An Empirical Investigation of the Defect Detection Capabilities of Requirements Specification Languages

Erik Kamsties[1], Antje von Knethen[1], Jan Philipps[2], and Bernhard Schätz[2]

[1]Fraunhofer Institute for Experimental Software Engineering
Sauerwiesen 6, D-67661 Kaiserslautern, Germany
Phone: +49 6301 707 219 Fax: -200
{kamsties, vknethen}@iese.fhg.de

[2]Department of Computer Science, Technical University of Munich,
Arcisstraße 21, D-80290 München, Germany
Phone: +49 89 289 22398 Fax: -25310
{philipps, schaetz}@informatik.tu-muenchen.de

Abstract

It is a frequently reported effect of applying requirements specification languages that the formalization of informal requirements leads to the detection of defects such as omissions, conflicts, and ambiguities. However, there is little quantitative data available on this effect.

This paper presents an empirical study with requirements specification languages, which addresses two research questions. First, which types of defects are detected by a requirements engineer during the development of a requirements model, and second, what happen to those defects that are not detected?

The results indicate that ambiguities require special care during formalization, because they are less frequently reported than other types of defects. Instead, ambiguities tend to become often disambiguated unconsciously, which is a serious problem, because implicit assumptions are more likely than in our study to be wrong when the system is more complex. Moreover, ambiguities are misinterpreted more often than other types of defects. Finally, ambiguities, if noticed, require immediate clarification.

1. INTRODUCTION

The use of a requirements specification language (RSL) in requirements engineering (RE) has many-fold benefits. Precise requirements models allow for better communication among various stakeholders, checks of completeness and consistency and proofs of safety properties can be automated, and the dynamic behavior of the requirements models can be simulated. Furthermore, they make the RE process more repeatable than if ad hoc techniques were be applied. According to Sommerville and Sawyer [SS97], RSLs are a "vehicle for the analyst to add clarity to the fuzzy picture provided by the stakeholder requirements, domain constraints.... They are concerned with imposing a structure on the vague notation of a system". It is this characteristic that leads to a frequently reported side-effect of the application of RSLs: defects in the initial requirements are detected during the development of requirements models [e.g., Win90, SS97, EC97].

We subsume under the term *requirements specification languages* requirements modeling languages and formal methods for describing requirements. A *requirements modeling language* offers a graphical language with a formal syntax, that is, a set of diagram elements, and a semi-formal semantics, which is typically stated in natural language. Examples of requirements modeling languages include the Unified Modeling Language (UML) [UML99]. A *formal method* offers a language with a formal syntax and formal semantics. In most cases, this language is mathematical, but also graphical and tabular languages have been proposed. A formal method allows describing requirements rigorously and allows analyzing them extensively. Examples of formal methods include

SCR [HJL96], SDL [ITU93], VDM [Jon90], and Z [Spi92]. A *requirements model* is a set of requirements that is represented using a RSL. It is a formalized statement of requirements.

This paper reports on an empirical study aimed at answering two research questions about the defects spotted in informal requirements during formalization. A *defect* is a product anomaly in a requirements, design, or code document that leads to a misbehavior of a software system. We focus on conflicts, incompletenesses, and ambiguities in requirements documents. We address the following questions: *Are there differences in the numbers of conflicts, incompletenesses, and ambiguities*

- *that are found during creation of a requirements model*?
- *that are **not** found and, thus, are contained in the final requirements model*?

We expected differences, because a RSL forces the requirements engineer to be precise, that is, to resolve ambiguities before creating a requirements model. Thus, ambiguities become unambiguously right or wrong statements in a requirements model. Only a few types of ambiguities become conflicts, which are detected by the consistency checker of a CASE tool. On the other hand, a requirements model can be inconsistent or incomplete. Incompletenesses and, in particular, conflicts can violate syntactic or semantic rules of a RSL and then are detected by a CASE tool.

Seventeen graduate students from the University of Kaiserslautern and the Technical University of Munich participated in the study. The task was to develop a requirements model of a consumer electronics product, namely the Tamagotchi toy [Ban97]. The most popular RSLs, e.g., UML, SDL, as well as research prototypes, e.g., FOCUS [HMR+98], were used together with CASE tools.

This paper is structured as follows. First, the applied RSLs are briefly described. Second, the previous empirical research is reviewed. Then, the evaluation framework is discussed and the design of the study. Finally, the results are presented, threats to validity are discussed, and conclusions are drawn.

