completing this thesis. Their expertise, insights and meaningful discussions have been of great importance in shaping the direction and quality of this work.

I would like to acknowledge and express my gratitude to my second supervisor, Prof. Dr. Sjaak Brinkkemper, especially for his valuable feedback during the early stages of this thesis.

I would also like to thank all members of the RE-Lab for their motivational words and great insights.

Lastly, I would like to express my sincere gratitude to the companies involved in the case studies for their trust, cooperation, and openness in sharing both their insights and data. Without their willingness to collaborate and provide access to this crucial information, this research would not have been possible.

Appendix A

Case Study Protocol

Protocol based on the guidelines of Maimbo and Pervan

In this appendix we will give a complete case study protocol based on the guidelines as described in Designing a Case Study Protocol for Application in IS research [27].

Preamble

This case study protocol is created for a Business Informatics master thesis where a case study is conducted for three different cases at two different companies. The case study functions to collect data from requirements conversations, the requirement specification document, and from the project management system the requirements are stored in. Parts of the case study that are important for the thesis, and the results of the case study will be published as a Business Informatics master thesis at Utrecht University. All the data from the case studies will be anonymized when added within the thesis.

General

The aim of the research is to explore what types of requirement evolution can be identified through the course of a software project. There has been done quite some research on RE and techniques on how to elicit requirements [9], and we know that conversations are one of the most used techniques to elicit requirements [8]. Yet there is not much research done on this specific part, the requirements elicitation conversations. Additionally, requirements evolution from these conversations is also acknowledged to be important [21], yet also not discovered in combination with the above discussed conversations. So it is important to do this research to on the one hand to discover a new field in research, and on the other hand to collect real-world data.

To gather this data and explore this part of research, a case study of three cases is conducted. We aim to follow three cases which are all internal software projects, at two different companies. For each case there is an internal requirements analyst whom will

create or update a requirements list. We will use this list to determine an evolution from one conversation to the next. Not only what the evolution is, but also why it happened when it happened. Did the requirements analyst misunderstand a requirement, did a discussion take place to change it? However, an evolution can also take place in the development process of a product. Therefore, we will also extract as much information as possible from the project management system the cases use, to see if changes were made in the requirements in that system. We will record the structured requirements elicitation conversations so that we will be able to determine where the requirements came from.

Procedures

The choice to conduct the case studies at company A and company B is based on the affiliation two members of the research team have with these companies. The cases have to fit the following criteria:

- It has to be an internal project. So internal stakeholders, internal use, internal development. This criterion is added to minimize the number of hard-to-control variables, to add a uniformity in the different cases.
- The project has to have started at latest in September 2022.
- The project has to have had four requirements elicitation conversations by November 2022.
- The project has to have someone that will act as a requirements analyst that creates and updates a requirements list after each conversation.
- The functional requirements should be documented in user story format. This criterion is added for uniformity in the different cases. This is mostly for the functional requirements identified from the conversations. Requirements that come forth out of necessity when developing for example do not have to use this format.
- The project has to keep track of the requirements in the development process with a project management system
- The project has to have at least four conversations of requirements elicitation that result in a (updated) requirements list, of the structure that is discussed in the next part.

So the flow of the projects also have one thing that has to be the similar. This is an iterative process so there can be any number of additional conversations between two steps, but at minimum all projects should have the following structure:

- *Kick off*
  First conversation about the project to get every stakeholder on the same page. Results in a first requirements list.

- *Recap*
  Second conversation is about the recap and review of the requirements list. The result of this conversation is either the approval of the list or an altered version. The alteration of this list does not necessarily depend on this specific conversation. As the projects are all internal, we assume there can be some informal discussions that take place besides these main conversations that alters some conversations. Depending on the outcome this step can be repeated until there is a consensus on the requirements list.

- *Early validation*
  After an agreement is reached, the development of the prototype can be started. The early validation is some kind of demo. This can be with any kind of prototype, drawing or other visible demo type, and the result is either approval or improvements or changes to be made.

- *Progress meeting*
  This meeting is about the progress that is made from the early validation until now. For larger cases it can be a first progress meeting which is iterated over a couple of times, for smaller cases this could even be the last meeting if everything is approved of.

