

UTRECHT UNIVERSITY

MASTER THESIS

**Extracting high-quality end-user
requirements via a chatbot
elicitation assistant**

Author
Marc Valkenier

Supervisors
Dr. Fabiano Dalpiaz (1st)
Prof. dr. Kees van Deemter (2nd)



Utrecht University

Business Informatics
Department of Information and Computing Sciences
Faculty of Science
Utrecht University

January 23, 2020

Abstract

Requirements elicitation can be done using several methods, depending on what segment of users one would want to target. This thesis studies whether a chatbot can be used to elicit high-quality end-user requirements. This thesis introduces ReqBot, a prototype chatbot elicitation assistant; we investigate ReqBot's impact on the quality of requirements and its usability. Quality is represented by referential ambiguity and vagueness, a subset of ambiguity, which in turn is a quality metric for requirements. ReqBot aims to detect these ambiguities and resolve them through interaction with the user. ReqBot is tested against a Google Form, a representation of a current method of end-user requirement elicitation. While both gained a high usability score, possibly due to the simplicity of the user input interface, there was no significant difference in usability between the two. ReqBot was able to detect and resolve ambiguities with a decent recall, however, it still lacks in precision.

Key words— Requirements engineering, end-user requirements, chatbot, quality of requirements, ambiguity, referential ambiguity, vagueness, usability

Acknowledgements

This thesis has been a large and all-encompassing project for me and was not always as easy. There are some people I want to thank for their help during and with this thesis. First and foremost, Fabiano, for all your help and advice guiding me through the thesis. The end result and the research would not have been the same without your help. I also want to thank my friends and family to cheer me up when I got stuck and helped me stay motivated. Finally, I also want to thank everyone who took the time out of their day to participate in my experiment.

Contents

1	Introduction	1
2	Research Methodology	4
3	Literature	6
3.1	Frameworks for quality of requirements	6
3.1.1	Frameworks in literature	7
3.1.2	Selecting a requirements quality framework	10
3.1.3	Criteria of the QUS framework	11
3.1.4	Selecting a criterion	12
3.2	Ambiguity	14
3.2.1	Types of ambiguity	15
3.2.2	Selecting type(s) of ambiguity	18
3.2.3	Tools for detecting ambiguity	19
3.3	Chatbot	24
3.3.1	Introduction to chatbots	24
3.3.2	Classification of chatbots	24
4	Artifacts description	26
4.1	Development of ReqBot	26
4.2	ReqBot under the bot taxonomy lenses	28
4.3	Functionality of ReqBot	33
4.4	Other artifacts	38
4.4.1	Requirements form	38
4.4.2	Scenarios	39
5	Experiment and results	41
5.1	Experiment	41
5.1.1	Set-up	41
5.1.2	Experiment procedure	41
5.1.3	Participants	42
5.2	Results	43
5.2.1	Collected outputs	43
5.2.2	Usability	43
5.2.3	Correctness	47
5.2.4	Quality	50
6	Conclusion	51
6.1	Conclusion of sub-questions	51
6.2	Conclusion of main research question	53
7	Discussion	54
7.1	Limitations and threats	54
7.2	Future work	55

A	Considered platforms and frameworks for the chatbot artifact	63
B	Complete BPMN of ReqBot	64
C	All dialog	65
D	Scenarios	69
E	Experiment - scenario and condition planning	74
F	Experiment approach	76
G	Experiment introduction	77
H	Results - significance values	79

1 Introduction

Stakeholders in a software company, from management to product owners to developers, want to create systems that fit their users' needs best. This increases the chance for users to stay satisfied with and loyal to the system. These end-users have wishes, desires and expectations of a system. Their ideas can become the building blocks for requirements, and are gathered through several techniques such as brainstorm sessions, interviews, focus groups and user observation [56, 75]. Most existing techniques fail to reach a large number of end-users. It would, for example, be practically impossible to interview all end-users of a system used by thousands or millions. It is more practical to focus on representatives of the end-users, which is the case with a focus group [38]. This type of techniques is widely used and accepted but they inherently miss requirements from end-users since not all of them were a part of the elicitation process. This, in turn, results in the possibility that the system may not fit all the needs of its end-users.

There are commonly used approaches to tackle this problem, including feedback forums, review sections, app reviews, feature requests and user forms [51, 35]. However, it can be hard to get high-quality requirements from these unidirectional types of input [32]. In classic elicitation techniques, an expert analyst collects and organizes the requirements using conventional notations that express the key traits of requirements. End-users, on the other hand, are not expert requirements analysts and their formulated requirements do not adhere to standards and conventions. As a response to this limitation, some authors have proposed automated approaches to extract the key traits of requirements, e.g., from app reviews [33].

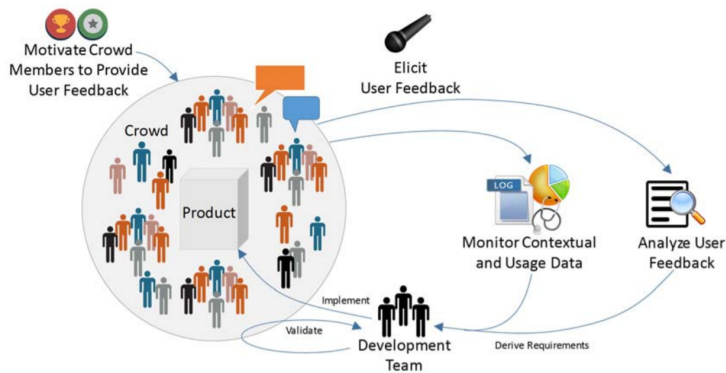


Figure. 1: Aspects of CrowdRE (from Groen et al. [32])

The research area which tries to incorporate these end-user requirements is called CrowdRE (or crowd-based requirements engineering). CrowdRE elicits feedback provided by the earlier mentioned channels and other sources if possible. This feedback can be analysed with various techniques, such as text mining for linguistic feedback. Combined with monitoring and usage data, this should yield new sets of requirements as visualised in Figure 1 [32]. A disadvantage of this general approach is that feedback elicitation and analysis are two separate phases in requirements engineering and by extension also in CrowdRE. This makes it difficult to gain more information on the given feedback since the interaction is completed after the feedback has been sent. This could be information that was left out of the feedback but would be essential to craft a high-quality requirement and satisfy the end-user.

Chatbots can be used for a wide range of goals, including answering frequently asked questions, assisting in online shopping and other assisting or knowledge based functions. Such a chatbot is essentially “a conversational agent that interacts with users in a certain domain or on a certain topic with natural language sentences” [36]. In the area of requirements engineering, the application of a chatbot is new and only a few prototypes are currently available, including CORDULA [27].

The popularity of chatbots has been increasing over the last few years as illustrated by Figure 2. This surge in popularity is caused by the increased popularity of messaging platforms and the advancements in AI and machine learning [34]. Through the advancements in AI and also natural language processing (NLP), the user queries can be recognised better and answered accordingly. Another factor is the indication that text communication has become a socially acceptable form of personal interaction [29], which goes hand in hand with the increased popularity of messaging platforms. In this thesis, we aim to reap the benefits of this quickly advancing technology and study the extent to which chatbots can be used as an assistant to elicit requirements.

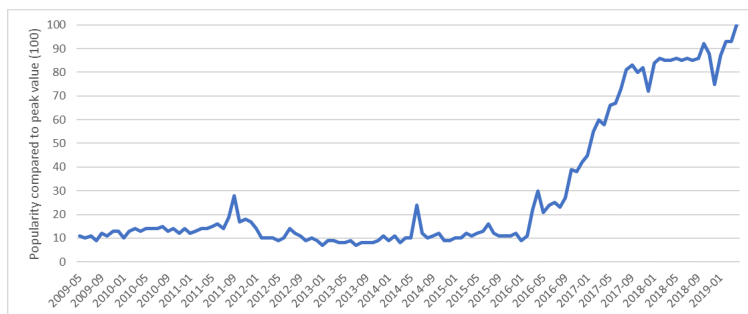


Figure. 2: Google Trends graph showing how the popularity of the term Chatbot has changed over the past 10 years [5])

While CORDULA showed that it is possible to create a chatbot for eliciting requirements, there are still several unanswered questions in the combined field of requirements engineering and chatbots. Can a chatbot help to create high-quality requirements? How could it even be determined that one of those resulting requirements has a high-quality? Would the chatbot affect usability compared to currently applied techniques? To gain a better grasp on this combined field and further explore it, the main research question (MRQ) we put forward is the following:

MRQ To what degree could a requirements elicitation chatbot improve the quality of end-user requirements and the experience of end-users while they express these requirements for a software product?

To structure the answering process of the MRQ, several supporting research questions are formed. As the quality of a requirement can be a subjective concept, it is first and foremost required to specify quality to a degree that becomes measurable. To explore this, the first sub-question of this thesis is:

SQ1 What measures determine the quality of a requirement?

With this information it will also be possible to focus on one specific measure in depth, leading to the second research question:

SQ2 What techniques can be used to automate the detection of defects in a requirement for a selected measure, as defined in SQ1?

This research question is the first half of the equation on the road to improve a specific quality measure of requirements, as the defects in a requirement can only be resolved if they are detected. The second half is part of the next research question:

SQ3 How to design and construct a requirements elicitation chatbot for improving the quality of requirements?

With the creation of a chatbot that aims to improve the quality of requirements by solving defects of a specific measure of certain requirements, the question remains how well it works in practice, leading to the last research question:

SQ4 How effective is the chatbot, as created in SQ3, in eliciting high-quality requirements?

The effectiveness will be measured from both a quality and usability aspect. The quality aspect will be measured by the defects of the selected measure that get resolved.

2 Research Methodology

According to Wieringa, the two main types of research in design science are design problems and knowledge questions [71]. A research concerns a design problem if the design or redesign of an artifact aims to contribute to achieve a goal. The proposed template for such a problem is: Improve <a problem context> by <designing an artifact> that satisfies <some requirements> in order to <help stakeholder achieve some goals>. A knowledge question requests knowledge about the real world as it is. Knowledge questions can be divided into explanatory questions, asking why something happened and descriptive questions, asking what happened. The main difference between a knowledge question and a design problem is that a design problem aims to make changes in the world, while a knowledge question observes the world without making changes. This thesis mainly concerns design problem.

The design cycle is a subset of the engineering cycle and consists of **problem investigation**, **treatment design** and **treatment validation**. Each phase can have one or multiple methods in it. These methods depend among others on the research and the availability of sources.

This research started with the problem investigation phase, which aimed to answer *SQ1* and a part of *SQ2*. To answer *SQ1* and thus find measures to quantify the quality of a requirement, an iterative literature study was conducted on quality measures of requirements. This literature study aims to find a measure by starting at a high level framework for quality and then iterative zooming in on a specific aspect until one satisfying measure was found. This phase also focused on solving *SQ2* by studying the current tooling and research on detecting defects in a requirement concerning the measure from *SQ1*.

The treatment design phase concerns both *SQ2* and *SQ3*. The first of these focuses on getting familiar with the practical implications on detecting these defects in requirements and the latter with designing and building a prototype chatbot.

The treatment validation phase is used to answer *SQ4* and thus determine the effectiveness of the chatbot. The prototype chatbot, created in *SQ3*, is validated in an experiment with participants researching both usability and quality. It is necessary to keep in mind that by introducing the chatbot, the quality of a requirement can be influenced by other less intended factors, as visualised in Figure 3. By introducing a chatbot, the user experience and general usability of requirement elicitation can change. This can have an effect on the quality of a requirement even without trying to resolve a detected defect. The quality of the defect detection also needs to be addressed, since resolving defects can not be done without first detecting the defects. These factors need to be taken into account to provide the most rounded results.

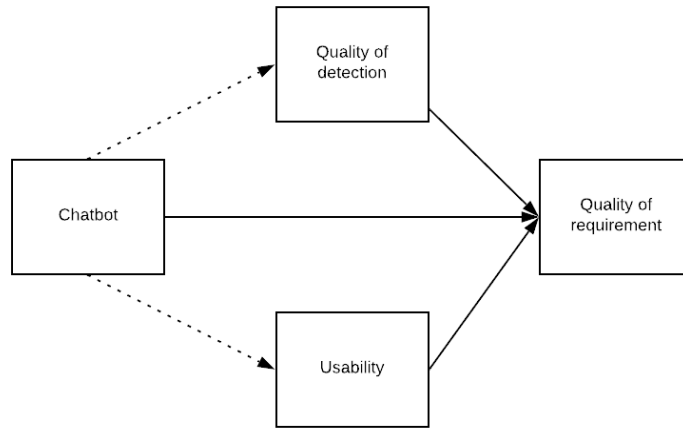


Figure. 3: Factors that can influence the measured quality

3 Literature

This chapter concerns the literature study of this thesis. The literature study is divided into three parts. First, in Section 3.1, different requirement quality frameworks are described and evaluated, whereafter one is picked and further investigated. From this framework, one measure is chosen to focus on. Second, the chosen measure, ambiguity, is explored in Section 3.2. This includes the different types of ambiguity and the existing tools for detection. Last, the current state of chatbots is explored in Section 3.3, providing insight in what a chatbot is and how one can be classified.

3.1 Frameworks for quality of requirements

To improve or measure improvement on the quality of requirements, a deeper understanding of what determines the quality of a requirement is needed. A first step in this can be achieved by following a specific quality framework, consisting of quality metrics. Such a framework should describe all quality aspects of a requirement. With more knowledge on all quality aspects of a requirement, it becomes possible to build or discover measurements for these aspects. This section will evaluate different frameworks for the quality of a requirement and choose the most useful. In Section 3.1.3, the criteria of this selected framework are evaluated and the best one will be selected in Section 3.1.4. Of this criterion, all types are evaluated, selecting the best ones for this research. By zooming in on quality in this manner, the research is focused on a few specific quality measures in depth instead of quality in a broad manner as visualised in Figure 4. The term framework will be taken as a broad overarching term in these sections since most of the lists of quality aspects on requirements were not specifically defined as a framework.

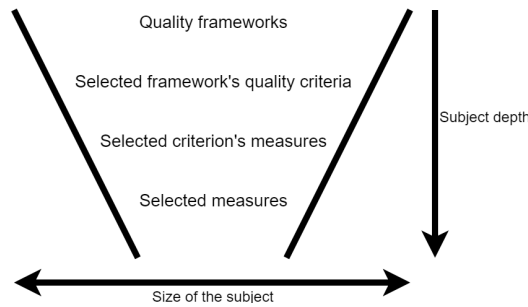


Figure. 4: Literature approach on selecting a measure for quality

3.1.1 Frameworks in literature

In the 1990s, several frameworks that can be used to determine the quality of requirements were created. Two prominent ones were the three dimensions by Pohl [57] and the framework of Lindland, Sindre and Solvberg for the quality of conceptual models [47]. First, the three dimensions framework is based on the idea of an initial input and a desired output. This output can be improved along three dimensions: specification, representation and agreement and is visualised in Figure 5 [57].

- The **specification** dimension starts at its lowest at opaque, moves to fair and ultimately to complete. This dimension represents the degree of understanding of a requirement at a certain time.
- The **representation** dimension begins at informal and is at its maximum at formal, with informal in between. This dimension deals with the manner in which the requirements are represented. A requirement could, for example, be represented as text in natural language (informal), ER-diagrams (semi-formal) or via a formal specification language.
- The **agreement** dimension, moves from a personal view up to a common view. It is tightly linked with the specification as it deals with the degree of agreement on the specification.

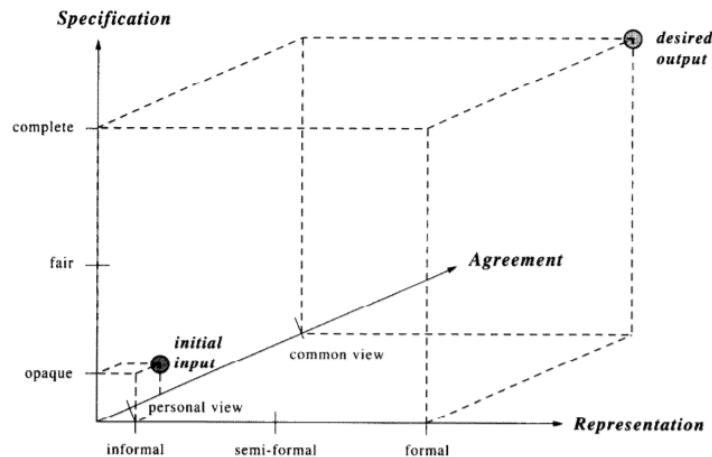


Figure. 5: Three dimensions of RE by Pohl [57]

In 2013, Pohl and Ulfat-Bunyadi evaluated the three dimensions framework and renamed the specification dimension to the content dimension and the representation dimension to the documentation dimension. Further, no large adaptations were made while it was used. However, this framework was not specifically

for the quality of requirements [58]. Since the framework is essentially quite simple, it will not be enough to base metrics of. It is however due to its simplicity a good tool to assist in explanations considering the quality of requirements.

The framework of Lindland, Sindre and Solvberg (shortened to the framework of Lindland) is aimed at conceptual models. However, it can also be used to determine the quality of a requirement represented as a conceptual model. This framework also focuses on three dimensions: syntax, semantics and pragmatics. These dimensions are the result of the comparison of sets of statements. There are four statements:

- The **model** is the set of statements that were made.
- The **language** consists of all possible statements possible in the syntax.
- The **domain** has all possible statements that are relevant as well as correct.
- The **audience interpretation** is the set of statements that the model contains according to the audience.

The degree of similarity between the model and the language results in the syntax dimension, the model and the domain results in the semantic dimension and the model and the audience interpretation results in the pragmatic dimension [47].