2. REQUIREMENTS SPECIFICATION LANGUAGES

The objects of the study were seven RSLs, namely Focus, SCR, SDL, OCTOPUS, ROOM, Statemate, and UML. The selection of these languages was driven by the availability of CASE tools, availability of experts for supervising the subjects, and practical relevance of languages. Furthermore, the languages should represent a good balance between emerging object-oriented RSLs, and traditional structural RSLs. The RSLs and the used CASE tools are described briefly in the following.

Focus is a formal method for modeling distributed systems. Its semantics is based on stream processing functions [Kah74]. AutoFocus is a prototype CASE tool developed by the Technical University of Munich, which implements the semantics of Focus [HMR+98].

The SCR (Software Cost Reduction) requirements specification language was developed by the Naval Research Laboratory (NRL) of the US Navy in 1978 [Hen80]. Recently, the NRL presented a formal semantics of SCR [HJL96] and developed a CASE tool called SCR* [HBGL95].

SDL (Specification and Description Language) is standardized by the International Telecommunication Union. It was developed in the late 1960s. Object-oriented concepts have been introduced into the version SDL 92 [ITU93]. SDL is considered today a formal description technique. We used the CASE tool ObjectGEODE from Verilog.

OCTOPUS [AKZ96] was developed in 1996 by Nokia. It is an object-oriented method based on OMT [RBP+91] and Fusion [Col94] for all phases of the development of embedded systems. There is no CASE tool for OCTOPUS, however, Awad et al. recommend among others Software through Pictures/OMT (StP) from Aonix, which was applied in the case study.

ROOM (Real-time Object-Oriented Modeling) is an object-oriented method for developing distributed systems created by Bell Northern Research, Ltd. in 1994 [SGW94]. We used the CASE tool ObjecTime Developer from ObjecTime Systems.

Statecharts are a state machine based description technique developed by David Harel [Har87]. The CASE tool Statemate MAGNUM from I-Logix extends the behavioral statecharts view with a structural and a functional view [HLN+90].

The Unified Modeling Language (UML) was presented by Booch, Rumbaugh, and Jacobson in 1997, the applied version was 1.3 [UML99]. The CASE tool Rhapsody by I-Logix was used, which allows generating executable code from a subset of UML diagrams.

All employed CASE tools, except StP/OMT, offer simulation of requirements models or full code generation. That is, there is some sort of formal semantics behind every RSL, except OCTOPUS.

3. PREVIOUS EMPIRICAL RESEARCH

Studies with RSLs have a long tradition at the International Workshop on Software Specification and Design (IWSSD) since the mid-80s and at the International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD) since the mid-90s.

There is an increasing number of empirical studies concerned with RSLs, e.g., Lewerenz and Lindner [LL94]; Ardis et.al. [ACJ+96]; Abrial, Boerger, and Langmaack [ABL96]; Bahill, Alford, Bharathan et.al [BAB+98], and the whole EMMSAD proceedings are devoted on this topic. However, there is a lack of commonly accepted criteria for method comparison [WGW97].

Finkelstein et.al suggested that empirical studies on RSLs should focus on processes rather than products, i.e., they should focus on causes rather than symptoms of strength and weaknesses of RSLs [FFFL97]. One process aspect is the previously mentioned side-effect of RSLs to help spot defects during formalization of informal requirements.

This side-effect was studied to our knowledge only by Wing *post-mortem* in specifications written for the fourth IWSSD workshop [Win88]. Wing made two conclusions. First, RSLs do not radically differ from one another in this respect. Second, the RSLs can be used to identify many, but not all, deficiencies in a set of informally stated requirements. Wing does not provide quantitative data.

4. EVALUATION FRAMEWORK

We extended the evaluation framework of Wing. We propose to track (1) the number of defects revealed in informal requirements *during* creation of a requirements model and (2) the number of defects that slip into the requirements model. Figure 1 shows all paths that a defect in the informal requirements can take. First, a specifier may observe a defect during formalization and report it to the customer. Then, he or she receives a solution to the defect. Thus, such a defect is removed in the requirements model. We assume that a specifier does not introduce a defect into the requirements model after knowing the solution, but it is possible. Defects that are removed in the requirements model because they were reported during formalization are called *customer-resolved defects*. Second, a specifier does not report a defect, because, e.g., he or she does not recognize it. Nevertheless, such a defect can be removed by chance. Defects that are removed in the requirements model but which were not reported during formalization are called *self-resolved defects*. Third, a defect that was not reported residues in the requirement document. Residual defects are further distinguished into forwarded and transformed defects. Some defects in the informal requirements are neither removed nor residual in the requirements model, because they are outside the scope of the requirements model. Therefore, the set of reported defects is not necessarily equal to the set of customer-resolved defects.