We aim to be present, or at least have recordings of, at least the four conversations as described above per project, but preferably all conversations. Which is all conversations that are held formally within the project between the requirements analyst and the stakeholders. As they are internal projects we assume there will be some informal conversations that are not planned but where some requirements are discussed as well. We will not be present for these conversations, and they probably will not be recorded. The conversations can be either digitally through Teams or in person.

After obtaining the requirements lists from the analyst of two subsequent conversations We will analyze the evolution of the requirements from the earlier to the latter list. This goes on until either the cases are finished or until November 2022.

Once that point is reached, we will go on to extract extra information from the project management software that was used. For Trello we aim to use the Trello Card History plugin to retrieve information about changes that were made to requirements within the system. For Jira the Issue History for Jira can be used to do the same. When all information is extracted we can create a timeline and see if changes were made from the conversations we also have requirements lists of, or if changes were also made besides that.

When all the data is extracted we will try to build hypotheses around the research questions on the evolution these requirements go through, and differences and similarities for all the cases.

Once the case is finished we will inform the company of the evolution to find out what their view on the evolution is.

Research Instruments

The main research instruments are the following items:

- A voice recorder for the in person conversations.
- Microsoft Teams for the online conversations.
- The list of requirements from the requirements analyst in user story format.
- Trello as the project management software for company B.
- Jira as the project management software for company A.
- Trello Card History plugin to extract information from Trello.
- Issue History for Jira to extract information from Jira.

Triangulation is used as data is gathered both from the list of requirements from the requirements analyst and the project management software.

Data Analysis Guidelines

The analysis of the data will be done as follows. Each conversation will create or update a list of requirements. The evolution these requirements go through from each conversation to the next is the thing I want to extract. We will code this data. The code will be based on findings from the literature review.

After this within case data analysis the cross-case analysis will take place, here the focus will be on identifying a set of patterns of requirements evolution. This set can then explain how and why requirements change the way they do. This is focused on the requirements elicitation conversations and project management system. When this data analysis is finished, we believe the research questions can be answered for this thesis and we will have gained insight in possible future research opportunities.