Krogsti et al. compared the framework of Lindland with the three dimensions framework and remarked that they are similar in the deeper structures but differ on the surface. The dimensions of both frameworks overlap as can be seen in the syntactic dimension, which corresponds with the representation dimension. Krogsti et al. argued that both these dimensions deal with the relation between the specification and the used language and thus overlap. However, he also noted that they differ in that the three dimensions framework discusses several languages and saw formal specification as a goal, while Lindland's framework does not consider multiple languages and sees formality only as a means. According to Krogsti, the semantic dimension corresponds with the specification dimension as both are concerned with the goal of completion. The difference is that Lindland's framework also includes validity and feasibility, whereas the three dimensions framework does not [41].

In the same time period, Rosenberg et al. proposed a structurally different set of metrics than Lindland and Pohl. While Rosenberg et al. did not call it a framework, he provided a set of metrics to determine quality. For the Automated Requirement Measurement (ARM) tool, a tool that scans software requirement specification documents, seven measures were constructed. These measures could indicate possible problems in requirements. The measures are lines of text (i), imperatives (ii), continuances (iii), directives (iv), weak phrases (v), incomplete (vi) and option (vii). Rosenberg et al. concluded that with this

tool, applying these measures, it is possible to “point out requirements that may be ambiguous or otherwise poorly worded and thus subject to testing problems” [59].

Ali defined a larger set of metrics for a quality requirements document in 2006 [11]. A division was made between so-called internal and external characteristics. Internal characteristics define how requirements should be specified, while external characteristics describe the outer appearance of a requirements document. According to the internal characteristics, requirements need to be: unambiguous, correct, complete, understandable, verifiable, internal consistent, traced, traceable, modifiable, annotated by relative importance, annotated by relative stability, annotated by version, not redundant, at right level of detail, precise and organised. These characteristics were probably based on work by Davis et al., who defined 24 qualities of a software requirement specification [24]. This larger set of characteristics or qualities of requirements can be used to grasp what constitutes a quality requirement. These frameworks are aimed at complete software documents, but also contain metrics that can be applied to individual requirements.

Lucassen et al. proposed a framework to determine the quality of user stories: the quality user story framework (QUS) [49]. The framework was created in the context of agile RE and is intended for the creation of high-quality user stories. It consists of 14 quality criteria, which are divided into three groups, similar to the three views of Lindland: syntactic, semantic and pragmatic. In a later paper, Lucassen reworked the framework to encompass the 13 criteria, as can be seen in Figure 6. In the latter paper, the quality of explicit dependencies has been removed and the term scalable has been changed to estimatable.



Figure. 6: Quality user story framework by Lucassen et al. [50]

3.1.2 Selecting a requirements quality framework

To select a valid and usable framework for this research, several requirements were created to evaluate the frameworks by:

- The framework needs to provide a sufficient level of detail on the measures or aspects it presents for determining the quality of a requirement. This includes a description of the aspect and forces aspects to be specific.
- The aspects presented by the framework should be measurable to a reasonable degree. This ensures that it is possible to have a measurable outcome later in this research.
- Both the novelty of a framework and the times it is applied in other research should be considered. An older framework has had more time to be established, used and verified compared to a new one, which can make it a more accepted framework. However, research in the field of RE is an ongoing process in which newer research and thus newer frameworks can be developed with the knowledge of older frameworks combined with the research done in the time between them.
- While not all previous mentioned frameworks were set out to be frameworks or models, it can be useful to use one of these which was intended as a framework.

The frameworks have been compared along with all previously mentioned requirements. By process of elimination, the three dimensions and Lindland's framework were dropped due to the measurability and the level of detail in the models. While both frameworks are usable in their own way and are also well established, they were too high level to be used in this research. The framework for the ARM tool by Rosenberg dropped off due to it being too focused on one tool and having very specific but not usable measures. This narrowed the selection down to the QUS framework and the 24 qualities by Davis. The QUS framework was selected as the framework to be continued to be used in this research since it is designed as a framework and it made a clear distinction between quality measures for individual requirements and sets of requirements.

3.1.3 Criteria of the QUS framework

The QUS framework consists of 13 criteria, these need to be further explored to gain insight in which of these could be best suited for improvement by a chatbot. To gain this insight, the literature on or related to the criteria was researched. The 13 criteria are described below, with their definitions according to Lucassen et al. [50].

Atomic: "A user story expresses a requirement for exactly one feature". Liskin et. al researched the granularity of user stories. With a focus on Expected Implementation Duration or EID, a survey was conducted yielding 72 results. It revealed that in general more coarse (larger/longer) user stories resulted in more problems, such as grow unexpectedly in EID. It was stated that "smaller user stories are perceived as more tangible and predictable, they help to keep feedback-cycles short, and they entail less post-development changes" [48].

Minimal: "A user story contains nothing more than role, means, and ends". In the Agile manifesto, user stories were defined as having three parts. These are a part of the question on who, what and why [16]. Wautelet et al. researched the different templates in which the user story format has been used. For the 'who-part', the term 'role' is most often used. Several different terms are used for the parts on what and why. These include goal, action or feature for what and business value, benefit or reason for why [70]. There were no additional parts that could expand a user story, showing that this is the maximal size.

Well-formed: "A user story includes at least a role and a means". Leffingwell states that user stories should contain at least a role and means [45]. With one of these parts missing, the user story would lose its value since it would either miss what needs to be done or from which perspective. This is crucial information, while the ends or reason is not necessarily.

Conceptually sound: "The means expresses a feature and the ends expresses a rationale".

Problem-oriented “A user story only specifies the problem, not the solution to it”.

Unambiguous: “A user story avoids terms or abstractions that lead to multiple interpretations”. Different types of ambiguity exist, including lexical ambiguity, syntactic ambiguity, semantic ambiguity, pragmatic ambiguity, vagueness, generality and language errors [18]. Different tools exist to find and reduce ambiguity, such as Smella, which works using so-called bad smells to find ambiguity [25].

Conflict-free: “A user story should not be inconsistent with any other user story”. Sommerville et al. noted that conflicting requirements exist and failure to detect and resolve them will lead to rework [65]. Kim et al. divides requirement conflicts into activity conflicts and resource conflicts. They also propose a quite general approach of 4 phases to detect and manage these conflicts: requirements authoring, partitioning, conflicts detection and conflicts management.

Full sentence: “A user story is a well-formed full sentence”.

Estimatable: “A story does not denote a coarse-grained requirement that is difficult to plan and prioritize”. Different techniques exist to estimate user stories. These are among others triangulation, planning poker and voting. Miranda proposes a paired comparison method to reduce the number of comparisons [53, 52].

Unique: “Every user story is unique, duplicates are avoided”. If duplicates exist and go undetected, it would result in a rework of the software feature according to Barbosa et al. Furthermore, they introduced an approach to detect possible duplicates [13]. With duplicates resulting in rework, the importance of a user story being unique is exemplified.

Uniform: “All user stories in a specification employ the same template”. There are several manners to write a user story as Wautelet et al. showed [70]. However, within one organisation it is important to use a specific template. If different people were to apply different templates within one team, the result might be that misinterpretations occur.

Independent: “The user story is self-contained and has no inherent dependencies on other stories”.

Complete: “Implementing a set of user stories creates a feature-complete application, no steps are missing”.

3.1.4 Selecting a criterion

The selection of a criterion is based on several requirements.

- Given the exploratory nature of this research, the focus will be on improving **individual requirements** as opposed to sets of requirements. Therefore, it would be preferable to use a criterion that concerns individual requirements;

- The **relevance** of the criterion in the context of a chatbot is important. A chatbot has interaction with a user and this should be exploited. A criterion that can be improved without interaction is thus less interesting;
- The **severity of the consequences** if the criterion is left unchecked and the gain if the criterion is positively influenced;
- The **achievability of implementation** should be taken in mind. The chosen criterion should have the possibility of being implemented within the time-span of this research.

Considering these four requirements, the criteria complete, independent, uniform, unique and conflict-free are not suitable candidates since these focus on (a part of) the entire set of requirements. For example, end-user requirements do not need to be unique as many users can have the same or a similar requirement. This could, in turn, be used to prioritise requirements. The criteria full sentence, atomic, minimal and problem-oriented are excluded on the base of relevance in the context of a chatbot. These criteria are on ways of writing requirements, that would not require interaction with the user to improve. Moreover, the end-users communicate to the chatbot by writing in natural language and not a predetermined format. All of these can also later be fixed by a requirements engineer examining the requirements. For instance, a requirements engineer can rework a user story that is not problem-oriented to one that is without having to interact with the user by rephrasing or removing the solution part.

By method of elimination, four criteria are left:

- Well-formed
- Conceptually sound
- Unambiguous
- Estimatable

Conceptually sound requires a semantic understanding of what is expressed in a user story, in short, it would require the chatbot to understand the meaning of what is written. The suspected achievability of this is considered as too hard for this research. The criterion estimatable could be positively influenced by asking the user how important the requirement is, gaining information on prioritisation. However, if this is not implemented, the consequences might not be too severe since end-user requirements are generally processed by a product owner or a requirement engineer. They can decide whether it is estimatable in its current size or if it needs to be divided into smaller parts. They should be able to do this to a degree, even with sub-optimal requirements. This criterion is thus excluded based on the fact that the suspected gain of another criterion would be greater.

A user story should always have at least a role, a means and an ends. This can be achieved by asking the correct questions for each requirement but might be a bit too basic to have a large impact on the quality on its own. Unambiguous on the other hand can require interaction with the end-user. If for example, a requirements engineer has an ambiguous user story, it can in cases be impossible to determine the meaning the end-user wanted to give to that user story. Thus direct interaction with the user could prevent ambiguity, by asking the user the meaning of the ambiguous part. Furthermore, if ambiguity is left unchecked, the results could be severe. As ambiguity means that a (part of a) user story has multiple interpretations. This could result in an implementation which does not conform to the intended meaning of the user story. As unambiguous can be used on individual user stories, has severe results if it is left unchecked and can require interaction to be improved, **unambiguous** is chosen as the criterion for this research.

3.2 Ambiguity

Ambiguity is defined as “*the quality of being open to more than one interpretation*” by the Oxford Dictionary [2] and as “*the fact of something having more than one possible meaning and therefore possibly causing confusion*” by the Cambridge Dictionary [6]. While both definitions share the part on something having multiple meanings or interpretations, the latter addresses the possibility that this, in turn, can cause confusion. This immediately illustrates the risk ambiguity poses to requirements since this confusion needs to be resolved, which can cost time, money or both.

Ambiguity can be either intentional or unintentional. Under the assumption that one does not specifically intend to be ambiguous while writing requirements, the main concern is unintentional ambiguity. However, whether a sentence or a requirement is communicated with the same meaning also depends on the writer and the reader of it. As language contains a large set of rules, not every rule will be understood by everyone. If two people know a rule and both write or read a sentence following that rule, the meaning of that sentence will be clear for both. In case they both apply the rule similarly wrong, the meaning of the sentence could still be clear. But if one of them does apply the rule, while the other does not, they could have a miscommunication [18]. Take for example the user story:

As a user, I do not want no scroll-bar on the homepage.

This user story contains a double negative. A reader would be unaware whether the writer did this on purpose and meant that he does not want the scroll-bar to be absent or that it was an error and the writer meant that he does not want a scroll-bar. While this sentence should have only one meaning, it is ambiguous since the knowledge of the writer is unknown and thus could have another meaning.

3.2.1 Types of ambiguity

Ambiguity as a whole encompasses several distinctly different types of ambiguity. Berry et al. defined them as lexical, syntactical, semantic and pragmatic ambiguity. These were supplemented with vagueness, generality and language errors. These types are listed below with the definitions provided by Berry et al. [18]. Most of these types are comprised of sub-types, as can be seen in Figure 7.

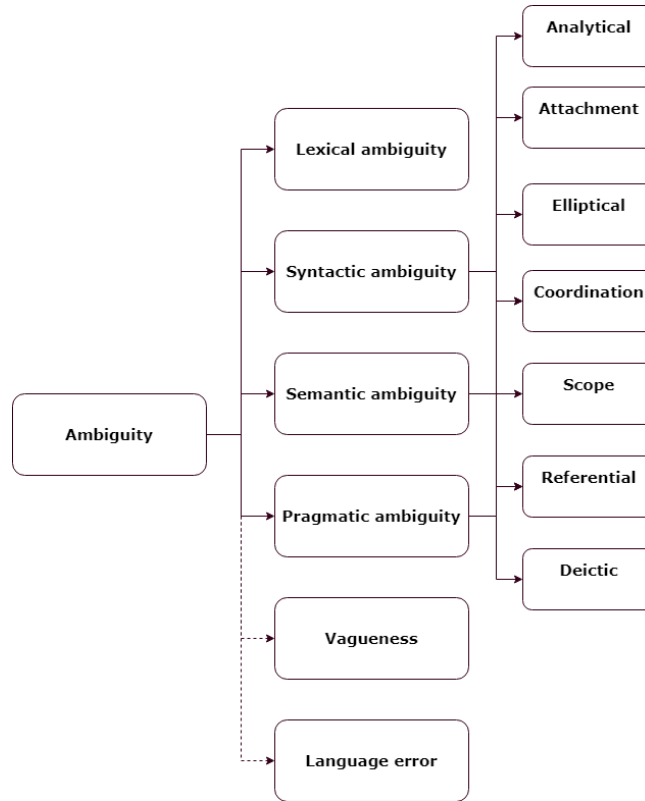


Figure. 7: Types of ambiguity, based on Berry et al. [18]

Lexical ambiguity “occurs when a word has several meanings”. While many words in the English language have multiple meanings, it is often possible to determine the meaning of such an ambiguous word with the help of contextual cues [61]. For example, nails can refer to fingernails or nails that you should hit with a hammer, but through the context of the rest of a sentence, the meaning can become clear. E.g. ‘I cut my nails because they grew too long’ makes it clear that fingernails are meant. However, this is not always the case, as ‘I broke a nail’ does not clarify what type of nails is meant. Lexical ambiguity can be divided in *homonymy* and *polysemy*. Homonymous words have multiple

meanings, but these are unrelated to one another, while polysemous words have related meanings [17].

Syntactic ambiguity “occurs when a given sequence of words can be given more than one grammatical structure, and each has a different meaning”. Berry et al. divided this type of ambiguity in analytical, attachment, coordination and elliptical ambiguity. Of these, coordination ambiguity is closely related and essentially also part of semantic ambiguity.

Analytical ambiguity “occurs when the role of the constituents within a phrase or sentence is ambiguous”. The definition of Berry et al. is substantiated by Fuchs et al. as they claim that analytical ambiguity “arises when the type of a constituent is undecidable” [28]. Both define it as an ambiguity that occurs when there is uncertainty on the role or the type of a constituent. An example of this would be *the German teacher*, which can be interpreted as a German that teaches a random course or a teacher of the German language.

Attachment ambiguity “occurs when a particular syntactic constituent of a sentence, such as a prepositional phrase or a relative clause, can be legally attached to two parts of a sentence”. Zhao and Lin further divide attachment ambiguity further in three categories: prepositional or PP-attachment ambiguity, relative clause ambiguity and pre-nominal modifier ambiguity [76].

Coordination ambiguity occurs “when more than one conjunction, and or, is used in a sentence or when one conjunction is used with a modifier”. The possible harm such an ambiguity can cause is stressed by Agarwal and Boggess as they state that “If a natural language understanding system fails to recognise the correct conjuncts, it is likely to misinterpret the sentence or to lose its meaning entirely” [10]. In this case, the term ambiguity is not used but the consequences overlap with the definition of an ambiguity: there are multiple interpretations with a chance of misinterpretation. Brouwer et al. divided coordination ambiguity into noun phrase and sentence coordination ambiguity [21]. This was in turn based upon research from Frazier, who researched how long it takes to disambiguate or process sentences. Frazier did however not specifically classify these as a coordination ambiguity [26].

Elliptical ambiguity “occurs when it is not certain whether or not a sentence contains an ellipsis”. Sellars states on elliptical ambiguity that “in ellipsis the context completes the utterance and enables it to say something which it otherwise would not, different contexts enabling it to say different things” [62]. This overlaps with the definition of Kiyavitskaya et al. of an ellipsis: “Ellipsis is the deliberate omission of some aspect of language form whose meaning can be understood from the context of that form” [37]. An example used by both Kiyavitskaya and Berry to illustrate this type of ambiguity is “*Perot knows a richer man than Trump*”, which can have two meanings. The first meaning is that Perot knows someone who is richer than Trump and in the second case it means that of all the people Trump and Perot know, only Perot knows the richest person. In the second case, it is implied that the sentence should be ‘Perot knows a richer man than Trump *knows*’. Through the omission of the ‘knows’ at the end of the sentence, an elliptical ambiguity is created.

Semantic ambiguity “occurs when a sentence has more than one way of reading it within its context although it contains no lexical or structural ambiguity”. It contains scope, coordination and referential ambiguity. Coordination and referential ambiguity overlap with syntactic and pragmatic ambiguity respectively and are thus discussed in those sections.

Scope ambiguity occurs when quantifier operators and negation operators can enter into different scoping relations with other sentence constituents. An example of this is *some people buy a Toyota*. This can mean that there are multiple people buying multiple cars by Toyota or that a group of people buy one Toyota together. In this case, ambiguity is created by two quantifier operators (some and a). The same can occur with a negation and a quantifier operator.

Pragmatic ambiguity “occurs when a sentence has several meanings in the context in which it is uttered”. It encompasses referential and deictic ambiguity.