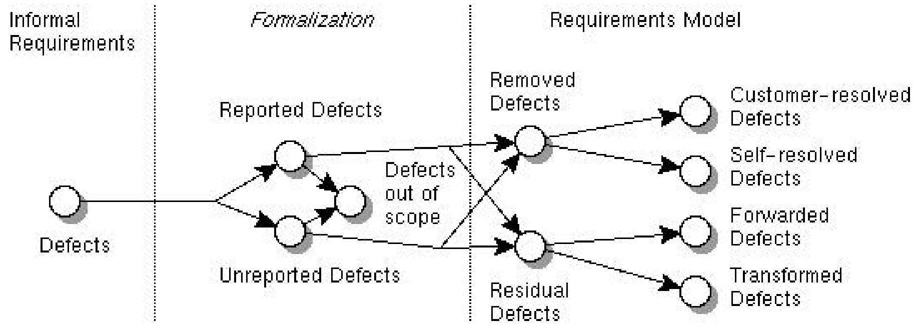

Figure 1. Classification of Defects

In the following, we explain the defect classes that are of particular interest in our study in more detail:

- *Reported Defect*. Precisely speaking, a defect was recognized by a specifier and was reported to the experimenters while *reading* an informal requirement or while *formalizing* an informal requirement. These two activities usually are intertwined and we made no attempt to distinguish them.
- *Removed Defect*. The requirements model is incorrect with respect to the informal requirements, but correct with respect to the customer's expectations.
- *Self-resolved Defect*. The defect has been removed, but it has not been reported, e.g., it has been removed by the requirements engineer using his or her background knowledge.
- *Forwarded Defect*. The same defect of the informal requirements is included in requirements model. The requirements model is correct with respect to the informal requirements, but incorrect with respect to the customer's expectations. For instance, an incomplete informal requirement has not been recognized and has become an incomplete statement in the requirements model.
- *Transformed Defect*. A defect in the informal requirements has been *transformed* into another type of defect in the requirements model. The requirements model is incorrect with respect to the informal requirements as well as with respect to the customer's expectations. For instance, an ambiguous requirement has been misinterpreted and has become an incorrect statement in the requirements model.
- *Defect out of scope*. The requirements model is incomplete with respect to the informal requirements as well as with respect to the customer's expectations. For example, if a defect is concerned with a timing requirement, and timing requirements cannot be specified with the applied RSL, e.g., because of lack of notational power of the RSL, then the defect is out of scope.

Defects newly introduced into the requirements models were not investigated in this study, because the effort would have been too high to inspect the requirements models in detail.

We limited the types of defects considered in the study to defects that can be identified without knowledge of the application domain, because we did not expect the subjects to have deep knowledge about the application domain. In particular, we were interested in incompletenesses (only those detectable without domain knowledge), conflicts, and ambiguities. For the same reason, we did not consider incorrect, unrealistic, or extraneous requirements.

A requirements document is *incomplete* if information is missing such as a function or a definition of a response to particular input data. A requirement is *ambiguous* if it has several interpretations. Ambiguities include not only linguistic ambiguities such as an uncertain pronoun reference, but also ambiguities about the actual system and its behavior [SMT92]. Two requirements are *inconsistent* if they state facts that cannot both be true, or if they express actions that cannot be carried out at the same time. This type of defect is also called *conflict*. Incompletenesses and ambiguities can be distinguished by the type of required correction activity. The former require adding information, while the latter just require rephrasing the present information so that a requirement unambiguously conveys its meaning.

5. EXPERIMENTAL STUDY

This section describes hypotheses, design, instruments, preparation, execution, and data validation.

Hypotheses. The hypotheses followed the research questions, which are stated in the beginning of this paper. We assume that there are no significant differences between the investigated RSLs in spotting defects, because they all address behavioral requirements and provide some state-machine-based language to describe them. Rather, we expect differences between the defect types. A RSL forces the requirements engineer to be precise. However, we expect that a specifier does not necessarily recognize an ambiguity as such and misinterprets it, while the structure imposed by the RSL on the requirements helps detect inconsistencies and incompletenesses (recall that we limited the considered kinds of incompletenesses to those that are detectable without domain knowledge). Therefore, we hypothesize that *fewer* ambiguities and *more* incompletenesses and conflicts are *reported* than one would expect based on the overall numbers of those defects in the requirements document. Our alternative hypothesis is

H_1: There is a difference between the observed and expected numbers of incompletenesses, conflicts, and ambiguities that are reported during formalization using a RSL.

