Electronic version of an article published in **Proceedings of the International Conference on System and Software Engineering and their Applications, December 5-8, Paris, France (ICSSEA'00)**

Taming Ambiguity in Natural Language Requirements

Erik Kamsties and Barbara Paech
Fraunhofer Institute for Experimental Software Engineering
Sauerwiesen 6, D-67661 Kaiserslautern, Germany
Phone: +49 6301 707 219 Fax: -200
{kamsties, paech}@iese.fhg.de

Abstract

One of the main tasks of requirements engineering (RE) is the creation of a requirements document that precisely, consistently, and completely describes the functional and non-functional properties of the system to be built. At some point during the RE process, the requirements are written down using a natural language or a requirements specification language. On one hand, natural language is flexible, universal, and wide-spread. On the other hand, natural language requirements are recognized widely as being incomplete, inconsistent, and inherently ambiguous. Semi-formal and formal requirements specification techniques, such as UML [13] or SCR [5], have been proposed to overcome these deficiencies. Completeness and consistency of these specifications can be tackled to some degree mechanically by tools. However, as a recent study shows, ambiguity rarely surfaces during the development of a requirements model [10]. Thus, since a requirements specification technique enforces precision, the resulting requirements model becomes unambiguously wrong.

In this paper, we show how to detect ambiguities in natural language requirements using a checklist. We distinguish between linguistic and RE-specific ambiguities. *Linguistic ambiguities* are those ambiguities that are usually discussed in RE textbooks such as ambiguous pronoun references. *RE-specific ambiguities* occur with respect to the RE context, which includes the application domain and the system domain. For example, the requirement

(1) If the bank customer maintains a minimum balance in his or her checking account, there is no monthly service charge.

is ambiguous not because of its linguistic representation, but because of the operational environment and application domain. In the requirements documents that we have investigated, RE-specific ambiguities account for the majority of ambiguities, while pure linguistic ambiguities played a less significant role.

In this paper, we present also an approach that allows identifying types of RE-specific ambiguity from a metamodel, e.g., of a requirements specification technique. Based on the identified types of ambiguity, we have developed an improved inspection technique for natural language requirements on the basis of checklists and scenario-based reading.

Keywords: Natural language requirements, ambiguity, metamodels, requirements specification techniques, reading techniques, inspections

1 Introduction

The requirements are usually stated in natural language in order to allow all stakeholders to read and understand what shall be delivered. The use of natural language to state requirements has several benefits. Natural language is universal; any type of requirement in any kind of application domain can be described. It is flexible; requirements can be described rather abstract or quite detailed. Finally, it is wide-spread; everyone can read and write such requirements.

However, natural language has one major drawback when it is used to state requirements, its inherent ambiguity. Words can have several meanings, for instance the 500 most used words in English have on average 23 meanings; phrases and whole sentences can be interpreted in more than one way.

Ambiguous requirements are a serious problem in software development, because often stakeholders are not aware that there is an ambiguity. Each gets from reading the requirements an understanding which differs from that of others, without recognizing this difference. Consequently, the software developers design and implement a system that does not behave as intended by the users while the software developers honestly believe they have followed the requirements. Also, components fail to interact properly, because the same requirement was allocated to different components, and the different developers have interpreted it differently.

Even if the software developers recognize the ambiguity, an ambiguous requirement can be a problem, because it may lead to the impression that the ambiguity was deliberately introduced, and they can decide on what is the most appropriate interpretation. This is because ambiguity is not only a drawback of natural language, but it is also a feature; it is used to leave issues deliberately open, for instance, because they are considered design decisions. However, whether or not an ambiguity was deliberately introduced can be answered only by the author of the requirement. The developer cannot deduce this from the linguistic expression of a requirement.

Ambiguity is not merely a linguistic problem that occurs in RE, since requirements are written usually in natural language. Actually, ambiguity has also a RE-specific dimension. We call a requirement RE-specifically ambiguous, if it allows several interpretations with respect to other requirements, the application domain (e.g., applicable standards, operational environment), and the system domain (e.g., conceptual models of software systems and their behavior). In the Requirement 1 shown in the abstract, the balance on the checking account increases when the monthly salary is deposited and then goes down steadily until the next salary is deposited. Therefore, it is ambiguous whether the average balance for one month must be over the minimum or the balance every single day must be over the minimum.