Referential ambiguity “occurs when an anaphor can take its reference from more than one element, each playing the role of the antecedent”. This definition has a similar basis with the one used by Nieuwland and van Berkum: “Referential ambiguity arises whenever readers or listeners are unable to select a unique referent for a linguistic expression out of multiple candidates” [54]. Their research concludes that readers make an inference to evaluate the referential candidate if they resolve the ambiguity. Though this only happens when both candidates have equally plausible antecedents.

Deictic ambiguity “occurs when pronouns, time and place adverbs, such as now and here, and other grammatical features, such as tense, have more than one reference point in the context”. This type of ambiguity is closely related to referential ambiguity since it is based on the same uncertainty of a reference point in the earlier text. The main difference is that referential specifically concerns an anaphor, while deictic focusses on pronouns and adjectives.

Vagueness, “a requirement is vague if it is not clear how to measure whether the requirement is fulfilled or not”. Vagueness with this definition is a term which can include many cases. Opinions can also differ on where the threshold lies for something to be vague. Adjective and adverb vagueness is a form of vagueness that can be seen as slightly more concrete. If an adjective or adverb is not concrete or measurable to a degree, the statement becomes vague. Adjective vagueness can be reduced by avoiding or replacing vague adjectives (such as fast or small) with concrete terms.

Generality, “a general expression can be made more precise”. *My sibling* is a form of generality, as one can have multiple siblings and with just this information it is unclear which one is intended.

Language error “occurs when a grammatical, punctuation, word choice, or other mistake in using the language of discourse leads to text that is interpreted by a receiver as having a meaning other than that intended by the sender”.

3.2.2 Selecting type(s) of ambiguity

Given the types of ambiguity described in the previous paragraph, a decision needs to be made on which type(s) should be used in this research. A viable type of ambiguity should adhere to a set of requirements specific to this research:

- The expected impact of the type of ambiguity on the requirement should be considered. Not all types of ambiguity might cause problems of the same magnitude if they are left in a requirement. This should be viewed from two sides: the negative consequences if an ambiguity of that type remains and the positive rewards the disambiguation of that type of ambiguity would yield.
- The probable practicality of resolving the type of ambiguity should be taken into account. This includes the ability to detect the type of ambiguity and the added value of direct user interaction to remove the type of ambiguity.
- The practical likelihood of a type of ambiguity occurring should be looked at. Types of ambiguity that do probably not happen often in requirements, will be considered as less applicable for this research.
- The time and resource constraints of this research should be kept in mind.

2 types of ambiguity were selected for this research:

- Referential ambiguity
- Adjective and adverb vagueness

Referential ambiguity can cause some confusion if it is not resolved. This type is probably common in a conversation in natural language such as a conversation with a chatbot. A user can refer to a word or a part of a phrase that was entered in an earlier interaction. During the conversation with the chatbot, a requirement will be formed and if referential ambiguity occurs during this conversation, the requirement may not have the meaning intended by the user. There is a practical way to solve this type of ambiguity by finding an anaphor that can take reference from multiple elements. Then the chatbot can ask the user from which of the elements the anaphor should take reference.

Adjective and adverb vagueness is a form of vagueness and can have a significant impact on a requirement. If a user provides a requirement containing e.g. *...I want to be able to send large files...*, the requirement is not usable. The definition of a large file would differ from user to user, making the requirement too vague to be implemented. The detection and disambiguation of this type of ambiguity seem to be fairly doable. There needs to be a list with vague adjectives and adverbs and for each a or several disambiguation questions that can be communicated to the user. This type of ambiguity presumably occurs often in end-user requirements as the vague adjectives and adverbs are easy ways to

communicate a desire without becoming concrete. For the rest of this thesis, vagueness will refer to adjective and adverb vagueness specifically.

The impact of **lexical ambiguity** can vary in different cases, depending on the word and the context. It is, however, a type that can have a straightforward method of detection and disambiguation. E.g. if a word has two distinct meanings, the chatbot could ask the user which of the two meanings he intended, resulting in quick and easy disambiguation. The likelihood of occurrence of this type of ambiguity is deemed relatively high as there are many homonyms in the English language increasing the chance that one can be used unintentionally. However, as many words in the English language can have multiple meanings, it would result in a lot of false hits without an elaborate way of filtering. Lexical ambiguity was thus discarded, but if a third type would be picked, lexical would be the first choice.

The other types of ambiguity are to some extent viable but lack in one or multiple aspects compared to those described above. Language errors can occur in requirements, however, these errors can be hard or impossible to detect. Language errors, such as spelling or grammar errors, can generally be detected by a spell checker. But for a sentence to become ambiguous as the result of a language error, the error needs to cause the sentence to have multiple meanings. To detect these, the chatbot would, in turn, have to be able to determine the meaning of a sentence, which is deemed out of the scope for this research. Analytical ambiguity, elliptical ambiguity and attachment ambiguity could essentially also be chosen, but they were not as there was no concrete plan to automatically detect them. These ambiguity types, along with lexical ambiguity, can be an interesting subject for future work.

3.2.3 Tools for detecting ambiguity

Before ambiguities can be resolved, they first need to be detected. Given the definitions in the previous section, it is possible to find these ambiguities manually in a text or a requirement. For a chatbot to act upon ambiguity with the goal of disambiguation, ambiguities must get detected automatically. Different tools and approaches exist in the literature to detect and sometimes even resolve ambiguities. 20 tools or approaches were evaluated to gain a better understanding in what techniques exist to detect ambiguities and how they can be applied.

Of all tools, Smella is one of the best defined and described. It is a tool developed by Femmer et al. that automatically detects defects in requirements, including ambiguity [25]. It does so by borrowing the concept of bad code smells and applying it to the field of requirements engineering, calling them requirement smells. Femmer et al. give a requirement smell four characteristics:

- A smell has a **indicator** for a quality violation of a requirement
- A smell does **not guarantee a violation** and thus, it always needs to be evaluated with the context in mind

- A smell has within the requirement a **concrete location**
- A specific smell has a specific **concrete detection mechanism** that was used to find the smell.

The smells do thus not indicate defects directly but rather findings that can later be classified as defects. A defect is, in this case, an instance of a quality violation. Smella gets requirements as input and gives a representation of the results as output, including a dashboard and list of all found smells. To do this, it has a 4 steps protocol: requirements parsing, language annotation, language annotation and presentation.

First, the **requirements parsing**, which is a process of getting the requirements from a certain format (e.g. a word document or a CSV file) to a desired format. This format is plain text files, one for each requirement. This is done so that they can be used in the later phases of the process.

Second, **language annotation** is the step of providing meta-information on the requirements. Smella has 3 ways of annotating the language in the requirements: POS tagging, morphological analysis and dictionaries & lemmatization. Part-of-speech (POS) tagging is a way of determining the function of a word in a sentence such as a verb, adjective, pronoun, etc. This technique is common among ambiguity detection tools and approaches. RESI is, for example, a tool that uses POS tagging first to later apply rules to it to find so-called problems, which include forms of ambiguity [39]. SREE also uses POS tagging in combination with parsing in its syntactic analysis [68, 69] and POS tagging is also a method used and mentioned by numerous others [22, 31, 12, 67, 73]. The second method Smella uses for annotation is morphological analysis, which is based on POS tagging. It is a more in-depth analysis in which the inflexion of the word is determined. Morphology is also used by Willis et al. to assist them in predicting coordination ambiguity [72]. Last, Smella uses dictionaries & lemmatization. A lemmatizer normalizes words to produce the original form of the word. E.g 'does', 'did' and 'done' will return to the word 'do' if this lemmatizer is used. These words are later matched against words from pre-defined dictionaries.

Third, the **identification of findings** occurs. In this process, the information from the language annotation is used to identify findings. Depending on the type of smell a different technique was used by Femmer et al. as can be seen in Table 1

Smell name	Detection
Subjective Language	Dictionary
Ambiguous Adverbs and Adjectives	Dictionary
Loopholes	Dictionary
Open-ended, non-verifiable terms	Dictionary
Superlatives	Morphological analysis or POS tagging
Comparatives	Morphological analysis or POS tagging
Negative Statements	POS tagging and dictionary
Vague Pronouns	POS tagging: Substituting pronouns

Table 1: Smella, detection per type smell by Femmer et al. [25]

Last, the **representation of findings**, which occurs in Smella. It is possible to view, review and blacklist findings on a requirement level. The findings are indicated in the requirements in a spell checker style.

The Quality Analyzer for Requirements Specifications (QuARS) is a tool for the analysis of requirements documents. It aims to detect linguistic inaccuracies and defects [42]. QuARS has two main analysis functions:

- Lexical technique: aimed at finding vagueness, subjectivity, optionality and readability.
- Syntactical technique: aimed at finding implicitness, weakness, under-specification and multiplicity.

The methods that QuARS uses are a syntactic parser, a lexical parser, an indicator detector, a view deriver and dictionaries. The syntactic parser works as an early POS tagger as they yield approximately the same output. The lexical parser identifies specific words and is used to support the detection of special terms. The indicator detector identifies the defects based on the output of both parsers and creates a log file which is visualised by the view deriver. The dictionaries are a passive part of QuARS and the contents are used by both analysis and the view deriver. The analysis for both the lexical and the syntactical technique is the same, but the output on the screen of the user will differ depending on the selected method.

The NL2OCL project aims to translate natural language to object constraint language. To accomplish this, the Stanford POS tagger and the Stanford Parser are used. In their paper, Bajwa et al. describe that while the tagger and parser are quite accurate, they can make some errors relevant to the project due to two types of ambiguity [12]. The attachment ambiguity and homonymy, a form of lexical ambiguity causing confusion on the syntactical identification of a word, are identified as the causes.

To address the attachment ambiguity, the project combines the two inputs it requires to function, namely a specification of a constraint and a UML class

model. By evaluating the parse tree with the UML, mistakes in the parse tree caused by the attachment ambiguity can be fixed. As for homonymy, this was fixed by mapping the tagged words to elements of the UML. With these approaches, the project was able to improve the precision and recall of the Stanford parser in the case of the attachment ambiguity and the accuracy concerning homonymy. In addition to these three tools or approaches, many others exist. The essentials of these tools are summarised in Table 2.

Name	Publication	Year	Detects
QuARS	QuARS: A Tool for Analyzing Requirements [42]	2005	Lexical, syntactical
n/a	Automatic identification of nocuous ambiguity [72]	2008	Nocuous ambiguity
n/a	An experimental ambiguity detection tool [60]	2008	Ambiguity in general
SREE	Avoiding ambiguity in requirements specifications [68]	2008	Lexical, syntactic
T1 and T2	Requirements for tools for ambiguity identification and measurement in natural language requirements specifications [37]	2008	Lexical, syntactic
AMBER	The Usability of Ambiguity Detection Methods for Context-Free Grammars [15]	2009	Ambiguity (has mode for ellipsis)
RESI	RESI - A Natural Language Specification Improver [39]	2009	Lexical and coordination among others (not mentioned as these terms)
Ambidexter	Ambidexter: Practical ambiguity detection [14]	2010	Ambiguity in general
n/a	Ambiguity Detection: Towards a Tool Explaining Ambiguity Sources [31]	2010	Lexical, syntactic, semantic, pragmatic, vagueness, language error
NAI	Automatic Detection of Nocuous Coordination Ambiguities in Natural Language Requirements [74]	2010	syntactical
n/a	Analysing anaphoric ambiguity in natural language requirements [73]	2011	Anaphoric ambiguity
n/a	Tool for Automatic Discovery of Ambiguity in Requirements [55]	2012	Lexical, syntactic, syntax
NL2OCL (project name)	Resolving Syntactic Ambiguities in Natural Language Specification of Constraints [12]	2012	Focussed on homonymy and attachment ambiguity

n/a	Quality factor assessment and text summarisation of unambiguous natural language requirements [67]	2013	Ambiguity in general
RQA	A framework to measure and improve the quality of textual requirements [30]	2013	Ambiguity as part of Correctness, Completeness and Consistency
DODT	The DODT tool applied to subsea software [66]	2014	Ambiguity and other metrics
n/a	Resolving Ambiguities in Natural Language Software Requirements: A Comprehensive Survey [63]	2015	Ambiguities
n/a	Detecting Vague Words & Phrases in Requirements Documents in a Multilingual Environment [22]	2017	Vague words and phrases
Smella	Rapid quality assurance with Requirements Smells [25]	2017	Subjective Language, Ambiguous Adverbs and Adjectives, Loopholes, Open-ended non-verifiable terms, Superlatives, Comparatives, Negative Statements, Vague Pronouns, Incomplete References
REVV	Pinpointing Ambiguity and Incompleteness in Requirements Engineering via Information Visualisation and NLP [23]	2018	(near-) synonymy, homonymy, incompleteness

Table 2: An overview of disambiguation tools and approaches

3.3 Chatbot

This section gives an overview of chatbots. It starts with an introduction to chatbots and ends with a framework to classify them. The classification is also used to describe the chatbot created for this thesis in Section 4.2.

3.3.1 Introduction to chatbots

A software bot is “an interface that connects users to services” [44]. Lebeuf remarks that people commonly and incorrectly use the terms software bot and chatbot interchangeably. Chatbots are a domain within all software bots, meaning that a chatbot is not necessarily a chatbot, but every chatbot is a software bot. The chatbots differentiate themselves from the other bots by having a conversational interface [9]. This definition is in line with both definitions from Shawar and Atwell, “a software system, which can interact or chat with a human user in natural language such as English” [64] and the Oxford Dictionary, “a computer program designed to simulate conversation with human users, especially over the internet” [3]. In short, a chatbot is a specific bot that **interacts with human users through conversation**.

3.3.2 Classification of chatbots

There are different ways to classify bots and chatbots depending on what attributes the person making the classification focuses. Chatbots and software bots are for these classifications again used interchangeably since most classifications do not make a distinction between them. The classification can depend on the environment or platform that the chatbots use. So there can be a distinction made between Slack-based bots and Telegram-based bots. Another distinction could be based on intention, namely good bots and bad bots. With good bots containing chatbots, crawlers, transactional bots, informational bots and entertainment bots while bad bots contain hackers, spammers, scrapers and impersonators [9].

Bunardzic classifies chatbots in 4 types [8]:

- **Stateless bots** are bots that do not track a conversation. They process the text send by the user and provide a reply, but there is no conversational end goal and the chatbot does not store information from previous sentences. Most of the chatbots today are this type and they are relatively easy to build
- **Semi-stateful bots** track conversations in a limited capacity. These bots can be compared to an automated phone menu: the choice the user makes earlier defines the options or responses the user can get later in the conversation
- **Stateful bots** track conversations even more than the semi-stateful bots. Where semi-stateful bots track the conversation only during a session,

stateful bots can track a conversation over multiple sessions. This means that the bot should e.g. be able to answer questions on previous conversations with the same user.

- **Loyal bots** are comparable to stateful bots in most ways. The distinction is that stateful bots review the history of interaction upon receiving a message while loyal bots do this on a scheduled basis. This results in the fact that loyal bots should be able to provide more sophisticated services.

Lebeuf et al. proposed a taxonomy for software bots based on the properties, behaviour and environment of a software bot [43]. The goal of this taxonomy is to create a structure to examine and study bots. Three main dimensions were created for the construction of the taxonomy in their research: the environment dimension, the intrinsic dimension and the interaction dimension. These can be seen with their respective facets in Figure 8. The **environment** dimension describes the surroundings in which the bot operates. This can, in turn, have an influence on the behaviour and capabilities of the bot. The **intrinsic** dimension focuses on facets of the bot, describing the properties of the bot, such as its goal. While it is often hard to determine some of these facets without having the knowledge from the developers of the chatbot, all facets are designed so that they can be identified. The **interaction** dimension consists of the different facets on the interaction of a bot with entities in its environment. This can be users but also other systems.

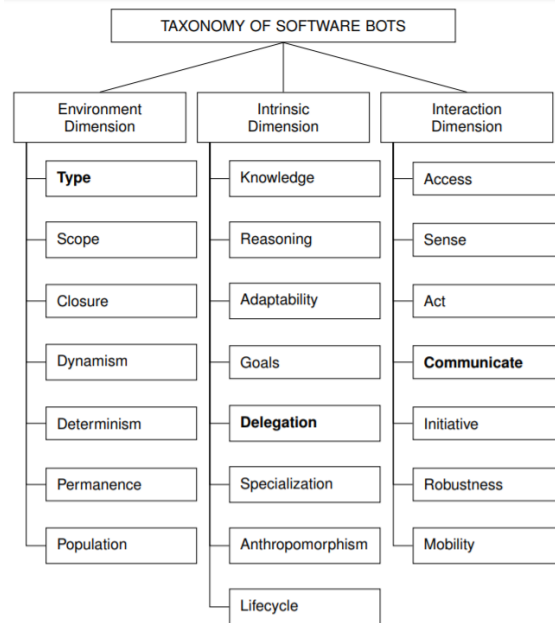


Figure. 8: High-level taxonomy of software bots by Lebeuf et al. from [43]

4 Artifacts description

In this chapter, all artifacts that are used in the experiment are described. The main focus is the major artifact of this thesis, a chatbot. First, the development of this chatbot is described, followed by placing it in a scientific context using the taxonomy described in Section 3.3.2. Finally, the specific functionalities are described and explained. Next to the chatbot, a requirements form complementing the chatbot is presented. Lastly, scenarios are described, which contain requirements that participants should extract in the experiment.

4.1 Development of ReqBot

We, this thesis' supervisor and this thesis' author, created an artifact in the form of a chatbot to answer SQ4, answering SQ2 and SQ3 in the process. The chatbot was later named ReqBot, short for requirements chatbot and will be referred to as such. We designed ReqBot as a proof-of-concept tool to use in the experiment phase of this thesis. ReqBot had to adhere to a few basic requirements, R1 - R5, to fulfill this purpose.