Following the above argumentation, we hypothesize that *fewer* ambiguities and *more* incompletenesses and conflicts are *removed* than one would expect based on the overall numbers of those defects in the requirements document, because ambiguities are less frequently recognized. Rather, one portion of the ambiguities that was not recognized is misinterpreted. Thus, we hypothesize that *more* ambiguities and *fewer* incompletenesses and conflicts are *transformed* than one would expect based on the overall numbers of those defects in the requirements document. The other portion of ambiguities that was not recognized is correctly interpreted with the help of other requirements and real-world knowledge. Thus, we hypothesize that *more* ambiguities and *fewer* incompletenesses and conflicts are *self-resolved* than one would expect based on the overall numbers of those defects in the requirements document. Finally, we hypothesize that *fewer* ambiguities and *more* incompletenesses and conflicts are *forwarded* than one would expect based on the overall numbers of those defects in the requirements document. A requirements model can be inconsistent and incomplete to some degree, even if it is checked by a CASE tool. However, ambiguities must be removed or transformed, thus, we expected virtually no forwarded ambiguities in the requirements model. Our alternative hypotheses are

H_2: There is a difference between the observed and expected numbers of incompletenesses, conflicts, and ambiguities that are {removed, self-resolved, forwarded, and transformed} in a requirements model.

In order to save space, Hypothesis H_2 is parameterized, which is indicated by the parentheses. That is, it subsumes four similar hypotheses. The first hypothesis concerns removed defects, the second self-resolved defects, the third forwarded defects, and the fourth transformed defects.

Subjects. The empirical study was performed at the University of Kaiserslautern (UKL) and the Technical University of Munich (TUM). 10 students from UKL and 9 students from TUM participated in the empirical study. All students were enrolled in a joint seminar. The students were from the third year and above and had knowledge of the principles underlying the RSLs such as finite state machines and object-orientation, but no experience with the particular languages or CASE tools. The students worked together in teams of 2 or 3 students.

Design. The RSLs were examined in a *replicated project* scope according to Basili's classification of experimental scopes [BSH86]. This means that each team performed the same task, the development of a requirements model based on a set of informal requirements, but using a different RSL. Six teams were formed such that there is a one-to-one relation between team and RSL. This design was a compromise, because the effort per person and week is required to be low for a seminar (4 hours). Ideally, each team should apply each RSL. However, the required resources were not available.

Teams were formed according the students' preferences. We decided to assign teams non-randomly to RSLs, because of the distributed nature of the seminar. The UKL students had a choice between only OCTOPUS, SCR, Statecharts, and UML, and the TUM students had a choice between only ROOM, SDL, and Focus. This restriction was due to the location of supervisors experienced with these RSLs.

Instrumentation. The teams received an informal requirements document about 9 pages, which was written by the authors of this paper. This document described a consumer electronics product, the Tamagotchi toy, which is an event-driven system (note that all selected RSLs are well-suited for specifying event-driven systems). The requirements document had two parts, a problem description of 4 pages that defines the background of a fictional software development project, and the customer requirements of 5 pages that describe the desired behavior of the Tamagotchi toy. The customer requirements consist of 42 textual requirements; each requirement has on average 2 or 3 sentences. Some requirements were derived from a book describing the Tamagotchi [Ban97], others were reverse engineered from the original toy, and some were invented.

The requirements document contained 57 defects; 38 ambiguities, 13 incompletenesses, and 6 conflicts. Our experience with previous empirical studies in RE has shown that is not practical to seed defects in a requirements document after it was written. Defects in requirements documents are so diverse that "invented" defects could only cover a small portion of the defects actually occurring in practice. Furthermore, a requirements document written in natural language always contains some redundancy. This does not mean necessarily that requirements are mentioned twice, but also that contextual information is given, which backs up the requirements. To introduce a defect "consistently", these redundant parts need to be changed as well. Doing this change consistently and uniformly has proven not feasible.

We seeded defects *during* the creation of the requirements. Thereby, we relied on the observation that the first versions of a requirements document contain lots of defects even if they are written carefully. This observation was made by Martin and Tsai who found in a well-written 16 page requirements document 92 defects [MT90]. Our experience from a previous empirical study confirms this observation [KKR+98]. We have identified the defects through an intensive review and through the questions raised by the students.

Data Collection. Data collection was performed in several ways. The teams were required to write a brief report about each issue they encountered in the informal requirements. A solution to the issue was sent in reply by the author of the requirements document. This solution was *not sent* to the other teams.