In the requirements documents that we have investigated, RE-specific ambiguities account for the majority of ambiguities, while pure linguistic ambiguities played a less significant role. It should be mentioned that missing requirements or missing parts of requirements, i.e., omissions, can lead to ambiguities in known requirements. This relationship should be utilized for detecting omissions, which are usually more difficult to detect than commissions. By analyzing requirements for ambiguity, we can increase the chance of detecting omissions.

The most prominent solution to the problem is the use of semi-formal or formal requirements specification languages such as UML [13] or SCR [5], rather than natural language. These languages have a well-defined semantics thus the degree of ambiguity in requirements is at least significantly diminished if not eliminated.

However, requirements specification languages can solve the problem only if all stakeholders can write requirements using these languages. Usually, this is not the case, the initial requirements are written in natural language, and these initial requirements are translated later into a formal notation by a requirements engineer. During this translation, ambiguous informal requirements become unambiguously wrong statements in the requirements specification language if they were misinterpreted. This would not be a problem if the resulting requirements model can be rigorously validated with the customer. In practice, however, stakeholders refrain from reading requirements written in artificial languages. Simulation, the other way of validating requirements models, suffers from the same limitations as testing. One can show the presence of defects but not their absence.

More solutions to the problem of ambiguous requirements have been proposed including glossaries, style guides for writing requirements, and inspections of requirements documents. These solutions suffer from at least one of the following three deficiencies:

- *Lack of acceptance by intended users and infeasibility*—style guides are accepted by practitioners, if at all, only when they are brief; extensive lists of rules are usually ignored. The most effective inspection approach that was suggested for detecting ambiguities is to hand requirements to several different stakeholders, to ask for an interpretation, and to compare these interpretations afterwards. If the interpretations differ, the requirements are ambiguous [4]. Obviously, this approach is feasible only for small sets of requirements.

- *Lack of specificity*—it is assumed that inspectors are able to detect ambiguities just by reading; no guidance is provided by the technique on how to find an ambiguity. Current inspection techniques relying on checklist-based reading are unspecific. There is usually one checklist item asking "is the requirement ambiguous?". Recall that the major problem of ambiguity is that one is not aware of it. Thus, simply asking whether there is an ambiguity is not much help.
- *Uni-dimensionality*—only the linguistic dimension of ambiguity is addressed, but the pragmatic dimension is not. Rupp and Götz [15] proposed a checklist for requirements that was derived from Neuro-Linguistic Programming (NLP). This checklist is specific. However, it does not address all types of linguistic ambiguity. Moreover, it is uni-dimensional. It helps to reveal linguistic ambiguities, but not RE-specific ones. Glossaries address only one type of ambiguity, namely lexical ambiguity.

The paper is structured as follows. First, the notion of RE-specific ambiguity is discussed in more detail. Then, we present a comprehensive checklist for ambiguities. Finally, we sketch an approach to tailor the checklist to a particular RE context in order to make it more specific.

2 RE-Specific Ambiguity

This section introduces the notion of RE-specific ambiguity, which arises when a requirement is interpreted with respect to the RE context. We motivate RE-specific ambiguity by a discussion of an analogy to linguistics and machine translation. The following sentence contains two ambiguities.

(2) John hit the wall.

The first one is a linguistic ambiguity, in particular, a lexical one that concerns *hit*. The word is ambiguous, because it has transitive senses, e.g., *to reach*, and intransitive senses, e.g., *to attack* or *to strike*. The second ambiguity arises for a German reader from *wall*. The word is translated into German as *Wand* if the wall is inside a building and it is translated as *Mauer* if the wall is outside. This type of ambiguity is a not a linguistic one. In machine translation, it is called conceptual translational ambiguity, and is by no means a rare phenomenon, even between closely related languages [6].