R1: ReqBot has to be able to ask users questions which would yield all information a requirement needs. Going with the user story format this entails the **means**: what a user wants changed or added; the **ends**: why a user wants the means; and the **role**: what type of user the user is to the system.

R2: ReqBot has to be able to **support a conversation**. As the minimum information we would want users to provide already consists of the 3 aforementioned parts, ReqBot has to be able to ask this information in a set of multiple questions. It could be argued that all information (means, ends and role) could be asked in one question. However, this could give an information overload to the user, who could end up answering only a part of that question. After all, is a chatbot that can only ask one question that different from an online form?

R3: ReqBot has to **lead the conversation**. This is necessary as we know what information we need from a user to form a complete requirement. Not every user can be expected to know this information and neither should they if they get asked the right questions. This results in a conversation in which ReqBot asks questions and the user answers questions and by answering these questions the user specifies his or her wishes.

R4: ReqBot has to have the ability to **detect certain types of ambiguity and vagueness**, more specifically referential ambiguity and vagueness for we have chosen these as the best candidates to improve quality, as described in Section 3.2.2.

R5: ReqBot has to be able to **initiate a disambiguation process** on the detected ambiguity or vagueness. This entails an interaction with the user in

which ReqBot asks for clarification which would resolve the initial ambiguity. The aim of this should be to resolve the ambiguity, which would positively impact the quality of the requirement.

With these 5 basic requirements, we started the design process of ReqBot. The first step was to find a platform and create a low-fi prototype. With this, we could test the capabilities of the platform and discover the critical interactions the chatbot would need to have. Most platforms that allowed sequential interactions and thus supported the option for a conversation, had limited options of adding custom code which was primarily needed for the detection of ambiguities. See Appendix A for the tested platforms and frameworks including their strengths and weaknesses. After several iterations, we decided to use the Bot Framework from Microsoft [1]. This framework provides a basis to develop a chatbot. Unlike most of the earlier used platforms, Bot Framework has no drag and drop functionalities to build dialogue so everything has to be programmed by hand. This does allow more freedom in terms of what the bot can do, recall and process. This made Bot Framework the best choice for the development of a chatbot for this thesis project.

Development was done in an iterative manner. During several meetings with the thesis' supervisor, we evaluated the newly created functionalities. During these meetings, we also discussed what functionalities should be added, changed or removed. These meetings continued until the week before the start of the experiment, after which no changes were made.

About halfway through development, we planned sessions with 3 participants to test the current version of the chatbot. The main goal for us was to get a grasp on how people would interact with the chatbot and what the weak points were. The sessions were set-up in a semi-structured manner. We created a scenario of a fictive person who is struggling with a system for these sessions to give the participants some context to use the chatbot in. The scenario starts with a general description of the person using the system and is followed by 5 requirements concealed in the text. More on scenarios can be read in Section 4.4.2 and all complete scenarios are located in Appendix D.

This thesis' author gave the participants a short explanation of the research, the chatbot and the goal of this session. The participants were encouraged to think aloud if he or she would get stuck and to remember both positive and negative aspects of the chatbot. After this, the participants were given the scenario and the chatbot, whereafter he or she would start interacting with the chatbot. The participants were observed while interacting with the chatbot and after the session, there was an unstructured interview on the experience with the chatbot. Using the information gathered in these sessions, we made improvements on the chatbot.

The major improvements were:

- ReqBot initiates the conversation
- A more elaborate explanation of the difference between a new feature and an existing feature (later renamed to *new idea* and *request a change*)
- More diversity in responses from the chatbot
- An indication that the requirement will be used

The chatbot, ReqBot, will be further discussed in the next two sections. First by placing it in the taxonomy of Lebeuf [44]. This taxonomy is made to compare chatbots using a back box approach, meaning they can be classified without having to know the inner structures of the bot. However, having developed the chatbot, we can provide more information on functionality and also on design decisions. This is detailed in the latter section. The project's code has been placed on GitHub [4].

4.2 ReqBot under the bot taxonomy lenses

Lebeuf's taxonomy was made to classify and evaluate software bots. As Figure 8 in Section 3.3.2 shows, Lebeuf divided the taxonomy in 3 parts: the environment dimension, the intrinsic dimension and the interaction dimension. Lebeuf provided possible values or ranges for each of the facets in each dimension. Figures 9, 10 and 11 show the taxonomy of ReqBot in the environment, intrinsic and interaction dimension respectively. The boxes with solid lines are the facets and those with dotted lines are ReqBot's values for these facets.

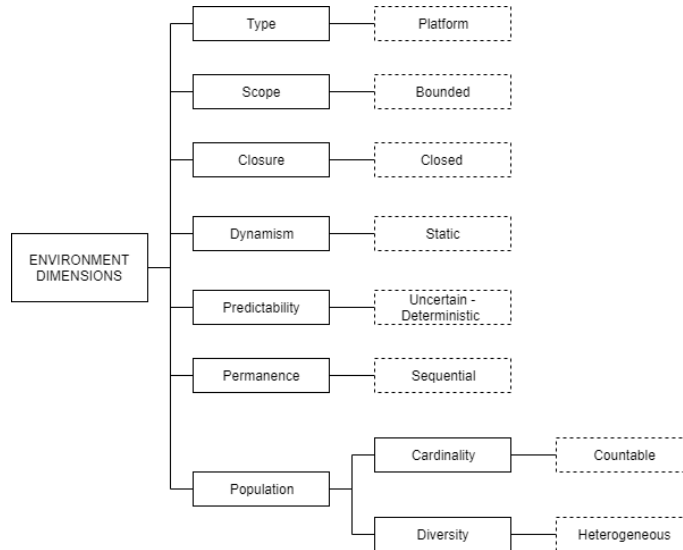


Figure. 9: Environment dimension based on the taxonomy by Lebeuf [43]

Starting with the **Environment dimension**, Lebeuf defines the environment relatively vague or open to interpretation as: “describes the surrounding in which the bot lives and operates” [44]. This thesis’ interpretation of it is that ReqBot lives operates from the Bot Framework emulator, making this its environment and can interact with a user in that environment, namely the one talking to ReqBot.

With this in mind, ReqBot’s environment type is a platform as it can only run in the Bot Framework Emulator at this point in time. This also results in the scope of the bot, being the size of its environment, to be limited and thus bounded. The closure, who has access to the environment of the bot is thus also closed, as only those with access to ReqBot’s code can access it at this time. If an updated version would go live via Azure, these facets of the environment of the bot would change. ReqBot would not be influenced by changes in the environment so it is classified as static. The predictability of ReqBot varies between deterministic and uncertain. All the actions of the bot can be predicted in general, however for most responses we created different textual formulations. Different responses with the same semantic meaning are used at random to make the chatbot feel more human and less repetitive, thus making it not fully predictable. The permanence of ReqBot is sequential as choices or responses made by a user can affect the rest of the conversation, at least until the conversation is ended. The environment has a countable population of 2, namely the chatbot and the user. This, in turn, makes the environment heterogeneous as both members are of a different type: one a human, the other a bot.

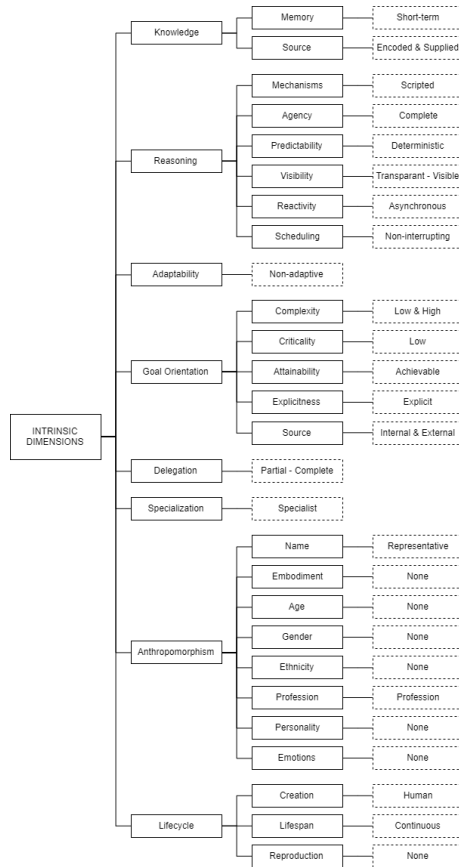


Figure. 10: Intrinsic dimension based on the taxonomy by Lebeuf [43]

The **Intrinsic dimension**, as visualised in Figure 10, concerns among others the facet knowledge. The memory of this knowledge is short-term and the source is partially encoded and partially supplied. All knowledge of an interaction is reset after each termination of the bot, but during run-time, the bot does use supplied knowledge such as a person’s name.

Concerning the reasoning of ReqBot, the mechanisms are scripted and the predictability is deterministic as the same inputs would always yield the same results. The predictability might seem mixed due to the earlier mentioned randomisation of textual formulation, but as all have the same semantic meaning this does not apply. ReqBot has complete agency as it can carry out its goals and tasks without any approval from an external party. Its reasoning is in between transparent and visible as it explains at times why something is happening to make the user more aware of the situation and hopefully get a better response.

Furthermore, the reactivity is intentionally asynchronous to make the bot appear more human. A direct response was met with negative responses during the testing sessions, while a delayed response gave the impression of the bot thinking or processing the input. Lastly concerning the reasoning of ReqBot, its scheduling is non-interrupting, meaning that only one task can exist at a time. All other stimuli are ignored until that task is done.

ReqBot is non-adaptive as it is not able to change its behaviour at runtime so all adaptability sub-facets are void. The bot has multiple goals, with the main one being: to elicit end-user requirements. The bot's other goal is to reduce ambiguity. The complexity of this is considered as rather low since it can be fulfilled by having a user answer a few questions, there are no complex calculations needed to complete the goal. The criticality is low as well as the consequence of failing the goal coincides with one end-user not entering his or her requirement. While this should not happen, no critical business functions depend on the bot to achieve its goal. By having such an easy or well-defined goal, it becomes achievable. Furthermore, the goals are quite explicit as parts of them are even mentioned to the user during interactions to help the user understand the intentions of ReqBot. The source of the two main goals differs as the bot always tries to elicit requirements making it an internal goal, meaning the goal originates from and is triggered within the bot. However, the goal of reducing ambiguity only is triggered if ambiguity occurs in the environment, moving its source to external.

Concerning the delegation of ReqBot, the degree in which the bot has the authority to act on the behalf of others, it sits in between partial and complete. The bot has the authority to act as an elicitation assistant, but only provides potential requirements. At this stage, the requirements would need to be manually analysed. ReqBot is a very specialised bot as it is made for one purpose: to elicit end-user requirements. If e.g. one were to attempt to make a random conversation with the bot, it would not be able to respond accordingly.

As for anthropomorphism, the bot has besides its name, ReqBot, and its profession of a requirement elicitation assistant no anthropomorphic features. These features were not deemed crucial for the bot and some could even be counterproductive as they could present an overload of information to the user. Closing this dimension is the lifecycle. ReqBot is created by a human when it is started and will not terminate until it is instructed to, making its lifespan continuous. The bot is also not able to create other bots or instances of itself.

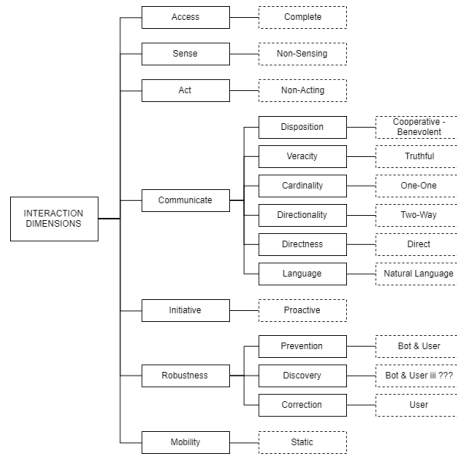


Figure. 11: Intrinsic dimension based on the taxonomy by Lebeuf [43]

The third and final dimension, the **interaction** dimension, is visualised in Figure 11. ReqBot is allowed complete access to every part of its environment it is able to access, so there are no artificial restrictions. However, discounting the user it can not access or sense anything in its environment and is also unable to make changes to it. The bot has no sensors to pick up stimuli or physical output options. However, ReqBot is communicative with the user via text messages.

The disposition of the bot, its willingness to help, falls between cooperative and benevolent as it focuses on one task and assuming it is the same task as the user has, ReqBot will provide help. During this, it is truthful as no intentional lie or deception is built in. During the interaction it interacts one-on-one with a user and the direction of the conversation is a limited two-way. The user is not at any point in the conversation able to drastically influence the responses of the chatbot, but at several points, the bot and the user can influence one another’s reaction. At all times, the bot is direct with its interaction, asking the information it needs. The language capabilities it uses to do this vary from keywords to natural language. While ReqBot is not able to semantically understand natural language, it uses it for various purposes, such as processing it to find ambiguities.

An important aspect to realise of ReqBot is that this chatbot is proactive. Unlike a vast number of chatbots, ReqBot does not answer questions by matching the best response to the question. It leads the conversation, asks questions and guides the user. This makes it possible to collect requirements as the user might not know what information is needed for a complete requirement. However, by splitting the required information up in smaller pieces and let the chatbot ask these to the user it can elicit the right information and thus the requirement.

Finally concerning the robustness of the bot, it has several options for error prevention, which can both be done by the user and the bot. ReqBot provides for certain questions a few clickable options, preventing the user from making an error at that point as any other input will not be accepted. The user has the option to prevent errors in an acceptance interaction in which the bot shows the full requirement and requests if it is correct as it is. If the user thinks that this is not the case, he or she can make a change in the means, ends or role. Whereafter the bot returns to the same confirmation question. ReqBot is also able to discover errors in the form of ambiguity. The user can correct these by specifying his or her intention once the bot asks for specifics.

4.3 Functionality of ReqBot

Figures 12, 14, 16, and 15 show all functionalities of ReqBot using the Business Process Modeling Notation (BPMN) 2.0. The complete BPMN diagram is visualised in Appendix B. This diagram shows the path through the chatbot with all interaction possibilities of a user.

Most activities represent one response, either from the chatbot or from the user. Some do however represent more than one response to keep the diagram cleaner and focused. Most responses from ReqBot have several variants with the same semantic meaning, e.g., an affirmative response could be “All right, thanks”, “Noted it”, “Thank you, I got it” or “Okay, I got that”. If several variants exist, one will be picked at random. This was implemented to make ReqBot less stale and provide variation to keep a user engaged if that user would enter multiple requirements and thus would go through the same dialog. All possible dialog responses can be read in Appendix C.

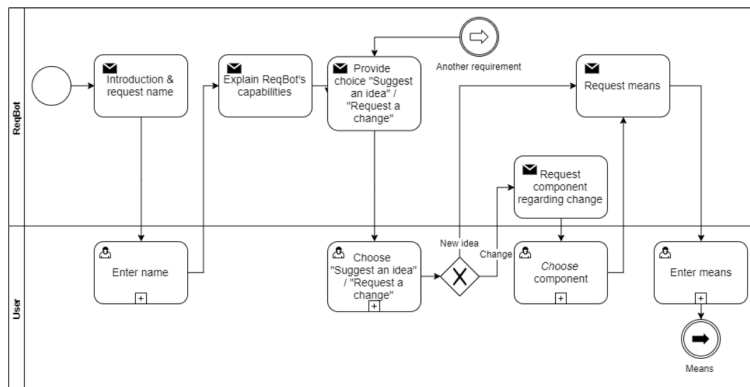


Figure. 12: BPMN model of ReqBot 1/3

The chatbot starts each conversation with an explanation of itself and requests the name of the user in the next message. This is visualised in Figure 12. After a name is entered, ReqBot explains that it can help with entering a change request or a new idea and the user is presented with two buttons, one for each option. At this point, the user has to pick one of these options, all other attempts would yield a message urging the user to choose one of the options. There are 3 standard ways to pick an option in the Bot Framework: (1) click the button, (2) type the text of the button and send, (3) enter and send the number corresponding to the number of the option. So the left button would always be *1*, the one next to it *2*, etc. This holds for all option-interactions of the user. In the diagram, these can be recognised by the word *choose* in the user lane. Figure 13 gives an example of what this part of the conversation amongst others looks like in practice.

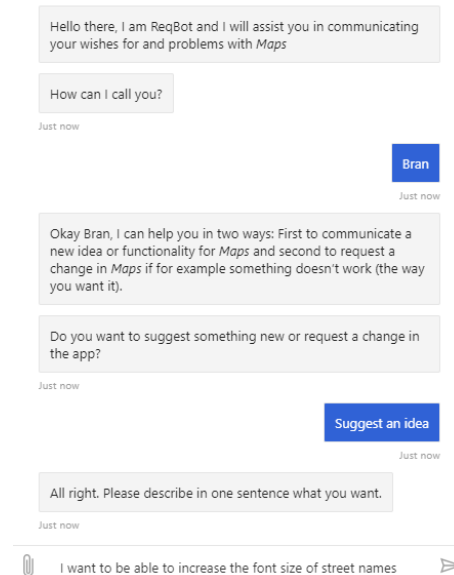


Figure. 13: Screenshot of a conversation with ReqBot

If the user picks the option *Request a change*, ReqBot asks an additional question, namely which component this change concerns. The user has several options, including the option *Other...* which allows the user to type an answer other than the given options. ReqBot tells the user that it understands that the option was not in the list and asks if the user can write it down. This is thus an example of multiple responses within one process in the diagram. All processes with *choose* in italic also have this option of the user adding a non-listed response. The additional question has been added to further clarify and

classify the requirement. If a requirement is written in a manner that is unclear or ambiguous, it might be solved by placing it in the context of a component. Moreover, it would make sorting requirements easier if ones concerning comparable components could automatically be sorted together. This question is only triggered if the user requests a change as a change, per definition applies to something existing. On the other hand, a new idea can be but does not have to be part of an existing component. With software development in mind, we wanted to keep it as high-level as possible, so the user would have a realistic chance to identify where the change should apply to. Therefore, we opted for components and made the suggestions broad and clear such as account, login or menu.