At the end of the seminar, we interviewed each team about its requirements model using the list of 57 known defects in the informal requirements. For each defect, it was checked whether the defect has been removed, forwarded, transformed, or is out of the scope of requirements model.

Preparation. The preparation phase took 4 weeks. During this phase, the students read material about their RSL and produced a tiny requirements model of the ESFAS (Engineered Safety Feature Actuation System) [CP93]. Then, each team wrote a 1-to-2-page essay about the RSL. At the end of this phase, the students had an opportunity to discuss all the problems they encountered with the RSL or CASE tool with their supervisor. The outcome of this phase was the requirements model and the essay. Based on these two deliverables, the supervisor got an impression of the students' current understanding of the treatment. In the case a team's understanding was poor, the supervisor could discuss the problems with the team. However, this case actually did not occur.

Execution. The execution phase took 8 weeks and comprises the development of a requirements model of the Tamagotchi toy. All issues and defects that were detected during the formalization of informal requirements were reported to the supervisor. The questions were forwarded to and answered by the author of the requirements document, i.e., the customer, in such a way that if two teams encountered the same issue or defect, they received the same answer. A team spent 100 hours minimum on the development of the requirements model.

Data Validation. Several activities were performed to validate the data. First, the validity of the reported issues was checked by the author of the requirements document. There were cases in which the description of an issue that was submitted by a team was not clear. In such a case, the issue was clarified in a discussion between the author and the team.

Second, the validity of the data collected regarding removed, forwarded, and transformed defects were checked by simulation and inspection of the requirements models. During the interviews in which the data were collected, each team was asked to show certain behaviors of the requirements model by using the CASE tool's simulation feature. After the course was over, further validation was performed by inspection of the requirements models as part of a diploma thesis.

6. RESULTS

The study could not be performed as planned. Due to different university holidays at TUM, the schedule for the execution phase was tighter at TUM than at UKL. Thus, the TUM teams did not have enough time for writing defect reports and waiting for answers. Effectively, this means that the processes at UKL and TUM differed in *customer participation*. At UKL, the customer was involved from the beginning of the formalization process. At TUM, the customer was involved only at the end of the formalization process, when the final requirements models were evaluated by an interview with the team. This deviation from the plan described in the previous chapter allows investigating a third hypothesis regarding the impact of customer participation on the detection of defects.

On the other hand, the deviation from the plan makes it necessary that we block the whole analysis of this study with respect to the site (UKL, TUM). It is necessary to separate the analysis of the UKL teams from the analysis of the TUM teams, because the differing customer participation has an effect on the results. The fact that customer and user participation can influence the RE process was shown by El Emam et al. in an empirical study [EQM96].

We tested the hypothesis H_1 by a Chi-Square test, which allows testing whether a set of observed frequencies departs significantly from a set of theoretical frequencies. We demanded a significance level α=0.05, which is most common in software engineering experiments. Concretely, we tested whether the reported numbers of ambiguities, incompletenesses, and conflicts depart significantly from the expected numbers of those defects.

Based on the numbers of known defects in the requirements document (38 ambiguities, 13 incompletenesses, and 6 conflicts), the theoretical probabilities of detecting an ambiguity, incompleteness, and conflict were 0.67 (38 ambiguities divided by 57 defects in total), 0.23, and 0.1, respectively. The expected number of defects of a particular type $f_{e(j)}$ was calculated $f_{e(j)} = n * p_j$, where n is the total number of reported defects and p_j, is the probability of detecting a defect of type j. For example, if a team reports 20 defects, we would expect 13.4 ambiguities (20 reported defects * 0.67 probability of detecting an ambiguity).

We tested the hypotheses H_2 in the same way.

Table 1 summarizes the observed ("O") and the expected ("E") numbers of reported, removed, self-resolved, forwarded, and transformed defects for each team and the results the Chi-square tests. A precondition of this test is that the expected frequencies are not below 5. Because most expected frequencies are below 5, we test the hypotheses H_1 and H_2 on the *aggregated data* of the UKL and of the TUM teams, respectively. We used the procedures described in [FT89] to aggregate data.