The same situation occurs in RE. The customers and users speak a different language than the requirements engineers and developers. As in Example 2, ambiguities can be already contained in the requirements as uttered by the customers or they can occur due to the translation into the requirements engineers' language. We call the translational ambiguities that occur due to the translation from a requirement in the customer's language to a requirement in a requirements engineer's language *RE-specific ambiguities*. RE-specific ambiguities require a background in RE and the particular project to be detected. The following example

(3) Aircraft that are non-friendly and have an unknown mission or the potential to enter restricted airspace within 5 minutes shall raise an alert.

contains two ambiguities. First, there is a linguistic ambiguity, in particular a semantic ambiguity, that concerns the relative priority of *and* and *or*, because we cannot assume priority rules of Boolean logic for natural language requirements. Second, there is a RE-specific ambiguity, that concerns the alert. Some requirements specification languages, such as UML, distinguish actions from activities, i.e, discrete from continuous operations. An action is executed when a transition fires, a state is entered, or a state is exited. An activity is an action that is performed while being in a state. The phrase *raise an alert* can be interpreted as an action or as an activity, which leads to two different UML state diagrams shown in Figure 1.

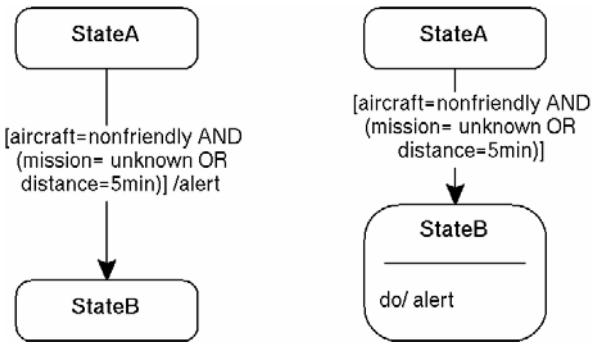

Figure 1. UML State Diagrams for Requirement 3

In the left state diagram, the phrase *raise an alert* is interpreted as discrete action, and in the right one it is interpreted as continuous activity.

3 A Checklist for Ambiguity

The current notion of ambiguity is quite general in RE, e.g. [14]. An ambiguous requirement is often simply defined as a requirement that can be interpreted in multiple ways [7]. The attempts that have been made to identify particular facets of ambiguity in RE are rather incomplete; only a few of the many types of ambiguity known in linguistics [2, 11] are mentioned in the RE literature. This section introduces our definition of ambiguity, which provides a checklist for detecting ambiguities.

We define ambiguity by a main definition and two supplementary definitions. The main definition defines an ambiguous requirement as a requirement that has more than one interpretation, but takes the RE context into account. The RE context is important, because every natural language requirement has the potential to be ambiguous. However, we consider only those requirements ambiguous, which cannot be disambiguated by a reader who has knowledge of the relevant RE context.

Definition 1. A requirement is ambiguous if it admits multiple interpretations despite the reader's knowledge of the RE context.

This definition helps to distinguish spurious claims of ambiguity from genuine ones. For example, the requirement *Generate a dial tone* may appear ambiguous to naive readers, because they do not realize that national standards exist that define the frequency of a dial tone.

Two supplementary definitions define the types of ambiguities that lead to multiple interpretations and the RE context for disambiguation, respectively.

Definition 2. The RE context that is required to disambiguate a requirement comprises of four parts, the requirements document, the application domain, the system domain, and the development domain.

This definition is inspired by Jarke's Four World model, which structures the RE context [8]. The application domain includes the world of the subject domain and the world of the usage environment. The system domain comprises of the world of the current and future information systems. The development domain is the environment in which the change of the other worlds takes place.

Actually, also real-world knowledge and language knowledge allow disambiguating an ambiguity. However, we assume that this knowledge is sufficiently shared by requirements authors and readers. Therefore, it is not included in the above definition.

The second supplementary definition aims at capturing the basic types of ambiguity that allow a reader to find multiple interpretations of a requirement.

Definition 3. A requirement allows multiple interpretations if it contains linguistic or RE-specific ambiguity. Linguistic ambiguity arises independently from any context and comprises vagueness and genuine ambiguity in the form of lexical, syntactic, and semantic ambiguity. RE-specific ambiguity arises from the RE context and comprises vagueness, generality, and genuine ambiguity in the form of lexical, requirements-document, application-domain, system-domain, and development-domain ambiguity.

Note that the RE context plays a dual role. It causes RE-specific ambiguity (Definition 3) and it disambiguates linguistic ambiguity (Definition 2). We use linguistic terms in Definition 3 in order to keep it short. The types of ambiguities are explained below.