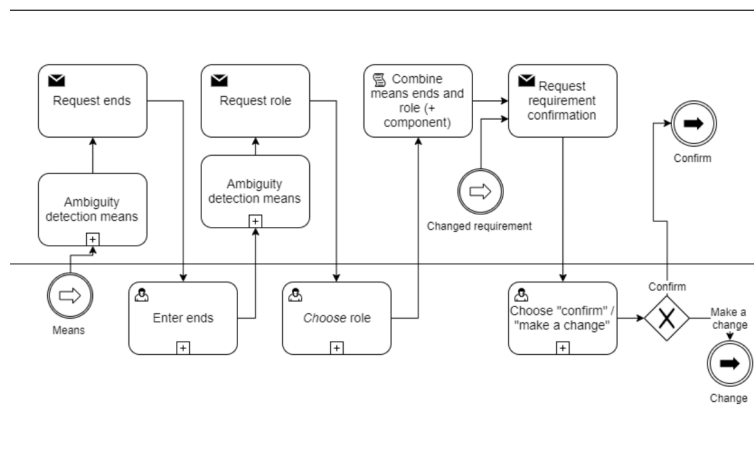


Figure. 14: BPMN model of ReqBot 2/3

After picking a component or answering *Suggest an idea*, the user is asked the means, the ends and the role, as depicted in Figure 14 which has been connected to the other parts of the model via link events. The role is choice based with the option *Other...* to write down a role if it is not one of the options. The order of these three is based on importance and the train of thought of the user. Asking why one would want something before asking what they would want would clash with this train of thought. How can you answer why you want something if you have not yet formulated what you want? This results in the means being placed before the ends. The role is less intertwined with those and could thus be placed before or after the means and ends. By putting the role last, ReqBot starts to ask what the user wants, which is probably the reason someone would use the chatbot in the first place: they want to add or change something. Therefore, we deemed it best to start with the means to keep the user engaged.

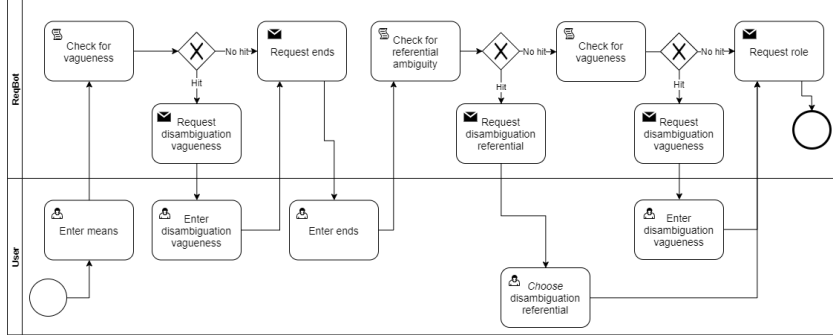


Figure. 15: BPMN model of ambiguity detection and resolution

After entering the means and after entering the ends an ambiguity control takes place. This sub-process is visualised in more detail in Figure 15. Once the user enters the means, this text will be analysed for vague words. These vague words are stored in a separate CSV file and are largely based on the words used to detect vagueness in QuARS tool, originally proposed by Lami [42]. We combined an initial list of vague adjectives and their comparatives and superlatives used for testing with the complete list from the QuARS tool to get the final list of vague words. Once a vague word is detected, ReqBot asks the user if it would be possible to specify that word, preferably using units. After the user enters this, the bot proceeds to ask the ends. If more vague words are used in one sentence, only the first one will be detected as we decided that no more than 1 disambiguation attempt can be made per response of the user. By asking 2 or more questions, we might discourage the user to continue and lower their satisfaction. Detection of referential ambiguity does not occur in this process as this would be the first full sentence the user would type, meaning that there is little to refer to at this point.

The ambiguity control that occurs after the ends are entered works similar to the one after the means is entered. The check for vagueness works exactly the same, however, before this check is initiated a check for referential ambiguity is performed. If a referential ambiguity is found, no detection for vagueness will be done keeping in line with a maximum of 1 disambiguation attempt per response of the user. The detection of referential ambiguity works as follows:

1. Means and ends are POS tagged via an API by NLTK [7]
2. Check if 2 or more nouns exist in the means
3. Check if 2 or more nouns exist in an earlier sentence in the ends (only applies if multiple sentences are used in the ends)

4. Check for personal pronouns in the ends
5. If 4 and either 2 or 3 are true, the disambiguation process is triggered

ReqBot asks in the disambiguation process to what the personal pronoun refers and gives all nouns from 2 and/or 3 as options. The options *None of the options* and *Multiple words...* are added. The former continues the conversation directly to the request for the role, while the latter gives the user the option to type in an answer.

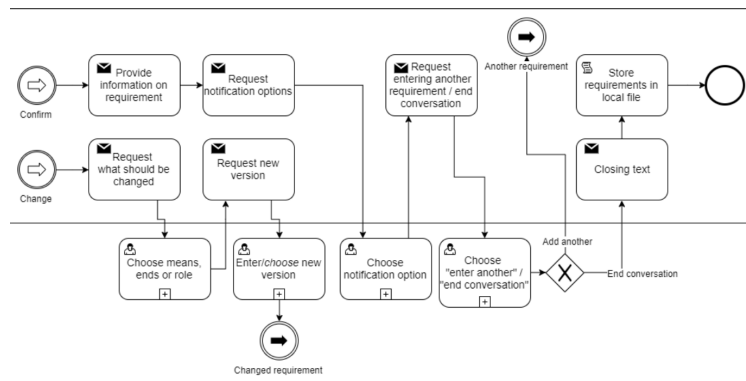


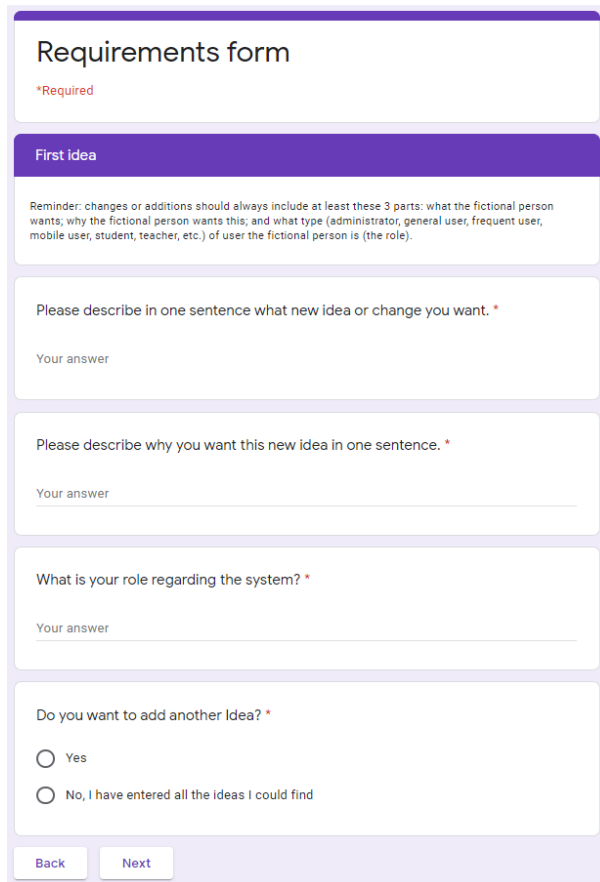
Figure. 16: BPMN model of ReqBot 3/3

Once the role is entered, ReqBot will give an overview of what the user has entered for the means, ends, role and possibly component. Thereafter it asks if the user wants to submit the requirement or make changes. The option of making changes lets the user choose what part should be changed: the means, ends or role. After changes are made, ReqBot shows the overview again with the same question. Once the requirement is submitted, the bot tells the user on how to track the requirement using a code and asks if the user wants to be notified by email. If the user chooses to be notified by email, the bot will ask for the email address. These interactions were added to give the bot a sense of more realism and to show the users that their input is being used. In this version, those options are available but have no consequences. So no one will be emailed if they enter their email address. Lastly, ReqBot asks if the user wants to enter another requirement or end the conversation. If the user chooses to enter another one, they are taken back to the first question: whether this one is a new idea or a change. Once the user chooses to end the conversation, all requirements in that session will be written to a CSV file and the conversation will end.

4.4 Other artifacts

4.4.1 Requirements form

We created several other artifacts besides the main artifact, ReqBot, to conduct the experiment. We created a Google form containing the same questions as the chatbot, as can be seen in Figure 17. This is the artifact with which the chatbot will be compared. The Google form, also referred to as the requirements form, represents current non-interactive methods of gathering end-user requirements. The questions and the formulation of each question remained the same in both the chatbot and the form to improve comparability between the two. If in one the questions were phrased differently or combined in larger questions, the difference in any results could be contributed to the formulation of questions instead of the method.



The screenshot shows a Google Form titled "Requirements form" with a red asterisk indicating required fields. The form is divided into sections by purple headers. The first section, "First idea", includes a reminder: "Reminder: changes or additions should always include at least these 3 parts: what the fictional person wants; why the fictional person wants this; and what type (administrator, general user, frequent user, mobile user, student, teacher, etc.) of user the fictional person is (the role)." This is followed by three text input fields with prompts: "Please describe in one sentence what new idea or change you want.", "Please describe why you want this new idea in one sentence.", and "What is your role regarding the system?". The final section asks "Do you want to add another Idea?" with radio button options for "Yes" and "No, I have entered all the ideas I could find". At the bottom are "Back" and "Next" buttons.

Figure. 17: A screenshot of a section of the requirements form

The form starts with an explanation and then asks the name, notification preference and email of the user. After which, each page consists of one requirement, consisting in turn of means, ends and role. For each of these three parts of the requirement, the exact same wording was used as in the chatbot. Each requirements page contained a fourth question, asking whether the user would want to add another requirement. If they answered yes, they could enter another requirement with a maximum of 7 and on a no, the form jumps to the final page. This way we made it possible for users to enter as many requirements as they deem necessary.

4.4.2 Scenarios

We also created scenarios that participants would use during the experiment. A scenario is a fictive story about a fictive individual that uses a real system. In the scenario, the fictive user addresses some of the problems he or she has with that system.

We created the scenarios for several reasons. First, scenarios ensure that participants enter requirements that should be semantically similar for each scenario. Without scenarios, participants could come up with wild ideas or things that are not actually requirements making analysis harder. Second, thinking up requirements on the spot can be hard and time consuming for participants. According to Bloom's taxonomy [19], later revised by Kratwohl [40], 6 levels of learning exist hierarchically. The highest and thus hardest level is to create something, which is what we ask from participants, namely to create a requirement. By making sure that all the pieces are present to make a requirement we lower the cognitive load compared to if participants would have to come up with requirements on their own. Last, scenarios make it possible to measure the correctness of a certain requirement. A participant either did fill in a certain part of the requirement or did not. The main drawback of using scenarios is that it creates an artificial situation far removed from reality.

We created 5 scenarios, 1 for the sessions during the development of the chatbot and 4 to be used during the experiment, all can be found in Appendix D. A scenario starts with the name of the system. In the experiment, we used Google Maps, Facebook, Youtube and Thuisbezorg.nl (a Dutch food delivery app). After that, the fictive person is introduced and the way that person uses the system and with which goals. This part contains no requirements. Below the introduction are 5 stories, each entailing one requirement. Each story contains 1 thing the person would want to change or add and at least one reason why. In other words, the means and the ends. Take this story on Google Maps:

While looking at the map, he cannot read all of the street names. Bran does not want to zoom in to enlarge them as that would lower his overview. He thinks there should be a way to increase or double the size of these characters.

In this story the fictive person wants to increase the size of the characters of the street names (without zooming in) (*=means*) so that he can read the street names (*=ends*).

All scenarios concern relatively well-known systems so that participants can visualise what each requirement means. A more niche system might deter participants that have never heard of the system as they might not know the system and have trouble understanding the scenario. However, participants do not have to be users of the system in the scenario to be able to use it in the experiment. The way the scenarios are written only requires the most basic of knowledge of that system to complete the scenario. The application of a fictive person is applied to avoid the possibility that participants do not see these requirements as their own ideas and react to that. By making them part of someone else's wishes and only let the participant translate those, we hope to create a level of detachment, preventing interpretation and input from out of the scope of the scenario. Final, all scenarios were kept at approximately the same length preventing large fluctuations in results by the sheer size of the scenario.

5 Experiment and results

In this chapter, the experiment and the obtained results are discussed. First, the experiment is described in detail, followed by the selection and contacting of participants. Whereafter, the method of analysis, the hypotheses and statistical results following the experiment are described.

5.1 Experiment

In this section, the experiment is described in detail. First, the preparations and set-up are described, followed by the experiment procedure including handling questions, see Appendix F for a step-wise version of the procedure. Last, the participants are discussed and how they were contacted. The results of the experiment are discussed in Section 5.2

5.1.1 Set-up

During the experiment, we used participants to test 3 different conditions of entering requirements: the requirements form, the chatbot with ambiguity detection enabled (hereafter referred to as ReqBot+) and the chatbot with ambiguity detection disabled (hereafter referred to as ReqBot regular). Each participant completed the requirements form and one of the chatbots using 2 different scenarios during the session. We decided to let every participant do 2 conditions to gain more data from the same number of participants while simultaneously gathering paired data. To counter influence on the results based on which condition was done first and which was done second, they were assigned using a randomised planning. So one participant starts with the form and then does one of the chatbots, while the other could start with one of the chatbots. The scenarios were also assigned using the same randomised planning.

With 4 scenarios and 3 different conditions, there are 12 combinations, discounting which of the 2 scenarios comes first. The planning ensures that every combination is evenly distributed. This means that e.g. the scenario Facebook in combination with ReqBot regular will be as frequent as the scenario Google Maps with ReqBot+. The exception is the combinations with the requirements form as they are as frequent as both versions of the chatbot combined and thus twice as frequent as one of the versions of the chatbot. See Appendix E for the resulting planning.

5.1.2 Experiment procedure

Each experiment started with a short explanation of the experiment after welcoming the participant. After this, they were given an informed consent form and a written down introduction to the experiment. After they signed and read these, they were given their first scenario and condition. Once they entered all

requirements, they were given the system usability scale (SUS) questionnaire [20]. Once they completed this, they were given the second scenario and condition followed by the second SUS questionnaire. Hereafter, the participants would be thanked for their participation and the session ends. The detailed approach and the aforementioned description of the experiment can be found in Appendix F and Appendix G, respectively.

We started with a pilot, which involved 2 participants and was used to determine whether the experimental approach needed any changes. After evaluation, we concluded that no changes were required. Therefore, the experiment approach remained unchanged and the data from the pilot was used as regular data in the analysis. Each session was performed one-on-one with a participant in rooms with as little as possible external stimuli, meaning no crowded or noisy locations were used. This was to minimise distractions for the participants during the experiment, which could influence the results. A session took about 20 to 40 minutes from beginning to end.

During a session, participants were able to ask questions. The thesis' author answered all linguistic questions, such as what a specific word means. All questions on the proceedings before a participant received the condition and scenario were also answered. If a participant were to ask questions such as whether they should add another requirement or what the role of the fictive user is, the answer would be more generic e.g. "do what you think is best" or "both options are fine". This way, the resulting influence of questions from participants was attempted to be minimised.

5.1.3 Participants

Participants were not selected based on demographics as anyone using a system can be an end-user who could submit a requirement. The only people that were excluded were those who do not use the internet on a regular basis, have no affinity with technology at all or can not read and write English. We gathered participants from my personal and study network, combined with the network of the supervisor. In their invitation, potential participants were given a short explanation of the experiment and an indication of the time it would take. They could select a time and date over a period of 3 weeks, using a calendar tool. The aim was to reach 40 participants so we could do a meaningful quantitative analysis.

5.2 Results

The results of the experiment described in Section 5.1 are presented. First, an introduction to the analysis of the data is presented along with general descriptive results. Then, the results from the system usability scale questionnaire yielding the usability scores are described. Whereafter, the correctness of the requirements that are entered by the participants is described. Finally, the quality aspects of the requirements are analysed. This focuses on the detection and resolving of ambiguities in the requirements. In the sections on usability and correctness, the applied metrics are described first. Whereafter, the hypotheses are given and explained, followed by the statistical results. The implications of these results are presented and discussed in Chapter 6.

5.2.1 Collected outputs

Over a period of 4 weeks, 40 participants took part in this research. Each of them completed 2 of the 3 conditions (Form, ReqBot regular and ReqBot+). This resulted in a total of 395 requirements produced by the participants. 197 of them came from the form, 98 from ReqBot regular and 100 from ReqBot+. This does not add up to 400 (40 participants * 2 conditions * 5 requirements per condition) as participants were practically free to enter as many requirements as they perceived.

The Google form had an upper bound of 7 requirements, which was not reached as no participant entered more than 5 requirements in any condition. The data that has been collected on one requirement consists of the participant number, the condition, the means, the ends and the role. Both chatbot conditions also gathered the component the requirement relates to if it is a change and old means, ends and role if the participant used the make changes function. Last, ReqBot+ also contains detected ambiguities and their resolution.