Appendix B

Sequences

B.1 All Sequences Project Management System

ID	Length 1	Length 2	Length 3	Length 4	Length 5	Length 6	Length 7	Length 8	Length 9	Length 10	Length 11	Length 12	Length 13	Length 14	Length 15	Length 16	Length 17	Length 18	Length 19
1	Add	Delete																	
2	Add	Delete																	
5	Add																		
6	Add	Rewording																	
7	Add																		
9,86	Add	Indirect Tr	Traceabilit	Refinement															
10,11,12	Add	Descriptiv	Traceabilit	Refinemer	Refinemer	Prescriptiv	Indirect Tra	Prescriptiv	Traceabilit	Traceability	Traceability	Traceability	Traceability	Traceability	Indirect Tra	Traceability	Indirect Tra	Traceability	
13	Add	Rewording	Traceabilit	Indirect Tr	Indirect Traceability														
14	Add	Descriptiv	Traceabilit	Indirect Tr	Prescriptiv	Traceabilit	Prescriptiv	Traceability											
15	Add	Prescriptiv	Indirect Tr	Traceabilit	Traceabilit	Traceabilit	Indirect Traceability												
16	Add	Rewording	Rewording	Traceabili	Prescriptiv	Prescriptiv	Traceabilit	Traceabilit	Indirect Tra	Indirect Trac	Refinement								
17	Add	Traceabilit	Prescriptiv	Traceabilit	Traceabilit	Traceabilit	Traceability												
18, 33,34	Add	Prescriptiv	Rewording	Traceabili	Refinemer	Traceabilit	Prescriptiv	Rewording	Prescriptiv	Traceability	Traceability	Refinement	Prescriptive	Traceability	Traceability	Traceability	Refinement	Traceability	Indirect Tra
19	Add	Prescriptive explanation																	
20	Add	Prescriptive explanation																	
21	Add	Indirect Traceability																	
22,83	Add	Modify	Modify	Indirect Tr	Traceabilit	Traceabilit	Refinement	Traceabilit	Prescriptiv	Refinement									
23,47,64	Add	Indirect Tr	Traceabilit	Refinemer	Descriptiv	Indirect Tra	Refinement	Traceabilit	Indirect Tra	Indirect Trac	Traceability	Refinement	Traceabilit	Prescriptive explanation					
24,44	Add	Prescriptiv	Prescriptiv	Indirect Tr	Indirect Tr	Traceabilit	Refinement	Prescriptiv	Indirect Tra	Indirect Trac	Indirect Trac	Traceability	Traceability						
25	Add	Refinement																	
26	Add																		
27	Add	Indirect Tr	Prescriptiv	Indirect Traceability															
28	Add	Refinement																	
29	Add	Indirect Tr	Refinemer	Indirect Traceability															
30	Add	Indirect Tr	Indirect Tr	Indirect Traceability															
32	Add	Prescriptiv	Indirect Tr	Indirect Tr	Traceability														
36	Add	Modify	Descriptive explanation																
37	Add	Traceabilit	Traceabilit	Traceabili	Indirect Tr	Indirect Traceability													
38	Add	Traceabilit	Traceabilit	Refinemer	Traceabilit	Indirect Traceability													
39	Add	Traceabilit	Prescriptiv	Traceabili	Indirect Tr	Refinement	Traceabilit	Indirect Traceability											
40	Add	Prescriptiv	Traceabilit	Indirect Tr	Refinemer	Indirect Traceability													
41	Add	Descriptiv	Traceabilit	Indirect Traceability															
42	Add	Traceabilit	Indirect Traceability																
45	Add	Prescriptiv	Indirect Tr	Indirect Tr	Refinemer	Indirect Traceability													
52	Add	Prescriptiv	Indirect Tr	Indirect Traceability															
53	Add																		
54	Add	Prescriptiv	Traceabilit	Indirect Traceability															
55,56,57	Add	Prescriptiv	Refinemer	Prescripti	Descriptive	Descriptive	Refinement	Modify	Prescriptiv	Traceability									
58	Add	Prescriptiv	Traceabilit	Traceabili	Refinement														
59	Add	Indirect Traceability																	
60	Add	Prescriptiv	Indirect Traceability																
61	Add	Prescriptive explanation																	
62	Add	Prescriptiv	Traceabilit	Traceabili	Refinement														
67,68,69	Add	Traceabilit	Refinemer	Refinemer	Refinemer	Refinement	Refinement	Refinemer	Traceability										
75	Add	Prescriptiv	Refinement																
94	Add																		
100,101	Add	Descriptiv	Modify	Prescripti	Refinement														

B.2 All Sequences Requirements Conversation

ID	Length 1	Length 2	Length 3	Length 4	Length 5	Length 6	Length 7	Length 8	Length 9	Length 10	Length 11
1	Rewording										
2	Modify	Prescriptive	explanation								
3	Refinement	Modify									
4	Add	Descriptive	explanation								
5	Prescriptive ex	Traceability	Modify	Traceability	Prescriptive	Prescriptive	Rewording	Prescriptive	Descriptive	Prescriptive	Prescriptive e
6	Prescriptive ex	Descriptive	Prescriptive	Prescriptive	Prescriptive	Prescriptive	Refinement	Prescriptive	Prescriptive	Prescriptive	Prescriptive e
7	Descriptive ex	Modify	Prescriptive	Modify	Prescriptive	explanation					
8	Descriptive ex	Prescriptive	explanation								
9	Modify	Modify	Prescriptive	Add	Modify	Add					
10	Prescriptive ex	Refinement	Modify	Prescriptive	Prescriptive	Prescriptive	Traceability				
11	Modify	Modify	Prescriptive	explanation							
12	Descriptive ex	Prescriptive	Add	Prescriptive	explanation						
13	Prescriptive ex	Prescriptive	Prescriptive	Prescriptive	Prescriptive	Prescriptive	Prescriptive	Prescriptive	Prescriptive	explanation	
14	Prescriptive ex	Prescriptive	Prescriptive	Prescriptive	Prescriptive	Prescriptive	Descriptive	explanation			
15	Refinement	Prescriptive	explanation								
16	Refinement	Refinement									
17	Descriptive ex	Prescriptive	Traceability	Add							
18	Prescriptive ex	Prescriptive	explanation								
19	Refinement	Prescriptive	Descriptive	Descriptive	Prescriptive	explanation					
20	Descriptive	explanation									
21	Add										

Bibliography

[1] Neil A Ernst, John Mylopoulos, and Yiqiao Wang. Requirements evolution and what (research) to do about it. In *Design Requirements Engineering: A Ten-Year Perspective*, pages 186–214. Springer, 2009.