RSL	Site	Observed / Exp.	Reported			Removed			Self-resolved			Forwarded			Transformed		
			Incomplete.	Conflicts	Ambiguities	Incomplete.	Conflicts	Ambiguities	Incomplete.	Conflicts	Ambiguities	Incomplete.	Conflicts	Ambiguities	Incomplete.	Conflicts	Ambiguities
SCR	UKL	O	8	2	8	10	4	27	1	2	20	2	2	3	1	0	8
		E	4	2	12	9	4	26	5	3	15	1	1	5	2	1	6
Statecharts	UKL	O	6	2	5	9	3	27	3	1	23	4	3	4	0	0	7
		E	3	1	9	9	4	26	6	3	18	3	1	7	2	1	4
OCTOPUS	UKL	O	4	2	3	-	-	-	-	-	-	-	-	-	-	-	-
		E	2	1	6	-	-	-	-	-	-	-	-	-	-	-	-
UML	UKL	O	2	1	5	9	3	28	7	2	23	2	2	3	2	1	7
		E	2	1	5	9	4	27	7	3	22	1	1	5	2	1	7
Chi-square (p-level)	UKL		**.009985**			.997092			**.037973** (no UML)			**.004681**			**.021563** (no UML)		
ROOM	TUM	O	-	-	-	9	2	19	-	-	-	2	3	3	2	1	16
		E	-	-	-	7	3	20	-	-	-	2	1	5	4	2	13
SDL	TUM	O	-	-	-	8	3	23	-	-	-	3	3	3	2	0	12
		E	-	-	-	8	4	22	-	-	-	2	1	6	3	2	9
Focus	TUM	O	-	-	-	5	2	21	-	-	-	7	4	3	1	0	14
		E	-	-	-	6	3	19	-	-	-	3	2	9	3	2	10
Chi-square (p-level)	TUM		-			.959297			-			**.000009**			**.023047**		

Table 1. Collected Data and Results of Chi-Square Tests

As Table 1 shows, there were differences in the absolute numbers of incompletenesses, conflicts, and ambiguities that were reported, removed, and so on. For example, the SCR team reported 8 incompletenesses, 2 conflicts, and 8 ambiguities, while the OCTOPUS team reported only 4 incompletenesses, 2 conflicts, and 3 ambiguities. Because of the similarity of the applied RSLs and based on our personal judgement of the capabilities of the teams, we believe that these differences cannot be attributed to the RSLs. On the other hand, the single team results show a consistent profile of incompletenesses, conflicts, and ambiguities that were reported, removed, and so on, except for the UML team. For example, each of the SCR, Statecharts and OCTOPUS teams reported more incompletenesses and fewer ambiguities than one would expect based on the total numbers of these defects in the requirements document. However, the UML team behaved differently from the other UKL teams. The UML team reported a number of defects quite close to the expected number. This team said in the final interview that they did not report every issue they became aware of, but only the ones that they believed they could not solve themselves. Because of this lack of conformance with the experimental process, we treated the UML team as an outlier in two tests. The OCTOPUS requirements model was not very detailed. Therefore, it is omitted from further analysis.

The results of the statistical tests are discussed in the remainder of this section. We do not discuss conflicts in detail, because their numbers are too low.

H_1—Reported Defects

We can reject the null hypotheses H_{01} for the numbers of reported incompletenesses, ambiguities, and conflicts, as Table 1 shows. The observed numbers differ significantly from the expected ones. The application of a RSL leads to *higher numbers* of detected incompletenesses and conflicts and *lower numbers* of detected ambiguities, as one would expect based on the defect numbers in the document. A UKL team reported on average 14% of the known ambiguities, but 39% of the known incompletenesses. This result is noticeable. It shows that ambiguities are not detected just because the informal requirements are formalized. If the requirements engineer is not aware of an ambiguity while developing a requirements model, then a RSL does not help to detect the ambiguity. On the other hand, a RSL seems to help detect incompletenesses and conflicts, because they were reported more frequently than expected.

H_2—Removed Defects

We cannot reject the null hypotheses H_{02} for the numbers of removed incompletenesses, ambiguities, and conflicts both at UKL and TUM, as Table 1 shows. When a RSL is applied, there is no difference between the numbers of removed incompletenesses, conflicts, and ambiguities and what one would expect based on the defect numbers in the document. 72% ambiguities and incompletenesses were removed on average by a UKL team. 56% ambiguities and incompletenesses were removed on average by a TUM team.

H_2—Self-resolved Defects

We can reject the null hypothesis H_{02} for the numbers of self-resolved incompletenesses, ambiguities, and conflicts at UKL, as Table 1 shows. There is a significant difference between the numbers of defects that are self-resolved and their expected numbers. On average, a UKL team (except for the UML team) resolved 57% of the known ambiguities, but it resolved only 16% of the known incompletenesses without asking the customer. During the final interviews it became apparent that the teams often did not recognized ambiguities as such. Therefore, we conclude that ambiguities are more often *unconsciously* removed than are other types of defects. Unconscious disambiguation is a serious problem, because implicit assumptions are more likely than in our study to be wrong when the system is more complex.