Concerning the SUS questionnaire, the condition and participant number combined with the scores of all 10 questions was collected and stored. These scores range from 1 to 5, with 0 representing the answer completely disagree and 5 representing completely agree. Each participant filled in 2 questionnaires, one for each condition. This resulted in 40 SUS scores on the form, 20 on ReqBot regular and 20 on ReqBot+.

5.2.2 Usability

The system usability scale is a 10 item scale that gives a subjective assessment of the usability of a system [20]. It was proposed by Brooke and adheres to a 5 point Likert scale. The scale yields a score between 0 and 100, which is computed as follows:

- The results were transformed into numerical scores ranging from 1 to 5, with 1 representing the option completely disagree and 5 representing the option completely agree.
- For questions 1, 3, 5, 7 and 9, 1 is subtracted from the numerical score.
- For questions 2, 4, 6, 8 and 10, the numerical score is subtracted from 5.
- The resulting scores are added and multiplied by 2.5 creating a scale from 0 to 100

Lewis and Sauro argue that the system usability scale can be divided into both usability and learnability. They make the case that question 4; “*I think that I would need the support of a technical person to be able to use this system*” and question 10; “*I needed to learn a lot of things before I could get going with this system*” measure learnability, while the other 8 questions measure usability [46]. The 8 question usability variant will be referred to as **restricted usability** and the original approach by Brooke as **usability**.

Lacking a proper scale for restricted usability and learnability, both are put on a 0 to 100 scale to be more comparable with regular usability. To accomplish this, restricted usability scores are multiplied by 3.125 instead of 2.5 and the scores of learnability with 12.5 instead of 2.5. To ensure completeness, **both approaches**, thus Brooke’s and Lewis and Sauro’s approach, are used in this analysis.

5.2.2.1 Hypotheses

With 3 conditions (form, ReqBot regular and ReqBot+) tested on participants and 2 usability plus another learnability interpretation, there will be 4 comparisons made on the SUS score for each interpretation. These are form - ReqBot regular, form - ReqBot+ , form - chatbot combined and ReqBot regular - ReqBot+. By applying statistical tests, the research aims to draw conclusions on which condition is rated highest in terms of usability.

4 hypotheses were created which would be tested to determine whether a condition yields a significantly higher score than another. Both interpretations of usability were grouped together in these hypotheses as they were the hypotheses same for each interpretation.

U1: Form - Chatbots combined

- H0 There is no significant difference between the usability/learnability score of the form and the chatbots combined
- H1 The chatbots combined have a significantly higher usability/learnability score than the form

U2: Form - ReqBot regular

H0 There is no significant difference between the usability/learnability score of the form and the chatbot

H1 The chatbot has a significantly higher usability/learnability score than the form

U3: Form - ReqBot+

H0 There is no significant difference between the usability/learnability score of the form and ReqBot+

H1 ReqBot+ has a significantly higher usability/learnability score than the form

U4: ReqBot regular - ReqBot+

H0 There is no significant difference between the usability/learnability score of ReqBot regular and ReqBot+

H1 ReqBot regular has a significantly higher usability/learnability score than ReqBot+

The rationale behind these hypotheses is that a user would prefer to interact with a chatbot, rather than a one-sided interaction with a form. As for U4, it is expected that the interaction will be longer because if an ambiguity is detected, extra dialog will follow. In this dialog the user would be requested to clarify him or herself, which would lower the usability due to this possibly being interpreted as annoying or time-consuming.

5.2.2.2 Statistical results

Table 3 shows the mean usability scores of each condition and interpretation. The scores are quite close to one another, but in each interpretation, ReqBot regular has the highest scores. So ReqBot+ was also rated slightly lower than ReqBot regular. It should also be noted that the learnability scores are higher than the restricted usability scores, almost to the point of perfect scores. This indicates that the learnability of each condition was deemed high by the participants, which can be caused by e.g. the conditions being simplistic, intuitive or well explained. Whether the high learnability scores were due to one of these factors, a combination or other factors could not be concluded from the data.

Condition\Interpretation	Usability	Restricted usability	Learnability
Form	78.1	74.2	93.8
Chatbots combined	79.6	76.1	93.8
ReqBot regular	81.4	77.8	95.6
ReqBot+	77.9	74.4	91.9

Table 3: Mean usability/learnability scores (scale 0 - 100)

To discover whether the difference in usability is statistically significant between the conditions, the related samples Wilcoxon signed-rank test was used for U1, U2 and U3 as these deal with paired values. This is due to the participants completing 2 conditions each. For U4 the Mann-Whitney U signed rank test was used as both versions of the chatbot are compared in this case and the participants were assigned one version of the chatbot and the form, never both versions. This means the samples of these are not paired.

Figure 18 shows the frequency of the differences in usability scores between the form and both chatbots combined. In this chart, the scores from the form are subtracted from the scores from the bots. This results in the bots scoring 25 times higher than the form; the form scoring 14 times higher than the bot and 1 draw.

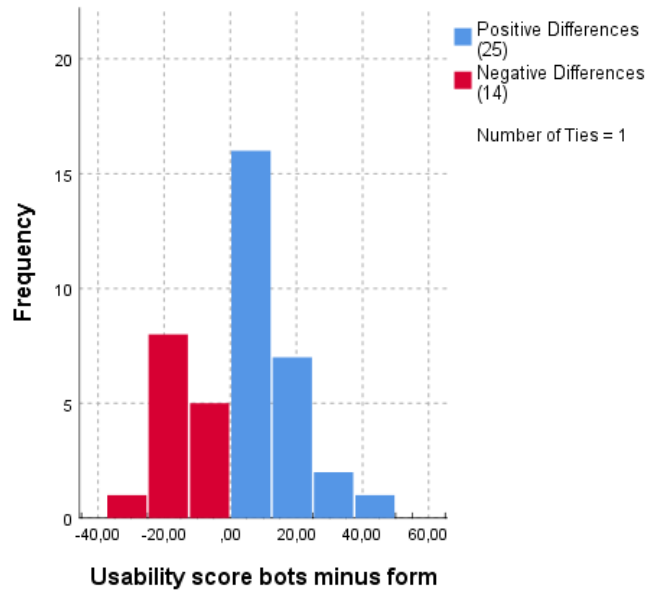


Figure. 18: Frequency in differences between usability scores between the bots and the form

This apparent preference for the chatbots does, however, not translate into a significant difference. In fact, U1, U2, U3 and U4 yield for every interpretation a p-value well above 0.05, meaning H0 can not be rejected in any case. This

means that ReqBot regular, ReqBot+ and the form do not differ significantly in terms of usability, restricted usability and learnability scores. All p-values can be found in Appendix H.

5.2.3 Correctness

The correctness score of a participant on a condition ranges between 0 and 10. It is made up of all the individual correctness scores of a requirement, each varying between 0 and 2 points. 1 point can be scored for the means and 1 point can be scored for the ends. To determine whether a means or an ends was correct, we, this thesis' supervisor and this thesis' author, created a golden standard for each scenario.

The golden standard consists of tagged parts of sentences of the story making up a requirement. It represents the essence of the means and the ends that has to be included in the requirement of the participant for it to be tagged as correct. To get this golden standard, we individually tagged the parts of sentences that contain the means or the ends for each requirement from every scenario. After which, we compared and discussed the results ending in a consensus being the golden standard.

Using the golden standard, we individually tagged every means and ends of the participants' responses as either correct or incorrect. Whereafter, we deliberated all disagreements resulting in a consensus on whether a means or an ends was correct. In a few cases, a participant combined 2 requirements in 1, thus only entering 4 in a scenario. If it was clear that this was the case and both the means and ends from both requirements were present in that single requirement, it would get full points for both requirements.

5.2.3.1 Hypotheses

Comparable hypotheses were made to usability were made for correctness as the same conditions are compared, just a different aspect of it. However, unlike usability correctness is not split up but remains as one variable. By testing the following 4 hypotheses, differences in correctness could come to light.

C1: Form - Chatbots combined

H0 There is no significant difference between the correctness score of the form and the chatbots combined

H1 The chatbots combined have a significantly higher correctness score than the form

C2: Form - ReqBot regular

H0 There is no significant difference between the correctness score of the form and ReqBot regular

H1 ReqBot regular has a significantly higher correctness score than the form

C3: Form - ReqBot+

H0 There is no significant difference between the correctness score of the form and ReqBot+

H1 ReqBot+ has a significantly higher correctness score than the form

C4: ReqBot regular- ReqBot+

H0 There is no significant difference between the correctness score of ReqBot regular and ReqBot+

H1 ReqBot+ has a significantly higher correctness score than ReqBot regular

These hypotheses are based on the rationale that it is expected that with a chatbot asking for blocks of information (means, ends and role), the participant stays focused on the task of providing this information and that this would trigger the participants to actively search for this information in the scenarios. This would result in a higher correctness score for both chatbots compared to the form. There is not a significant difference expected in terms of correctness score between both chatbots. If it exists, it is expected that the correctness score in ReqBot+ would be higher as it keeps the participants longer engaged. This gives them more time to recognise a mistake and correct it.

5.2.3.2 Statistical results

Table 4 shows the mean correctness scores for each condition. There is not that much difference between the correctness scores, but overall the chatbots score combined better than the form. This is largely due to ReqBot regular yielding the highest correctness score.

Condition	Correctness score
Form	8.1
Chatbots combined	8.3
ReqBot regular	8.7
ReqBot+	7.9

Table 4: Mean correctness scores

To discover if there would be a significant difference between the conditions in term of correctness, the related samples Wilcoxon signed-rank test was used for C1, C2 and C3. Comparably to the usability analysis, these concern paired scores, while C4 has unpaired scores. For that hypothesis, the Mann-Whitney U signed rank test was used.

Figure 19 shows the frequency of the differences in correctness scores between the form and both chatbots combined. The scores from the form are subtracted

from the scores from both bots for each participant and the frequency of the result is displayed. Interestingly, there are 10 ties, meaning a quarter of the participants scored equally high in both conditions. Furthermore, most of the differences are minimal, concentrated at 0 and there seems to be little between both conditions.

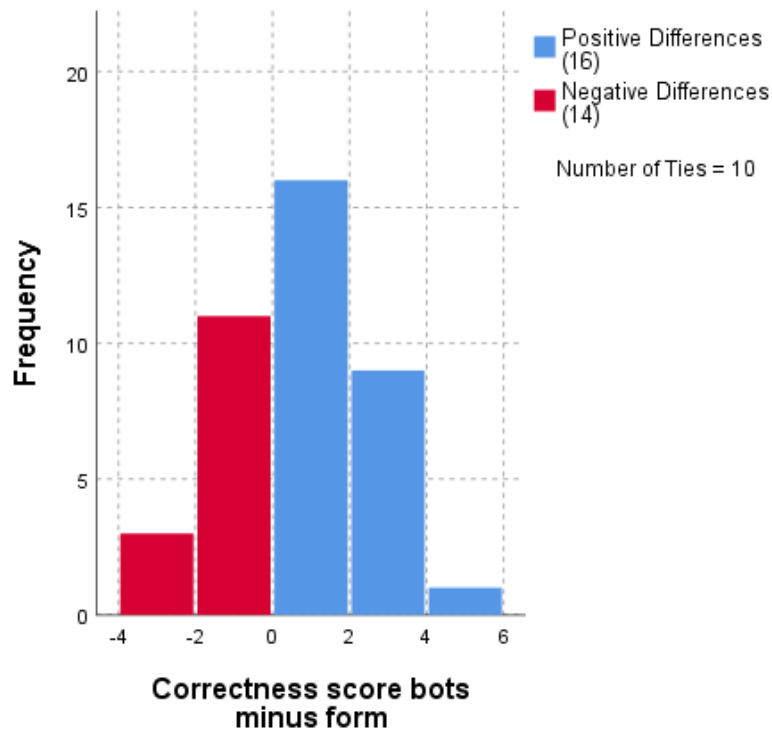


Figure. 19: Frequency in differences between correctness scores between the bots and the form

The similarity in scores can also be traced to the results of the statistical tests. In all 4 cases, the p-value was too high to reject H_0 , meaning none of the conditions are significantly different from one another in terms of correctness. Interestingly, the comparison between both chatbots came closest to significant results with a p-value of 0.13. With the mean correctness scores of ReqBot regular being higher than those of ReqBot+, it almost rejected both H_0 and H_1 , defying the expectation from C4. See Appendix H for all p-values.

5.2.4 Quality

ReqBot+ detects and resolves ambiguities as described in Section 4.3. The 2 major parts are detection and resolution and are evaluated separately. ReqBot+ processed 200 means and ends from 100 requirements created by 20 participants in total. All means and ends were individually analysed for referential ambiguity and vagueness to check whether all were detected or detected correctly.

For each response, the bot could either (i) correctly detect an ambiguity; (ii) incorrectly detect an ambiguity; (iii) correctly detect no ambiguity; or (iv) incorrectly detect no ambiguity. These options result in (i) true positives; (ii) false positives; (iii) true negatives; and (iv) false negatives respectively. These results are listed in Table 5. The 3 ambiguities that can be detected are vagueness in the means; vagueness in the ends and referential ambiguity

	Vague means	Vague ends	Referential ambiguity	Total
True positive	3	2	1	6
True negative	91	97	86	274
False positive	1	0	13	14
False negative	5	1	0	6

Table 5: Detection of ambiguity in the ReqBot+ condition

This yields an overall **precision of 0.3** and a **recall of 0.5**. These are not great scores but they are highly influenced by the detection of referential ambiguity, which has a quite basic algorithm for detection. Only looking at the detection of vagueness would yield a precision of 0.8 and a recall of 0.5.

In the other two conditions, the form and ReqBot regular, there are also ambiguities. However, these were not automatically detected and resolved as there was no mechanism active to do this nor was this desired for these conditions. The ambiguities were detected using the same manual analysis as for ReqBot+. This results in a total of 29 ambiguities in all conditions combined. 20 of these are vague means, 5 are vague ends and 4 are referential ambiguities. For this dataset of 395 requirements, it means that 7.3% of them contains one of the researched ambiguities.

6 Conclusion

In this chapter, we discuss the results of our study and use them to answer the research questions. We first address the sub-questions and then answer the main research question.

6.1 Conclusion of sub-questions

To provide structure to the research process, 4 sub-questions were formulated. The first sub-question is fully based in the theoretical foundation and the second is partially based in literature and partially an own design. The third is mainly focused on an iterative design process with links to the theory and the fourth is based on the results of the experiment.

SQ1 *What measures determine the quality of a requirement?*

What measures determine the quality of a requirement depends on the type of requirement. As there are many formats to write requirements, not all formats can adhere to the same detailed measures. More generic measures such as Phol's three dimensions framework can be used for any format [58, 57]. However, to get easier-to-assess measures, the requirements format needs to be specified. For this research, that format was user stories. For user stories, the quality user story framework provides 13 measures, so-called criteria, to determine the quality, see Figure 6 in Section 3.1.1. These criteria are not always the most atomic measures but can consist of measures of their own. The criteria this research focuses on, ambiguity, has multiple different types each with its own implications, see Figure 7 in Section 3.2.1. This research zooms in on two types of ambiguity that can be measured in a requirement:

- Vagueness: When it is not clear or impossible to measure whether the requirement is fulfilled
- Referential ambiguity: When an anaphor can take its reference from more than one element, each playing the role of the antecedent

Concluding, there are multiple ways of measuring the quality of a requirement. Picking the right one depends on the requirement format and where you want to place the emphasis. In this thesis, because the expected impact of these ambiguities on the requirements was high and the practicality of detecting and resolving them seemed achievable, we chose vagueness and referential ambiguity.

SQ2 *What techniques can be used to automate the detection of defects in a requirement for a selected measure, as defined in SQ1?*

The two measures for ambiguity and thus quality, have to be detected after an interaction to be resolved during the conversation. Vagueness and especially

vagueness of words can be detected using a list of vague words [69, 42]. If a word on that list occurs during a conversation, the detection should trigger a response.

Referential ambiguity can be resolved by implementing coreference tagging. However, the free-to-use tools focus on coreference resolution instead of just tagging all the possible words an anaphor can take its reference from. This resulted in the creating of a new algorithm, that focuses on finding nouns in sentences placed before a pronoun, see Section 4.3 for the algorithm.

SQ3 *How to design and construct a requirements elicitation chatbot for improving the quality of requirements?*

Designing a requirements elicitation chatbot starts with understanding what kind of chatbot should be made. First, a set of requirements must be chosen for the chatbot, whereafter a platform or framework can be chosen based on those requirements. After which development can start, using prototypes in iterations to see what works and adjust the chatbot accordingly.

With the use of this approach, we constructed ReqBot. ReqBot leads the conversation with the user by asking questions to the user in a specific order. These questions include the means, the ends and the role, which are needed to construct a requirement. Depending on the answer or choice of the user, ReqBot can give alternate responses or ask additional questions. By applying the methods from SQ2, ReqBot checks for ambiguity after the user enters the means and the ends. If an ambiguity is detected, ReqBot will ask for clarification. In the case of vagueness, ReqBot asks the user to specify that vague word, preferably with units. For referential ambiguity, ReqBot asks to which noun the pronoun refers, providing a selection of nouns the user used earlier. In this manner, ReqBot attempts to improve the quality of requirements, by resolving certain types of ambiguity.