[2] Juan Li, He Zhang, Liming Zhu, Ross Jeffery, Qing Wang, and Mingshu Li. Preliminary results of a systematic review on requirements evolution. In *16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*, pages 12–21. IET, 2012.

[3] Daniela Damian and James Chisan. An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. *IEEE Transactions on Software Engineering*, 32(7):433–453, 2006.

[4] Alan M Davis, Nur Nurmuliani, Sooyong Park, and Didar Zowghi. Requirements change: What's the alternative? In *2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 635–638. IEEE, 2008.

[5] Sourav Debnath, Paola Spoletini, and Alessio Ferrari. From ideas to expressed needs: an empirical study on the evolution of requirements during elicitation. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pages 233–244. IEEE, 2021.

[6] Alessio Ferrari, Paola Spoletini, and Sourav Debnath. How do requirements evolve during elicitation? an empirical study combining interviews and app store analysis. *arXiv preprint arXiv:2208.00825*, 2022.

[7] Talat Ambreen, Naveed Ikram, Muhammad Usman, and Mahmood Niazi. Empirical research in requirements engineering: trends and opportunities. *Requirements Engineering*, 23(1):63–95, 2018.

[8] Tim Rietz. Designing a conversational requirements elicitation system for end-users. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 452–457. IEEE, 2019.

[9] Alan Davis, Oscar Dieste, Ann Hickey, Natalia Juristo, and Ana M Moreno. Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. In *14th IEEE international requirements engineering conference (RE'06)*, pages 179–188. IEEE, 2006.

[10] Carla Pacheco, Ivan García, and Miryam Reyes. Requirements elicitation techniques: a systematic literature review based on the maturity of the techniques. *IET Software*, 12 (4):365–378, 2018.

[11] Shreta Sharma and SK Pandey. Revisiting requirements elicitation techniques. *International Journal of Computer Applications*, 75(12), 2013.

[12] Masooma Yousuf and M Asger. Comparison of various requirements elicitation techniques. *International Journal of Computer Applications*, 116(4), 2015.

[13] Tjerk Spijkman, Fabiano Dalpiaz, and Sjaak Brinkkemper. Back to the roots: Linking user stories to requirements elicitation conversations.

[14] Tjerk Spijkman, Boris Winter, Sid Bansidhar, and Sjaak Brinkkemper. Concept extraction in requirements elicitation sessions: Prototype and experimentation. 2021.

[15] Neil A Ernst, Alexander Borgida, and John Mylopoulos. Requirements evolution drives software evolution. In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution*, pages 16–20, 2011.

[16] Alicia M Grubb and Marsha Chechik. Looking into the crystal ball: requirements evolution over time. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 86–95. IEEE, 2016.

[17] Chengying Mao, Yansheng Lu, and Xi Wang. A study on the distribution and cost prediction of requirements changes in the software life-cycle. In *Software Process Workshop*, pages 136–150. Springer, 2005.

[18] Rita J Costello and Dar-Biau Liu. Metrics for requirements engineering. *Journal of Systems and Software*, 29(1):39–63, 1995.

[19] Stuart Anderson and Massimo Felici. Quantitative aspects of requirements evolution. In *Proceedings 26th Annual International Computer Software and Applications*, pages 27–32. IEEE, 2002.

[20] Susan DP Harker, Ken D Eason, and John E Dobson. The change and evolution of requirements as a challenge to the practice of software engineering. In *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 266–272. IEEE, 1993.

[21] Neil Ernst, Alexander Borgida, Ivan J Jureta, and John Mylopoulos. An overview of requirements evolution. *Evolving Software Systems*, pages 3–32, 2014.

[22] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164, 2009.

[23] Mai Skjott Linneberg and Steffen Korsgaard. Coding qualitative data: A synthesis guiding the novice. *Qualitative research journal*, 2019.