Genuine Ambiguity. A requirement is genuinely ambiguous if it has a discrete number of interpretations, no general meaning is available which covers the distinct readings, and clarification is required to make sense of it. Genuine ambiguities can be further distinguished into lexical, syntactic, semantic, and genuine RE-specific ambiguities. The requirement

*(4) The customer enters a card and a numeric personal code. If **it** is not valid then the ATM rejects the card.*

is a clear case of genuine ambiguity. It is a referential ambiguity; exactly two distinct interpretations are available, *it* refers either to *card* or to *personal code*, and there is no general interpretation that covers the two interpretations.

Vagueness. A requirement is vague if it has a continuum of possible interpretations, a general meaning that covers these readings is available, and borderline cases exist. Borderline cases are interpretations for which it is uncertain whether they are included in the requirement or not. Furthermore, a vague requirement cannot be made precise; it is vague by its nature. Note that there can be linguistic and RE-specific vagueness. Vagueness can arise from adjectives and adverbs, e.g., *fast*, *user-friendly*, verbs, e.g., *support*, *process*, and also nouns, e.g., *user*. In the requirement

(5) Create a means for protecting a small group of human beings from the hostile elements of their environment.

the phrase *small group* is vague. A continuum of interpretations is available 3, 10, 50, 200, etc.. Normally, in a conversation, there is a general meaning available for *small* and thus it can be accepted in a conversational exchange. Borderline cases exist. When a group of 100 people is considered small, it is difficult to argue that a group of 101 people is not small.

Generality. A requirement is general if it has a continuum of possible interpretations, a general meaning that covers these readings is available, but borderline cases do not exist, and it can be made precise. Furthermore, the choice of an interpretation is not the subject of design or implementation activities. This additional criterion is necessary for RE, since requirements have to be design and implementation free; all requirements should be general, not vague, from the viewpoint of a designer or programmer. We consider a requirement general, if it has borderline cases, but it can be made precise. Note that there can be linguistic and RE-specific generality. The latter is more frequent. Generality is not bound to specific language patterns.

Requirement 5 is also general. The phrase *hostile elements* allows for a continuum of interpretations, for instance wind or rain, and there is a general meaning available which is *hostile elements*. Thus, it can be accepted in a conversational exchange. Borderline cases do not exist. The phrase *hostile elements* can be made precise by listing the hostile elements.

The distinction into ambiguity, vagueness, and generality is not mutually exclusive. Some requirements fall into two or even all three categories.

Lexical Ambiguity. A requirement is lexically ambiguous if a word in it has several meanings. Lexical ambiguity can arise from virtually any word. Each of the 500 most used words in English has, on average, 23 meanings [3]. In addition, a word can have special meanings within a particular application domain. The requirement

(6) For up to 12 aircraft, the small display format shall be used. Otherwise, the large display format shall be used.

is lexically ambiguous. The phrase *up to 12* can mean *including* or *excluding 12*.

Polysemy. A word is polysemous when it has several related meanings and one etymology. Polysemies are often of a RE-specific nature. They are a much larger problem in requirements documents than homonyms. The meanings of a polysemy are related, thus, more detailed contextual information is necessary to disambiguate it. Polysemies can occur despite of a glossary definition of that term, because shifts in the meaning of terms occurred during a project or terms are not defined that precisely. A *systematic polysemy* applies to a class of words. The volatile-persistent ambiguity, for instance, arises when a word of a requirement either refer to a volatile or persistent property of an object. In the requirement

(7) When the user presses the L- and R-button simultaneously, the alarm is turned off.

the phrase *turned off* can refer to an alarm that is currently sounded by the system or to the general ability of the system to raise alarms.

Syntactic Ambiguity. A requirement is syntactically ambiguous if words or phrases in it can play several grammatical roles and thus can be put together in several ways to form a correct sentence. In the terminology of compiler construction, syntactic ambiguity occurs when the requirement sentence has several parses. Syntactic ambiguity arises from: prepositional phrases as shown below; relative clauses, introduced by subordinating conjunctions, such as *when* and *which*; and ellipses, i.e., when a lexically or syntactically necessary part of requirements sentence is omitted. The requirement

(8) The product shall show the weather for the next twenty-four hours.

is syntactically ambiguous. The prepositional phrase *for the next twenty-four hours* can be attached to the verb *show* or the noun *weather*.