SQ4 *How effective is the chatbot, as created in SQ3, in eliciting high-quality requirements?*

The elicitation process of both versions of the chatbot (ReqBot+ and ReqBot regular) has been contrasted with a Google Form, which is a representation of a current non-interactive method of gathering end-user requirements. The analysis took 3 dimensions in account:

Usability

There is no significant difference detected in terms of usability between the bots and the form, meaning none is deemed better in terms of usability. However, this also indicates that none of them is deemed significantly worse. So even ReqBot+, which could and did falsely trigger a disambiguation process from time to time did not harm its usability scores in a significant way. This indicates that users are willing to clarify possible ambiguities, even if the subject matter

is not ambiguous at all, without letting it negatively impact their experience in a significant way. Moreover, the usability scores in their own right were acceptable to good, being 78.1 for the form and 79.6 for the chatbots combined.

Correctness

In terms of correctness, there was no significant difference between all conditions. This indicates that the type of interaction, from stale as with the form to interactive as with the chatbot, does not significantly affect how well users can formulate requirements given a scenario. Individual differences in correctness can occur, but these can have a variety of causes such as reading skill or the concentration of the user.

Quality

The quality of the requirements, represented by vagueness and referential ambiguity, is improved in a few cases of with ReqBot+. In 100 requirements, there are 6 cases in which an ambiguity was correctly detected and resolved. This indicates that it is possible to improve the quality of requirements using an interactive chatbot with ambiguity detection. However, another 6 ambiguities were missed and in 14 cases the detection was triggered wrongly. This resulted in a precision of 0.3 and a recall of 0.5 for ReqBot+, which is quite low. Including all conditions, there were 29 ambiguities in them combined, meaning 1 out of every 13.6 requirements contained one of these two types of ambiguity. Concluding, the principle of detecting and resolving ambiguities appears to work, but improvements in the detection can and have to be made for it to be usable in a product. This would be advantages though as even only these two types of ambiguities do occur with some frequency.

6.2 Conclusion of main research question

MRQ *To what degree could a requirements elicitation chatbot improve the quality of end-user requirements and the experience of end-users while they express these requirements for a software product?*

The goal of this research was to investigate how well a requirements elicitation chatbot could improve quality by reducing ambiguity in requirements and improve usability. After designing ReqBot and especially the ReqBot+ variant, it showed that selected quality defects can be detected and resolved within a conversation. However, this detection lacks in both terms of precision and recall and therefore it would have to be improved. There appears to be no difference in usability between the Google form and ReqBot, so on that front, it would not be beneficiary to create a chatbot as opposed to a form as long as the form has the same questions. However, it also does not discourage from implementing requirement elicitation chatbots as it did not score lower than a representation of a current method. Concluding, a requirements elicitation chatbot has the ability and potential to improve the quality of requirements but does not significantly improve current techniques in terms of usability.

7 Discussion

In this chapter, the research is discussed along with its limitations and threats. Furthermore, ideas for new studies and continuations of this project are provided in future work.

7.1 Limitations and threats

Over the course of this thesis, several limiting factors were encountered, mainly caused by practical limitations such as time or resources. These could, in turn, threaten the validity of the research. While they were avoided or prevented as much as possible, there still are some limitations and threats in this thesis.

Considering the experiment, the scenarios were set up as a way to give participants the support to write requirements so they would not have to come up with their own. However, these requirements were not their own ideas, making the experiment less natural. In a natural situation, one would notice something that could and should be changed in or added to a system in the opinion of that person. When a certain threshold of e.g. irritation or desire to improve would be reached, that person would look for a way of communicating the desired change or addition. Whereafter, that person would write down the requirement and send it along with possibly more requirements.

This unnatural situation can affect the results, especially those on usability as the chatbot is not used in the environment in which it would be used in practice. The results on quality, specifically on vagueness, could be impacted by the wording in the scenarios as participants can pick words from the scenario to form a requirement. Conversely, in a more natural situation, they would use their own jargon which could result in the use of either more or less vague words.

Correctness depends on the scenario approach as this embodies the comparison between the intended requirement from the scenario and the requirement created by the participant. Removing the scenarios would nullify the results from correctness. The idea behind correctness was that a more interactive manner of asking questions would lead to better-formed requirements. But even if there would be significant results in this area, it could be argued that these were due to the participant's concentration; proficiency in English; or capability to read carefully.

The results on quality should not be generalised too much as ambiguity is only one of 13 criteria of quality according to the QUS framework. Thus ambiguity should not be used to describe impacts on the overall quality. One quality criterion might be changed for the better, but to understand quality as a whole, the other criteria must be studied too. The same applies to referential ambiguity and vagueness compared to ambiguity as a whole. It is only a subset of

ambiguity and thus should not be generalised quickly to ambiguity. The choice to use subsets was made to enable this research to focus on specific measures, while we knowingly sacrificed generalisability.

To conduct the experiment, participants were needed. These ended up being mainly friends, study acquaintances and family, so the sampling was not random. However, it would not be possible from a practical view to have random sampling for an experiment which is performed on location and takes 20 to 40 minutes without, e.g., a reward. On its own, this can threaten the validity of the results but given their relation to the thesis' author, they might have an inclination to be overly positive with their answers for the system usability scale. Therefore this possibility of appeasing threatens the usability results. Moreover, the sample size of 40 participants is limited, but given the resources for this thesis, it can be deemed sufficient.

7.2 Future work

This thesis provides multiple opportunities for future work. It is one of the first times the fields of requirements engineering and chatbots have crossed and can provide a starting point for further research at the intersection of these fields. This thesis can be a reference point for new research or future research can be a continuation of this thesis. A continuation can be done by either adopting and changing ReqBot or starting over with a new chatbot.

First, it would be possible to broaden the scope from a quality point by researching different or more forms of ambiguity or different quality criteria. This would widen the range and, if successful, remove ambiguities from end-user requirements as much as possible.

Second, with a focus more shifted towards NLP, it could be possible to improve ambiguity detection and thus let fewer ambiguities go unnoticed while also reducing the incorrect hits of the detection. Other tools have already achieved these kinds of results using static data, the challenge with this would be to ensure it works quickly on a continuously updating set of data as the detection should happen in a conversation if it is to be resolved within that same conversation.

Third, the scenarios and the forced environment were not ideal to simulate real-world events. But with the basis of this thesis, ReqBot or a future variant could be implemented in a running system and interact with actual end-users of that system trying to communicate their requirements. This would enable an in-depth comparison between the end-user requirements from before the implementation of the chatbot and after the implementation.

Last, this thesis can be seen as a basis for the combination of the user story format with chatbots and could be expanded on by investigating whether other requirement formats would better fit a chatbot.

In short, there are many follow-up directions which future research can take and these are not mutually exclusive. The combined field of requirements engineering and chatbots is wide open and many projects can look into the different areas of this relatively new field.

References

- [1] Azure bot service documentation. <https://docs.microsoft.com/en-us/azure/bot-service/?view=azure-bot-service-4.0>. Accessed: 2020-12-27.
- [2] Definition of ambiguity in english. <https://en.oxforddictionaries.com/definition/ambiguity>. Accessed: 2019-01-30.
- [3] Definition of chatbot in english. <https://en.oxforddictionaries.com/definition/chatbot>. Accessed: 2019-05-13.
- [4] Github reqbot. <https://github.com/marcvalkenier/ReqBot>. Accessed: 2020-01-05.
- [5] Google trends on "chatbot". <https://trends.google.nl/trends/explore?date=2009-04-09%202019-04-09&q=Chatbot>. Accessed: 2019-04-09.
- [6] Meaning of "ambiguity" in the english dictionary. <https://dictionary.cambridge.org/dictionary/english/ambiguity>. Accessed: 2019-01-30.
- [7] Natural language processing apis and python nltk demos. <http://text-processing.com/>. Accessed: 2020-01-05.
- [8] Four types of bots. <https://chatbotsmagazine.com/four-types-of-bots-432501e79a2f>, 2016. 2019-05-15.
- [9] Types of bots: An overview. <http://botnerds.com/types-of-bots/>, 2017. 2019-05-12.
- [10] Rajeev Agarwal and Lois Boggess. A simple but useful approach to conjunct identification. In *Proceedings of the 30th annual meeting on Association for Computational Linguistics*, pages 15–21. Association for Computational Linguistics, 1992.
- [11] Mohammed Javeed Ali. Metrics for requirements engineering. 2006.
- [12] Imran Sarwar Bajwa, Mark Lee, and Behzad Bordbar. Resolving syntactic ambiguities in natural language specification of constraints. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 178–187. Springer, 2012.
- [13] R Barbosa, AEA Silva, and R Moraes. Use of similarity measure to suggest the existence of duplicate user stories in the srum process. In *Dependable Systems and Networks Workshop, 2016 46th Annual IEEE/IFIP International Conference on*, pages 2–5. IEEE, 2016.
- [14] Bas Basten and Tijs Van Der Storm. Ambidexter: Practical ambiguity detection. In *2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*, pages 101–102. IEEE, 2010.

- [15] Hendrikus JS Basten. The usability of ambiguity detection methods for context-free grammars. *Electronic Notes in Theoretical Computer Science*, 238(5):35–46, 2009.
- [16] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. The agile manifesto, 2001.
- [17] Alan Beretta, Robert Fiorentino, and David Poeppel. The effects of homonymy and polysemy on lexical access: An meg study. *Cognitive Brain Research*, 24(1):57–65, 2005.
- [18] Daniel M Berry et al. From contract drafting to software specification: Linguistic sources of ambiguity-a handbook version 1.0. 2000.
- [19] Benjamin S Bloom et al. Taxonomy of educational objectives. vol. 1: Cognitive domain. *New York: McKay*, pages 20–24, 1956.
- [20] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.
- [21] Harm Brouwer, Hartmut Fitz, and John CJ Hoeks. Modeling the noun phrase versus sentence coordination ambiguity in dutch: evidence from surprisal theory. In *Proceedings of the 2010 Workshop on Cognitive Modeling and Computational Linguistics*, pages 72–80. Association for Computational Linguistics, 2010.
- [22] Breno Dantas Cruz, Bargav Jayaraman, Anurag Dwarakanath, and Collin McMillan. Detecting vague words & phrases in requirements documents in a multilingual environment. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 233–242. IEEE, 2017.
- [23] Fabiano Dalpiaz, Ivor Van Der Schalk, and Garm Lucassen. Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and nlp. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 119–135. Springer, 2018.
- [24] Alan Davis, Scott Overmyer, Kathleen Jordan, Joseph Caruso, Fatma Dandashi, Anhtuan Dinh, Gary Kincaid, Glen Ledebor, Patricia Reynolds, Pradip Sitaram, et al. Identifying and measuring quality in a software requirements specification. In *[1993] Proceedings First International Software Metrics Symposium*, pages 141–152. IEEE, 1993.
- [25] Henning Femmer, Daniel Méndez Fernández, Stefan Wagner, and Sebastian Eder. Rapid quality assurance with requirements smells. *Journal of Systems and Software*, 123:190–213, 2017.
- [26] Lyn Frazier. Syntactic processing: evidence from dutch. *Natural Language & Linguistic Theory*, 5(4):519–559, 1987.

- [27] Edwin Friesen, Frederik Simon Bäumer, and Michaela Geierhos. Cordula: Software requirements extraction utilizing chatbot as communication interface. In *REFSQ Workshops*, 2018.
- [28] Norbert E Fuchs and Rolf Schwitter. Attempto controlled english (ace). *arXiv preprint cmp-lg/9603003*, 1996.
- [29] Alex Galert. Chatbot report 2018: Global trends and analysis. <https://chatbotsmagazine.com/chatbot-report-2018-global-trends-and-analysis-4d8bbe4d924b>, 2018. 2019-04-09.
- [30] Gonzalo Génova, José M Fuentes, Juan Llorens, Omar Hurtado, and Valentín Moreno. A framework to measure and improve the quality of textual requirements. *Requirements engineering*, 18(1):25–41, 2013.
- [31] Benedikt Gleich, Oliver Creighton, and Leonid Kof. Ambiguity detection: Towards a tool explaining ambiguity sources. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 218–232. Springer, 2010.
- [32] Eduard C Groen, Norbert Seyff, Raian Ali, Fabiano Dalpiaz, Joerg Doerr, Emitza Guzman, Mahmood Hosseini, Jordi Marco, Marc Oriol, Anna Perini, et al. The crowd in requirements engineering: The landscape and challenges. *IEEE software*, 34(2):44–52, 2017.
- [33] Emitza Guzman and Walid Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, pages 153–162. IEEE, 2014.
- [34] Paweł Hałabuda. What drives the growing popularity of chatbots? <https://blog.apptension.com/2017/07/18/popularity-of-chatbots/>, 2017. 2019-04-09.
- [35] Mahmoud Hosseini, Keith T Phalp, Jacqui Taylor, and Raian Ali. Towards crowdsourcing for requirements engineering. 2014.
- [36] Jizhou Huang, Ming Zhou, and Dan Yang. Extracting chatbot knowledge from online discussion forums. In *IJCAI*, volume 7, pages 423–428, 2007.
- [37] Nadzeya Kiyavitskaya, Nicola Zeni, Luisa Mich, and Daniel M Berry. Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. *Requirements engineering*, 13(3):207–239, 2008.
- [38] Jyrki Kontio, Laura Lehtola, and Johanna Bragge. Using the focus group method in software engineering: obtaining practitioner and user experiences. In *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on*, pages 271–280. IEEE, 2004.

- [39] Sven J Korner and Torben Brumm. Resi-a natural language specification improver. In *2009 IEEE International Conference on Semantic Computing*, pages 1–8. IEEE, 2009.
- [40] David R Krathwohl. A revision of bloom’s taxonomy: An overview. *Theory into practice*, 41(4):212–218, 2002.
- [41] John Krogstie, Odd Ivar Lindland, and Guttorm Sindre. Towards a deeper understanding of quality in requirements engineering. In *International Conference on Advanced Information Systems Engineering*, pages 82–95. Springer, 1995.
- [42] Giuseppe Lami. Quars: A tool for analyzing requirement. Technical Report CMU/SEI-2005-TR-014, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2005.
- [43] Carlene Lebeuf, Alexey Zagalsky, Matthieu Foucault, and Margaret-Anne Storey. Defining and classifying software bots: A faceted taxonomy.
- [44] Carlene R Lebeuf. *A taxonomy of software bots: towards a deeper understanding of software bot characteristics*. PhD thesis, 2018.
- [45] Dean Leffingwell. *Agile software requirements: lean requirements practices for teams, programs, and the enterprise*. Addison-Wesley Professional, 2010.
- [46] James R Lewis and Jeff Sauro. The factor structure of the system usability scale. In *International conference on human centered design*, pages 94–103. Springer, 2009.
- [47] Odd Ivar Lindland, Guttorm Sindre, and Arne Solvberg. Understanding quality in conceptual modeling. *IEEE software*, 11(2):42–49, 1994.
- [48] Olga Liskin, Raphael Pham, Stephan Kiesling, and Kurt Schneider. Why we need a granularity concept for user stories. In *International Conference on Agile Software Development*, pages 110–125. Springer, 2014.
- [49] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn EM van der Werf, and Sjaak Brinkkemper. Forging high-quality user stories: towards a discipline for agile requirements. In *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*, pages 126–135. IEEE, 2015.
- [50] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn EM van der Werf, and Sjaak Brinkkemper. Improving agile requirements: the quality user story framework and tool. *Requirements Engineering*, 21(3):383–403, 2016.
- [51] Walid Maalej and Hadeer Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *2015 IEEE 23rd international requirements engineering conference (RE)*, pages 116–125. IEEE, 2015.

- [52] Viljan Mahnič and Tomaž Hovelja. On using planning poker for estimating user stories. *Journal of Systems and Software*, 85(9):2086–2095, 2012.
- [53] Eduardo Miranda, Pierre Bourque, and Alain Abran. Sizing user stories using paired comparisons. *Information and Software Technology*, 51(9):1327–1337, 2009.
- [54] Mante S Nieuwland and Jos JA Van Berkum. The neurocognition of referential ambiguity in language comprehension. *Language and Linguistics Compass*, 2(4):603–630, 2008.
- [55] Ayan Nigam, Neeraj Arya, Bhawna Nigam, and Deepika Jain. Tool for automatic discovery of ambiguity in requirements. *International Journal of Computer Science Issues (IJCSI)*, 9(5):350, 2012.
- [56] Frauke Paetsch, Armin Eberlein, and Frank Maurer. Requirements engineering and agile software development. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*, pages 308–313. IEEE, 2003.
- [57] Klaus Pohl. The three dimensions of requirements engineering: a framework and its applications. *Information systems*, 19(3):243–258, 1994.
- [58] Klaus Pohl and Nelufar Ulfat-Bunyadi. The three dimensions of requirements engineering: 20 years later. In *Seminal Contributions to Information Systems Engineering*, pages 81–87. Springer, 2013.
- [59] Linda Rosenberg, Theodore F Hammer, and Lenore L Huffman. Requirements, testing and metrics. In *15th Annual Pacific Northwest Software Quality Conference*. Citeseer, 1998.
- [60] Sylvain Schmitz. An experimental ambiguity detection tool. *Electronic Notes in Theoretical Computer Science*, 203(2):69–84, 2008.
- [61] Roger W Schvaneveldt, David E Meyer, and Curtis A Becker. Lexical ambiguity, semantic context, and visual word recognition. *Journal of experimental psychology: human perception and performance*, 2(2):243, 1976.
- [62] Wilfrid Sellars. Presupposing. *The Philosophical Review*, 63(2):197–215, 1954.
- [63] Unnati S Shah and Devesh C Jinwala. Resolving ambiguities in natural language software requirements: a comprehensive survey. *ACM SIGSOFT Software Engineering Notes*, 40(5):1–7, 2015.
- [64] Bayan Abu Shawar and Eric Atwell. Different measurements metrics to evaluate a chatbot system. In *Proceedings of the workshop on bridging the gap: Academic and industrial research in dialog technologies*, pages 89–96. Association for Computational Linguistics, 2007.