[24] Johnny Saldaña. The coding manual for qualitative researchers. *The coding manual for qualitative researchers*, pages 1–440, 2021.

[25] Hanny Tufail, Muhammad Faisal Masood, Babar Zeb, Farooque Azam, and Muhammad Waseem Anwar. A systematic review of requirement traceability techniques and tools. In *2017 2nd international conference on system reliability and safety (ICSRS)*, pages 450–454. IEEE, 2017.

[26] Patrick Mäder and Orlena Gotel. Towards automated traceability maintenance. *Journal of Systems and Software*, 85(10):2205–2227, 2012.

[27] Graham Pervan and M Maimbo. Designing a case study protocol for application in is research. In *Proceedings of the ninth pacific asia conference on information systems*, pages 1281–1292. PACIS, 2005.

[28] Eva-Maria Schön, Jörg Thomaschewski, and María José Escalona. Agile requirements engineering: A systematic literature review. *Computer standards & interfaces*, 49:79–91, 2017.

[29] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM transactions on Software Engineering and Methodology (TOSEM)*, 6(1):1–30, 1997.

[30] Pamela Zave. Classification of research efforts in requirements engineering. *ACM Computing Surveys (CSUR)*, 29(4):315–321, 1997.

[31] Murali Chemuturi. *Requirements engineering and management for software development projects*. Springer Science & Business Media, 2012.

[32] Asma Sajid, Ayesha Nayyar, and Athar Mohsin. Modern trends towards requirement elicitation. In *Proceedings of the 2010 national software engineering conference*, pages 1–10, 2010.

[33] Fares Anwar and Rozilawati Razali. A practical guide to requirements elicitation techniques selection-an empirical study. *Middle-East Journal of Scientific Research*, 11(8): 1059–1067, 2012.

[34] Mohsin Irshad, Kai Petersen, and Simon Poulding. A systematic literature review of software requirements reuse approaches. *Information and Software Technology*, 93:223–245, 2018.

[35] Carla L Pacheco, Ivan A Garcia, José Antonio Calvo-Manzano, and Magdalena Arcilla. A proposed model for reuse of software requirements in requirements catalog. *Journal of Software: Evolution and Process*, 27(1):1–21, 2015.

[36] Gunter Mussbacher and Jörg Kienzle. A vision for generic concern-oriented requirements reuse re@ 21. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, pages 238–249. IEEE, 2013.

[37] Thiagarajan Ravichandran and Marcus A Rothenberger. Software reuse strategies and component markets. *Communications of the ACM*, 46(8):109–114, 2003.

[38] V Karakostas. Requirements for case tools in early software reuse. *ACM SIGSOFT Software Engineering Notes*, 14(2):39–41, 1989.

[39] Cristina Palomares, Carme Quer, and Xavier Franch. Requirements reuse and requirement patterns: a state of the practice survey. *Empirical Software Engineering*, 22(6): 2719–2762, 2017.

[40] Neil Maiden. Analogy as a paradigm for specification reuse. *Software engineering journal*, 6(1):3–16, 1991.

[41] Orlena CZ Gotel and CW Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of ieee international conference on requirements engineering*, pages 94–101. IEEE, 1994.

[42] Julia Krause, Andreas Kaufmann, and Dirk Riehle. The code system of a systematic literature review on pre-requirements specification traceability. 2020.

[43] Rosio Alvarez and Jacqueline Urla. Tell me a good story: using narrative analysis to examine information requirements interviews during an erp implementation. *ACM SIGMIS Database: the DATABASE for Advances in Information Systems*, 33(1):38–52, 2002.

[44] Tjerk Spijkman, Fabiano Dalpiaz, and Sjaak Brinkkemper. Requirements elicitation via fit-gap analysis: A view through the grounded theory lens. In *International Conference on Advanced Information Systems Engineering*, pages 363–380. Springer, 2021.

[45] Alessio Ferrari, Paola Spoletini, and Stefania Gnesi. Ambiguity and tacit knowledge in requirements elicitation interviews. *Requirements Engineering*, 21(3):333–355, 2016.

[46] Muneera Bano, Didar Zowghi, Alessio Ferrari, Paola Spoletini, and Beatrice Donati. Teaching requirements elicitation interviews: an empirical study of learning from mistakes. *Requirements Engineering*, 24(3):259–289, 2019.