Semantic Ambiguity. A requirement is semantically ambiguous if there are several ways of combining the meanings of its words into sentence meanings. If the sentence's meaning is expressed in predicate logic, if different logical forms can be derived for the sentence, then it is semantically ambiguous. Semantic ambiguity arises from: coordinating conjunctions, i.e., *and* and *or*; quantifiers, e.g., *a*, *all*, *each*, *every*, *some*; and negations, e.g., *not*. The requirement

(9) Aircraft that are non-friendly and have an unknown mission or the potential to enter restricted airspace within 5 minutes shall raise an alert.

is semantically ambiguous. The question is here whether *and* or *or* takes precedence over the other.

Genuine RE-specific Ambiguity. A requirement is RE-specifically ambiguous if it has a discrete number of interpretations in relation to the RE context. Following Definition 2, we divide genuine RE-specific ambiguity further into requirements-document ambiguity, application-domain ambiguity, system-domain ambiguity, and development-domain ambiguity. Language patterns that let arise genuine RE-specific ambiguity are not described in the literature; only examples are provided. The requirement

(10) If the timer expires before receipt of a [T-DISCONNECT indication], the SPM requests [transport disconnection] with a [T-DISCONNECT request]. The timer is cancelled on receipt of a [T-DISCONNECT indication].

is RE-specifically ambiguous. The ambiguity arises from the system domain. It is ambiguous whether or not the second sentence is part of the if-statement in the first sentence. This particular requirement could be disambiguated by RE context knowledge; the cancellation of an expired timer probably makes little sense.

Requirements-Document Ambiguity. Requirements-document ambiguity occurs because a requirement allows several interpretations with respect to what is known about other requirements in the requirements document. Requirements document ambiguity can arise from pronoun references, e.g., *it*, and definite noun phrases like the one below. The requirement

(11) The product shall show all roads predicted to freeze

is a requirements-document ambiguity. The definite noun phrase *roads* can refer to more than one set of roads that are specified earlier in the requirements document.

Application-Domain Ambiguity. Application-domain ambiguity occurs because a requirement allows several interpretations with respect to what is known about the application domain. The requirement

(12) Shut off the pumps if the water level (in the tank) remains above 100 meters for more than 4 seconds.

is an application-domain ambiguity. The tank is built into a vessel, which is affected by wind and waves, thus the water level of the tank can vary over the 4-s period. Consequently, one can find four interpretations for the water level: the *mean, median, rms (root mean square),* or *minimum* water level over the past 4 s remains above 100m. Another application-domain ambiguity is Requirement 1.

System-Domain Ambiguity. System-domain ambiguity occurs because a requirement allows several interpretations with respect to what is known about the system domain. The requirement

(13) All CTC device command messages transmit two 8-bit bytes. The meaning of the 16 bits of information received from the wayside computer is as follows: Byte 0, bit 0 when this track section is occupied, Byte 1, bit 0 ..., etc.

is a system-domain ambiguity. The ambiguity here lies in the numbering of bits and bytes. They can be numbered from left to right or right to left. This ambiguity requires the reader to have contextual knowledge of software/hardware interfaces, otherwise it is not observable. Requirement 10 is another instance of this type of ambiguity.

Development-Domain Ambiguity. Development-domain ambiguity occurs because a requirement allows several interpretations with respect to what is known about the development domain.

This section presented a new definition of ambiguity, which improves the current definitions of ambiguity in RE in two ways. First, our definition provides a comprehensive list of types of ambiguity and describes the language patterns that cause these types of ambiguity; linguistic and RE-specific types of ambiguity are described. Second, the definition defines the RE context required for disambiguation. Thus, genuine claims of ambiguity can be separated from spurious ones.

4 Refinement of Checklist Using a Metamodel

The checklist presented in the last section describes linguistic ambiguities and their language patterns in detail. However, it describes RE-specific ambiguities rather abstract and leaves their language patterns open, because they depend on the particular RE context in which an RE process takes place. Therefore, the checklist needs to be tailored to that particular RE context.