- [65] Ian Sommerville, Peter Sawyer, and Stephen Viller. Viewpoints for requirements elicitation: a practical approach. In *Requirements Engineering, 1998. Proceedings. 1998 Third International Conference on*, pages 74–81. IEEE, 1998.
- [66] Tor Stålhane and Tormod Wien. The dotd tool applied to sub-sea software. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pages 420–427. IEEE, 2014.
- [67] R Subha and S Palaniswami. Quality factor assessment and text summarization of unambiguous natural language requirements. In *Advances in Computing, Communication, and Control*, pages 131–146. Springer, 2013.
- [68] Sri Fatimah Tjong. Avoiding ambiguity in requirements specifications. *no. February*, 2008.
- [69] Sri Fatimah Tjong and Daniel M Berry. The design of sree—a prototype potential ambiguity finder for requirements specifications and lessons learned. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 80–95. Springer, 2013.
- [70] Yves Wautelet, Samedi Heng, Manuel Kolp, and Isabelle Mirbel. Unifying and extending user story models. In *International Conference on Advanced Information Systems Engineering*, pages 211–225. Springer, 2014.
- [71] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.
- [72] Alistair Willis, Francis Chantree, and Anne De Roeck. Automatic identification of nocuous ambiguity. *Research on Language and Computation*, 6(3-4):355–374, 2008.
- [73] Hui Yang, Anne De Roeck, Vincenzo Gervasi, Alistair Willis, and Bashar Nuseibeh. Analysing anaphoric ambiguity in natural language requirements. *Requirements engineering*, 16(3):163, 2011.
- [74] Hui Yang, Alistair Willis, Anne De Roeck, and Bashar Nuseibeh. Automatic detection of nocuous coordination ambiguities in natural language requirements. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 53–62. ACM, 2010.
- [75] Zheyang Zhang. Effective requirements development—a comparison of requirements elicitation techniques. *Software Quality Management XV: Software Quality in the Knowledge Society*, E. Berki, J. Nummenmaa, I. Sunley, M. Ross and G. Staples (Ed.) *British Computer Society*, pages 225–240, 2007.
- [76] Shaojun Zhao and Dekang Lin. A nearest-neighbor method for resolving pp-attachment ambiguity. In *International Conference on Natural Language Processing*, pages 545–554. Springer, 2004.

A Considered platforms and frameworks for the chatbot artifact

Platform/Framework	Advantages	Disadvantages
Microsoft Bot Framework	Allows flows Allows custom code Highly customisable Well documented No costs (for this research)	Has to be fully developed using code Complex Azure environment
Flow.ai	Supports flows Allows limited custom code Drag and drop to create bot Some documentation	Limited customisability Costs
FlowXO	Supports flows Drag and drop to create bot Nice interface	No customisation Costs
Diaglogflow	Supports flows Custom code via intents	Costs Intents do not seem to allow customisation

C All dialog

Process	Response	Response 2	Response 3	Response 4
Introduction & request name	Hello there, I am ReqBot and I will assist you in communicating your wishes for and problems with <code>{appName}</code> How can I call you?	What is your name?		
Explain ReqBot's capabilities	Okay <code>{user.Name}</code> , I can help you in two ways: First to communicate a new idea or functionality for <code>{appName}</code> and second to request a change in <code>{appName}</code> if for example something doesn't work (the way you want it).	There are two ways in which I can help you: One is to communicate a new idea or functionality for <code>{appName}</code> and the other is to request a change in <code>{appName}</code> if for example something doesn't work (the way you want it).		
Provide choice "Suggest an idea" / "Request a change"	Do you want to suggest something new or request a change in the app?			
Request component regarding change	Okay a change it is. What part of <code>{appName}</code> you want to change? Oh it seems like the part you intended was not on the list. What specific part do you want to change?	All right, what part do you want to change? Please name the specific part of <code>{appName}</code> you want to change.		
Request means	All right. Please describe in one sentence what you want. All right. Please describe what changes you want in the <code>{ExistingFeature}</code> in one sentence So you want to make a change in <code>{Means}</code> . Please retype it the way you want it All right, thanks.	Please write one sentence that describes what you want. Noted it.	Thank you, I got it.	Okay, I got that.

	All right, I changed **{OldMeans}** into **{Means}**.			
Request ends	Please describe why you want this {ReqType} in one sentence. So you want to make a change in **{Ends}**. Please retype it the way you want it. All right, thanks. All right, I changed **{OldEnds}** into **{Ends}**.	Please describe what benefit this {ReqType} would bring? It appears that **{Ends}** was not what you meant. Please type it the way you intended it. Noted it.	Thank you, I got it.	Okay, I got that.
Request role	What is your role regarding the system? So you want to make a change in **{Role}**. What is your role? Oh it seems like the part you intended was not on the list. What role do you have? Oh it seems like the part you intended was not on the list. What role do you have? What is your role regarding the system in one or two words? All right, thanks. All right, I changed **{OldRole}** into **{Role}**. All right, thanks. All right, I changed **{OldRole}** into **{Role}**.	Please describe your role regarding the system in one or two words? Noted it. Noted it.	Thank you, I got it. Thank you, I got it.	Okay, I got that. Okay, I got that.

Combine means ends and role (+ component)	So summarising, you told me the following {ReqType} : {CompleteUserStory}. {Enter} Do you want to submit this or do you want to make some changes?	I got the following information concerning the {ReqType} from you: {CompleteUserStory}. {Enter} Do you want to submit this or do you want to make some changes?		
Request what should be changed	Okay, what part do you want to change?	So, which part shall we change?		
Provide information on requirement	Thank you {user.Name}! With this code {UserStoryCode} you will be able to track your requirement on our forum.	All right {user.Name}! You can track this {ReqType} on our website with the code {UserStoryCode}.		
Request notification options	Do you want us to keep you posted about the progression of your requirement by mail? The last time you used the setting {user.NotificationOption} , do you want to keep this setting or change it in which case you will be mailed? The last time you used the setting {user.NotificationOption} , do you want to keep this setting or change it in which case you will not be mailed? Okay, we will not mail you. All right {user.Name}, what is your email address? We will keep you posted on {user.Email} I am sorry to say but {user.Email} is not an email address, maybe you made a typo. Please retype your email address	All right, we shall not mail you To which email address shall we send the updates? {user.Email} can not be an email address, maybe you made a typo. Please retype your email address.		

Request entering another requirement / end conversation	Do you want to submit other new ideas and changes or shall we end this conversation?			
Closing text	Thank you for participating {user.Name}. You can hand the laptop back.			
Request disambiguation vagueness	I noticed that you used the word {DetectedVagueness} , which can have several meanings or measures. To help us understand your intent as good as possible, could you try to specify it? If possible, please try to use units. For instance, fast could be specified by saying 0,5 sec or 500 ms. Thank you.			
Request disambiguation referential	I noticed that you used the word {DetectedTriggerReferential} , which can refer to several words. To help me understand your intent as good as possible, please tell me to what word {DetectedTriggerReferential} refers Apperantly I did not list to what {DetectedTriggerReferential} does refer. Please tell me to what {DetectedTriggerReferential} does refer. Thank you for specifying this.			

D Scenarios

Scenario: Google Maps

Person: Bran

Email: Bran@fictional.com

This scenario concerns a fictional person named Bran. He has just moved to Utrecht and he knows almost nothing of the city. He does not own a car and likes to walk to most places. If the walk takes too long, he will take his bike. To plan routes, Bran uses Google Maps generally on his phone and sometimes on his computer. Below, you can find 5 challenges Bran experienced while using Google Maps:

- Bran has used Google Maps a lot in the past and he intends on using it to explore this new city. To get to know the basics of the city he wants to take a long walk along important buildings, parks, streets and monuments in the city. Structured as Bran is, he wants to plan and plot the route in google maps. He opens his laptop and looks for an option to automatically generate a route along all important places, but he cannot find such an option. So he painstakingly searches the web for these places and plots them afterwards in Maps.
- While looking at the map, he cannot read all of the street names. Bran does not want to zoom in to enlarge them as that would lower his overview. He thinks there should be a way to increase or double the size of these characters.
- After some months, he has explored most of the city with Maps always recording his location. Bran wonders which areas he has not been yet. He opens his laptop and searches for a visualisation on Maps, but sadly such a functionality did not exist.
- Bran decided to create a walking route through the city, but he wanted to avoid streets with cars as he wants to bring his dog along for the trip, but he could not find which streets allow cars.
- Later that week he decided to pick up cycling again. He always hated crowded intersections and traffic lights as these slow him down. If there was only a way to create a route that avoid these as much as possible.

Scenario: Facebook

Person: Olivia

Email: Olivia@fictional.com

This scenario concerns a fictional person named Olivia. She is a very active Facebook user and only uses her phone. She has friends in multiple countries and uses Facebook to stay connected with them. Below, you can find 5 challenges Olivia experienced while using Facebook:

- One day Olivia sees a post that she immensely disagrees with. She wants to downvote the post or give it a thumbs down as is possible on many other websites, but then she remembers that this is not yet possible on Facebook.
- This also got her thinking about likes and the posts or pictures she liked. An overview of these positive posts might bring her in a better mood. She wonders if she can see a list of those again, but she can't find a way to do this.
- Olivia gets more posts from pages she liked and posts from reactions from her friends now, which becomes a bit tiring. She only wants to see posts placed by her friends. She started wondering whether she can change her Facebook homepage in a way that she only sees direct posts from her friends. Maybe a solution will come in the near future.
- While uploading a photo to Facebook, Olivia wanted to draw something on the photo to make her point clear. While she could apply filters and add text among others, drawing was to her discontent not available.
- Her next post would be to all her friends, but some can't speak English very well. She wondered if she could write and send her post in multiple languages, so that everyone of her friends would get a version they could read. Sadly, there was no function to accomplish this.

Scenario: Youtube

Person: Tim

Email: Tim@fictional.com

This scenario concerns a fictional person named Tim. He uses Youtube almost every day both on his phone and computer. He watches a great deal of videos and uses Youtube to listen to music a lot. Below, you can find 5 challenges Tim experienced while using Youtube:

- When he is listening to music on his phone, one thing still bothers him. Everytime he turns his screen off to put his phone in his pocket, the video with the music stops playing.
- When Tim has no wifi connection and watches videos on his phone, they tend to load slow as he lives in an area with poor connectivity. Tim wonders if the video could not start loading while the ads are playing. Now, ads load and he still needs to wait for the rest of the video to load, which bothers him.
- Tim has some difficulty finding some of the content he likes sometimes, namely educative videos. He tried to use the categories in Youtube, but there was only a handful of them and none on this topic so he closed his laptop.
- Even if Tim finds a selection of possibly entertaining videos, he wants to know how well they get rated by other youtubers, so he has an indication on the quality. He knows that he can click on a video and see the likes and dislikes, but shouldn't that be made available earlier?
- As Tim is getting older, he notices that he finds it harder to read the titles of videos. He does not yet want to get glasses, but maybe reading would be easier with slightly larger text.

Scenario: Thuisbezorgd.nl

Person: Roger

Email: Roger@fictional.com

This scenario concerns a fictional person named Roger. He uses Thuisbezorgd.nl quite often, as he has little time to cook. He always uses his smartphone to order and tries to eat a bit diverse. Below, you can find 5 challenges Roger experienced while using Thuisbezorgd.nl:

- Roger generally orders from 3 different locations as he does not always eat at home. He hates to retype each address every time and that gets him thinking: would it not be great if I can just save a few of these addresses?
- Sometimes ordering takes longer than expected, but as someone with sometimes little time to eat this can be a problem for Roger. He would not mind to pay extra if there was an option to have his order rushed.
- One thing that bothers Roger a lot is that even though there is a vast list of categories for restaurants, there is no way to search for one specific dish. If he wants that dish, he now would have to search sometimes multiple restaurants to find it. There should be some function to fix this he thinks.
- Roger is allergic to peanuts, cashew nuts and apples. This makes eating food that he did not cook himself always a bit risky. He thinks the app could make use allergic information to better their service. In the Thuisbezorgd.nl app, he searches for a form or place in which he can register all his allergies, but there is nothing like this available.
- The allergies part got Roger thinking, if the app has his allergies it could help him a lot. Now he sometimes orders a dish that contains something he is mildly allergic to without this being his intention. With his allergies, the app would be able to warn him before he places the order.

Scenario: Open Spotify (Not used during the experiment)

You are a general user of Spotify, you have the app on your phone and Spotify installed on your laptop. On most days you listen to music via the app, oftentimes multiple hours a day. Since a week or two, you also use Spotify at your association. The music boxes there can only be connected to one specific desktop, which won't allow you to install programs such as Spotify. As a workaround you started using the web version of it called Open Spotify (<https://open.spotify.com/>). However as you used it, you started having some irritations on this version of Spotify.

The first time you used it, it was a bit confusing. It seemed like you could play music, however that did not work because you had to be logged in. That seemed reasonable, but when you clicked on the login button, you were transported to another site. It looked different in style and layout but had a lot of spotify logos and the URL seemed alright too, so you logged in. Open Spotify looked fairly similar to the program on your laptop, but some buttons and options were in different places, resulting in you having to search for them. While searching, you noticed that you had to scroll a lot since only a bit of the content could be on the screen at once and the content on screen was quite large. You wanted to let people hear some of the remixes that you made at home and added to your own playlists. You added them to your spotify so that you would not have to switch apps while listening to music. However, your remixes were nowhere to be found on Open Spotify. At some point in time, you left the association and went home forgetting to sign out on Spotify. The next week, you started listening to your weekly recommended music at home and noticed a lot of strange music in it. It was strange because it was not a genre you listen to at all, could this be due to the music people listened that day?

E Experiment - scenario and condition planning

Scenario \ Method	Form	Chatbot	Chatbot Ambiguity detection
Maps	1	5	9
Facebook	2	6	10
Youtube	3	7	11
Thuisbezorgd	4	8	12

Participant no	First scenario & method	Second scenario & method
1	1	6
2	9	2
3	11	4
4	3	10
5	8	1
6	4	10
7	12	2
8	1	6
9	8	3
10	2	9
11	10	3
12	4	11
13	8	1
14	3	10
15	7	4
16	3	6
17	7	1
18	4	6
19	7	2
20	2	5
21	10	1
22	3	12
23	8	3
24	4	6
25	5	3
26	2	7
27	5	4
28	2	11
29	11	2
30	4	5
31	5	2
32	2	9
33	9	3
34	1	7
35	9	4
36	1	12
37	12	1
38	1	8
39	11	4
40	3	12

F Experiment approach

1. Chatbot and form are set up
2. Welcome the participant
3. Basic explanation of the experiment
 - What are we researching?
 - What is a chatbot?
 - You will get 2 scenarios and for each a different condition
4. Offer the participant an informed consent form
5. Offer the participant the general introduction form
6. Ask if the participant has any questions so far
7. Give the participant one scenario and the tools to use the first condition (laptop with chatbot or online form)
8. Participant works through the scenario using the first condition
9. Upon completion, the participant receives the SUS questionnaire
10. Upon completion, the participant receives the explanation of the second condition
11. Steps 7 to 9 are repeated with the second condition
12. Upon completion, the experiment will end and the participant is thanked.

G Experiment introduction

General introduction

Dear participant,

Over the next 20 to 40 minutes, you will be creating requirements and entering them via two conditions. The first one is described on the bottom of the page, the other you will receive later. For each condition you will be given a scenario from a relatively well-known system or app. It does not matter if you are not familiar with this particular system. The scenario will describe a person who uses this system and the challenges that this person has experienced. It is your goal to act on that person's behalf and imagine what changes or additions that person would want in that specific system.

Example: This scenario describes John, who uses an app that shows different maps of his country. However, John is colourblind and thus is not able to distinguish all colours on the map.

You should imagine what John's problem is, what he wants why. In this case, he cannot use the app properly due to the colours. He would want (1) to add a colourblind mode or (2) to change the colours the map uses, so that he can use the map even though he is colourblind. In this case there are at least two options, you only ever need to provide one per situation.

You use the condition to write these changes and additions down and send them. You act on behalf of the person in the scenario, so that means that for any question concerning personal information, you should use the information of that person. So in this case if your name is requested, you should answer "John".

After you finished each scenario, please contact the observer, you will receive a questionnaire considering your experience with that given condition.

If you have any questions, feel free to ask them.
Thank you for participating in this research.

Condition: Form

You will be given a laptop with a google form containing several questions. It is your task to answer these questions based on the scenario you receive. You will be specifying a new idea for the system by answering 5 questions. In this context, the idea is one of the desired changes or new functionalities the person from the scenario would want. These changes or additions should always include at least these 3 parts: what the fictional person wants; why the fictional person wants this; and what type of user the fictional person is (general user, mobile user, student, teacher, etc.).

Condition: Chatbot

You will be interacting with a chatbot. The chatbot will be asking you questions and by answering these questions, you will be able to communicate the desired changes and/or additions from the scenario. The chatbot communicates in English and thus also expects your responses to be English. In the unlikely event the chatbot gets stuck, please contact the observer.

H Results - significance values

	Usability	Restricted usability	Learnability
Form - Chatbots combined	0.590	0.561	0.617
Form - ReqBot regular	0.231	0.231	1.000
Form - ReqBot+	0.778	0.776	0.560
ReqBot regular - Chatbot +	0.532	0.523	0.873

Table 7: Significance values (p-values) for usability

Condition	P-value
Form - Chatbots combined	0.559
Form - ReqBot regular	0.565
Form - ReqBot+	0.610
ReqBot regular - Chatbot +	0.133

Table 8: Significance values (p-values) for Correctness