[47] Fangfei Lin and Hao Chen. Comparative study of requirements traceability in facing requirements change: Systematic literature study and survey. 2019.

[48] Ahmed Mubark Alsalemi and Eng-Thiam Yeoh. A survey on product backlog change management and requirement traceability in agile (scrum). In *2015 9th Malaysian Software Engineering Conference (MySEC)*, pages 189–194. IEEE, 2015.

[49] Gerald Kotonya and Ian Sommerville. *Requirements engineering: processes and techniques*. Wiley Publishing, 1998.

[50] Andrew Kannenberg and Hossein Saiedian. Why software requirements traceability remains a challenge. *CrossTalk The Journal of Defense Software Engineering*, 22(5): 14–19, 2009.

[51] Craig L Williams. A many-to-many (m: N) relational model to improve requirements traceability. 2011.

[52] Soo Ling Lim and Anthony Finkelstein. Anticipating change in requirements engineering. In *Relating Software Requirements and Architectures*, pages 17–34. Springer, 2011.

[53] Capers Jones. Strategies for managing requirements creep. *Computer*, 29(6):92–94, 1996.

[54] Muneera Bano, Salma Imtiaz, Naveed Ikram, Mahmood Niazi, and Muhammad Usman. Causes of requirement change-a systematic literature review. 2012.

[55] Johannes Ferdinand Hoorn. Software requirements: Update, upgrade, redesign: Towards a theory of requirements change. 2006.

[56] Christof Ebert and Jozef De Man. Requirements uncertainty: influencing factors and concrete improvements. In *Proceedings of the 27th international conference on Software engineering*, pages 553–560, 2005.

[57] Hussin Ahmed, Azham Hussain, and Fauziah Baharom. Current challenges of requirement change management. *Journal of Telecommunication, Electronic and Computer Engineering*, 8(10):173–176, 2016.

[58] Sharon McGee and Des Greer. A software requirements change source taxonomy. In *2009 Fourth International Conference on Software Engineering Advances*, pages 51–58. IEEE, 2009.

[59] Ateeqa Naseer and Muhammad Shoaib Farooq. Exploring causes of requirement change. 2014.

[60] W Lam, Martin Loomes, and V Shankararaman. Managing requirements change using metrics and action planning. In *Proceedings of the Third European Conference on Software Maintenance and Reengineering (Cat. No. PR00090)*, pages 122–128. IEEE, 1999.

[61] W Lam and V Shankararaman. Requirements change: a dissection of management issues. In *Proceedings 25th EUROMICRO Conference. Informatics: Theory and Practice for the New Millennium*, volume 2, pages 244–251. IEEE, 1999.

[62] *The Cambridge Dictionary*. Cambridge University Press, 1999.

[63] Jane Huffman Hayes, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. Prereqir: Recovering pre-requirements via cluster analysis. In *2008 15th Working Conference on Reverse Engineering*, pages 165–174. IEEE, 2008.

[64] Sharon McGee and Des Greer. Software requirements change taxonomy: Evaluation by case study. In *2011 IEEE 19th International Requirements Engineering Conference*, pages 25–34. IEEE, 2011.

[65] Fizor. URL `https://fizor.com/`.

[66] Low-code: Betty blocks. URL `https://www.bettyblocks.com/low-code-application-development`.

[67] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, 1993.

[68] Han Jiawei, Kamber Micheline, and Pei Jian. *Data Mining: Concepts and Techniques.-3rd*. Morgan kaufmann, 2012.

[69] Erna Hikmawati, Nur Ulfa Maulidevi, and Kridanto Surendro. Minimum threshold determination method based on dataset characteristics in association rule mining. *Journal of Big Data*, 8:1–17, 2021.

[70] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Introduction to data mining. ed. *Addison-Wesley Longman Publishing Co., Inc.*, 2005.

[71] Nicole Fabian-Weber. When do baby teeth fall out and in what order?, Dec 2022. URL `https://www.care.com/c/losing-baby-teeth-age-and-order/`.

[72] Klaas-Jan Stol and Brian Fitzgerald. The abc of software engineering research. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 27(3):1–51, 2018.