H_2—Forwarded Defects

We can reject the null hypotheses H_{02} for the numbers of forwarded incompletenesses, ambiguities, and conflicts both at UKL and TUM, as Table 1 shows. The observed numbers differ significantly from the expected ones. In accordance with our expectation, the application of an RSL leads to *higher numbers* of forwarded incompletenesses and conflicts and to a *lower number* of forwarded ambiguities, as one would expect based on the defect numbers in the document. On average, a UKL team forwarded only 9% of the known ambiguities, but it forwarded 21% of the known incompletenesses. In the case of the TUM teams, this difference is even bigger. On average, a TUM team forwarded only 8% of the known ambiguities, but it forwarded 31% of the known incompletenesses. This result confirms that the applied RSLs significantly reduce the level of ambiguity, however, they do not eliminate ambiguity.

H_2—Transformed Defects

We can reject the null hypotheses H_{02} for the numbers of transformed incompletenesses, ambiguities, and conflicts at UKL and at TUM, as Table 1 shows. The observed numbers differ significantly from the expected ones. The application of a RSL leads to more transformed ambiguities and fewer transformed incompletenesses than one would expect based on the defect numbers in the document. On average, a UKL team (except for the UML team) transformed 20% of the known ambiguities, but it transformed only 4% of the known incompletenesses. Again, the difference is bigger for the TUM teams. On average, a TUM team transformed 37% of the known ambiguities, but it transformed only 13% of the known incompletenesses.

If not detected and not removed, incompletenesses and conflicts tend to become forwarded, while ambiguities tend to become transformed (i.e., misinterpreted). This behavior of ambiguities is a problem, since such a misinterpretation can slip through undetected, because of the customers' reluctance to read requirements written in artificial language. Simulation, the other way of validating formal requirements, can show only the presence of misinterpretations but not their absence. Disastrous software failures may be the consequence.

H_3—Customer Participation

There was a difference in the customer participation between the sites UKL and TUM. At UKL, the customer was involved from the beginning of the formalization process. At TUM, the customer was involved only at the end of the formalization process, when the final requirements model was evaluated by an interview with the team. This deviation of the actual course of the empirical study from the plan allows us to investigate a third hypothesis regarding ambiguities. We do not analyze incompletenesses in this respect, because the investigated requirements document contained mainly incompletenesses that could be resolved by reasoning. UKL and TUM students were equal good in reasoning. Thus, there were no significant differences in the numbers of removed incompletenesses between UKL and TUM teams.

We expect a significant difference between UKL and TUM in the numbers of removed and transformed ambiguities. Humans are naturally skilled in resolving ambiguity. Thus, the ambiguities that were reported at UKL are those that need clarification. If there is no customer participation, as in the case of TUM, the likeliness of misinterpretations raises.

H_3: There is a difference between the UKL and TUM teams in the numbers of removed and transformed ambiguities.

Table 2 shows the average numbers of reported, forwarded, and transformed ambiguities for UKL and TUM teams and the results of a Mann-Whitney *U* test (nonparametric variant of *t* test [FT89]).

	Removed Ambiguities	Forwarded Ambiguities	Transformed Ambiguities
UKL	27.3	3.3	7.3
TUM	21.0	3.0	14.0
p-value	**0.49543**	.512695	**0.49543**

Table 2. Effect of Customer Participation

We can reject the null hypothesis H_{03}. There is a significant difference between the numbers of removed ambiguities at UKL and TUM. The UKL teams removed 72% of the known ambiguities, while the TUM teams removed only 55%. Consequently, there is also a significant difference between the number of transformed ambiguities at UKL and TUM. The TUM teams resolved twice as many ambiguities, 37%, the wrong way as did the UKL teams. The fact that there are no significant differences between UKL and TUM in the numbers of forwarded ambiguities shows the homogeneity of the two groups. Recall that a forwarded defect is a defect that was not observed. Therefore, the customer participation should not have an effect on the numbers of forwarded ambiguities.

We have analyzed the single ambiguities that were removed by the two groups. Interestingly, each ambiguity that was reported and removed by an UKL team was also recognized and removed by a TUM team. That is, there were no ambiguities that could not be resolved without help. The difference lies in the frequency; more UKL teams were able to remove an ambiguity, because they had access to the customer, than did the TUM teams. This observation confirms a key characteristic of ambiguity. Ambiguity, if it is noticed, needs immediate clarification. Any ambiguity that is removed by one team without a report, can be misinterpreted unconsciously by another team, and can raise a question for a third team. If this question is not answered, the number of transformed ambiguities grows.