The idea is to describe types of linguistic and RE-specific ambiguities in more detail by using metamodels. A metamodel defines a language for describing requirements models. An ambiguity in a natural language requirement leads to different requirements models written using a requirements specification technique, as Requirement 3 and Figure 1 have shown. Therefore, the idea is, by changing the abstraction level, to identify and describe *types* of ambiguity from the *metamodel* of this requirements specification technique.

Figure 2 provides an example of a metamodel. The figure depicts a part of the UML metamodel for statechart diagrams. The particular concept types (i.e., the classes in UML terminology) and relationships (i.e., the associations) are depicted, which are used in the requirements models of Requirement 3 shown in Figure 1.

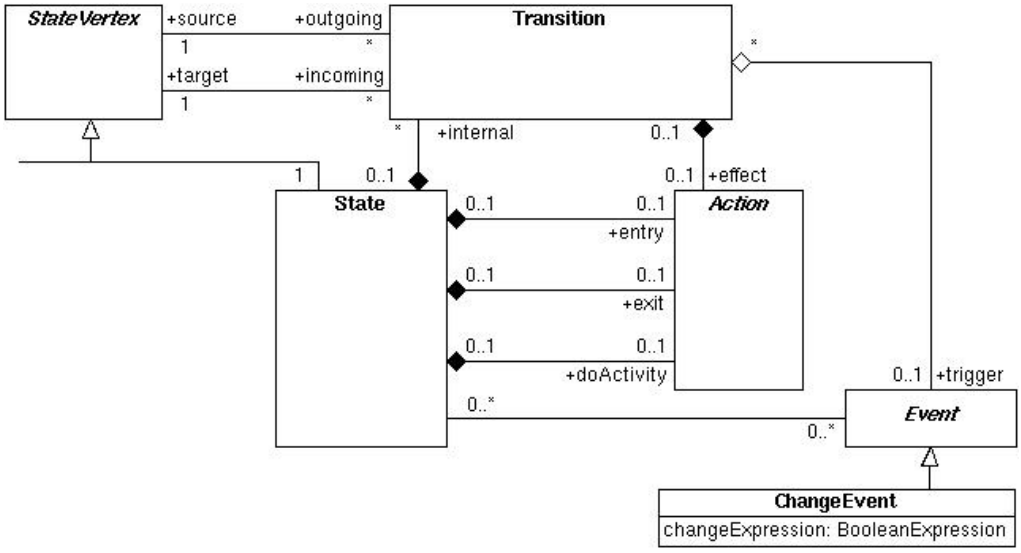

Figure 2. Part of UML Metamodel for Statechart Diagrams

We have developed a set of four heuristics to systematically identify and document types of RE-specific ambiguity in a metamodel. Each heuristic has a schematic part that can be automated and a human-based part. The schematic part is the investigation of metamodels by a human. The essentially human-based part is the search for ambiguous language patterns. The heuristics cannot automate the latter part, because there exists no metamodel of unbounded natural language. One heuristic, for example, suggests considering pairs of concept types in which one concept type can play several roles in alternative relationships to the other one. The investigation of the UML metamodel in leads to the identification of the previously described ambiguity regarding State and Action in Figure 1. The requirements engineer has to search for language patterns that could cause this ambiguity; the heuristics rely on the experience and language skills of the user in this respect. The search for ambiguous language patterns leads to that of a verb that expresses a functionality of the system to be built, as shown in Requirement 3. The result from the application of a heuristic is in this case a new subtype of system-domain ambiguity.

An *action ambiguity* concerns a verb and arises if it describes an action of a system, which can be interpreted as an action that is executed (1) when a state is entered, (2) when a state is exited, or (3) while being in a state. The following example shows an action ambiguity:

*(14) The system monitors the pressure and **sends** the safety injection signal when the pressurizer's pressure falls below a 'low' threshold.*

Sending the safety injection signal can be interpreted as an exit action that occurs when the 'low' state is exited, as entry action that occurs the state 'too low' is entered, or an action that is executed while being in that state. The result of the application of the heuristics is a more extensive checklist for ambiguities. The heuristics are described in detail in [9].

5 Summary and Contribution

This paper makes two contributions. First, a new understanding of ambiguity is presented in form of a checklist, and second, the idea of an approach is presented, which allows making the checklist for detecting ambiguity more prescriptive by using metamodels.

The RE literature provides an incomplete and partially inconsistent picture of ambiguity. We suggested in this paper a comprehensive definition of the various types of ambiguity, which covers linguistic and RE-specific ambiguities. The former are described in current definitions rather incomplete and the latter are mostly ignored.

A checklist is not the most effective technique for detecting defects. Scenario-based reading has been introduced as a more systematic technique for reading software artifacts [1]. The overall idea of scenario-based reading is to provide an inspector with an operational scenario, which requires him or her to first create an abstraction of the product, i.e., the requirements document in our case, and then answer questions based on analyzing the

abstraction with a particular emphasis or role that the inspector assumes. For example, the operational scenario requires the inspector to create test cases as an abstraction of the requirements document and a question could be "Do you have all information necessary to develop a test case?". If there is information missing then the inspector may have detected a defect in the requirements document. By creating an abstraction, the inspector gains a deeper insight into the requirements document than by reading it with a checklist.

We have developed a reading technique called ESBR, which is a reading technique for detecting ambiguities that consists of a checklist and an operational scenario. It can be applied for validating requirements documents of embedded systems. The technique addresses linguistic and RE-specific ambiguities, in particular discourse and system-domain ambiguities. The scenario was developed on the basis of the black-box from Cleanroom's Box Structure method [12]. In order to make black-box specifications more convenient, we supplemented it with notational elements from SCR [5].

This reading technique proved in controlled experiments to be more efficient than a pure checklist. Moreover, it helps detect more ambiguities than are detected when a requirements model is developed using a requirements specification technique [9].

References

[1] Victor R. Basili, Scott Green, Oliver Laitenberger, Filippo Lanubile, Forrest Shull, Sivert Sorumgard, and Marvin V. Zelkowitz. The empirical investigation of perspective-based reading. Journal of Empirical Software Engineering, 1(2):133–164, 1996.

[2] D. A. Cruse. Lexical Semantics. Cambridge Textbooks in Linguistics. Cambridge University Press, Cambridge, 1986.

[3] Jeff Gray. Collection of ambiguous or inconsistent/incomplete statements. Web page: http://www.vuse.vanderbilt.edu/ jgray/ambig.html, April 2000.

[4] Donald C. Gause and Gerald M. Weinberg. Exploring Requirements: Quality Before Design. Dorset House, 1989.

[5] Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Automated consistency checking of requirements specifications. ACM Transactions on Software Engineering and Methodology, 5(3):231–261, July 1996.

[6] W. John Hutchins and Harold L. Somers. An Introduction to Machine Translation. Academic Press, 1992.

[7] IEEE recommended practice for software requirements specifications. ANSI/IEEE Std. 830-1998, The Institute of Electrical and Electronics Engineers, New York, USA, 1998.

[8] Matthias Jarke. DAIDA: Conceptual modeling and knowledge based support of information systems development processes. Technique et Science Informatiques, 9(2):122–133, 1990.

[9] Erik Kamsties. Surfacing Ambiguity in Natural Language Requirement. PhD Thesis. University of Kaiserslautern, 2000.

[10] Erik Kamsties, Antje von Knethen, Jan Philipps, and Bernhard Schätz. Eine vergleichende Fallstudie mit CASE-Werkzeugen für formale und semi-formale Beschreibungstechniken. In Tagungsband des 9. GI/ITG-Fachgesprächs "Formale Beschreibungstechniken für verteilte Systeme", pages 103–112, München, Germany, June 17-18 1999. First Replication of 1998 Experiment.

[11] John Lyons. Semantics One and Two. Cambridge University Press, 1977. Two volumes.

[12] Harlan D. Mills. Stepwise refinement and verification in box-structured systems. IEEE Computer, June 1988.

[13] OMG Unified Modeling Language. Technical report, Rational Software Corporation, June 1999. Version 1.3, available at http://www.rational.com/uml.

[14] Suzanne Robertson and James Robertson. Mastering the Requirements Process. Addison-Wesley, 1999.

[15] Christine Rupp and Rolf Götz. Sprachliche Methoden des Requirements-Engineering (NLP) (in German). In Proceedings of the CONQUEST-1 First Conference on Quality Engineering in Software Technology, pages 117–126, Nürnberg, Germany, September 25-26 1997.