7. THREATS TO VALIDITY

The following possible threats to the validity of this study have been identified:

- The one-to-one mapping between team and RSL makes it difficult to prove that the observed differences between the UKL and TUM teams are really caused by the different customer participation (see hypothesis H_3). It could be the case that the differences are caused by inherent differences among the RSLs, i.e., between SCR, Statecharts, and UML on one hand and ROOM, SDL, and Focus on the other hand. However, as mentioned previously, all RSLs are based on finite state machines. Therefore, the second explanation is unlikely. Note that the results regarding H_1 and H_2 are not affected by this problem, because we did not test for differences *among the RSLs*.
- It was possible for the teams to exchange information. However, we told the teams not to do so and there was no exam at the end, thus, cheating made little sense. Since each team applied a different RSL, it was not possible for a team to copy the requirements model of another team in order to save effort.
- The requirements document might not be representative in terms of size, complexity, and numbers of defects. The Tamagotchi system already exists, therefore, the requirements were well-understood and the requirements document might expose a different defect profile compared to one describing a completely new system. However, we strongly believe that our results can be generalized to other requirements documents, as far as ambiguities and incompletenesses are concerned. The number of conflicts in the Tamagotchi requirements document is too low to draw significant conclusions on them.
- The results regarding incompletenesses are valid only for incompletenesses that can be detected without domain knowledge.
- The RE process that we followed in this case study assumes a relatively complete and detailed requirements document, before a RSL is used. However, if requirements models are created, they are usually created from a sketchy requirements document, in order to avoid describing things twice. The RE process that we followed is typical for safety-related domains such as the aerospace domain.
- The subjects who participated in the case study are unlikely to be representative of professional requirements engineers. Therefore, we cannot generalize the results to that population. However, this replicated project type of study could not be done in industry because of the high cost. We believe that student experiments are useful as a pilot for later industrial experiments. For example, we can test hypotheses in a student setting in order to decide whether it is worth investigating them further in industrial settings.

8. CONCLUSIONS

This paper presented the results of an empirical study with RSLs. The participants applied RSLs to develop an executable requirements model from a given set of informal requirements whose defects were known to the experimenters. The students were told to report every defect they encountered in the informal requirements during formalization. We analyzed the reported issues, and we analyzed the requirements models based on what happened to those defects in the informal requirements that slipped through undetected. The results of our study are: (1) ambiguities are less frequently reported than other types of defects; (2) ambiguities are removed more often unconsciously, i.e., are correctly resolved without being reported, than other types of defects; (3) ambiguities are misinterpreted more often than other types of defects; and (4) ambiguities, if noticed, require immediate clarification.

The lesson learned from this empirical study is that conflicts and certain types of incompletenesses can be easily detected through formalization and automated checking of the resulting requirements models. However, requirements engineers should not rely on the formalization of informal requirements helping to spot ambiguities in informal requirements; only some ambiguities are detected. Moreover, ambiguities tend to become misinterpreted if they are not recognized and if they are not unconsciously self-resolved.

Based on these results, we make two recommendations for the use of RSLs in RE processes:

1. **Inspection of informal requirements before their formalization.** Since RSLs enforce precision, an ambiguity can become an unambiguously wrong formal requirement, which can slip through undetected, because of the customers' reluctance to read requirements written in artificial language and theoretical limitations of simulation. We recommend the inspection of informal requirements for ambiguities to avoid these problems. An inspection technique for spotting ambiguities is introduced in [Kam01].
2. **Participation of customers and users during formalization.** The development of requirements models from informal requirements is a task of requirements engineers, not customers or users. Nevertheless, we recommend participation of customers and users *during* the development of these models, not afterwards, in order clarify observed ambiguities as soon as possible. Spotted ambiguities that cannot be clarified immediately with the customer tend to become misinterpreted. If those misinterpretations are clarified later, costly rework of models may be required.

The phenomenon of ambiguity was investigated empirically only in psycho-linguistics. We would like to encourage researchers to perform empirical research in requirements engineering to further investigate the impacts of ambiguity during formalization of requirements. Requirements specification languages are unambiguous, but customer requirements are usually stated in natural language first.

ACKNOWLEDGEMENTS

We would like to thank Martin Glinz for his comments on the design of this study and we thank Dan Berry and Barbara Paech for their comments on previous versions of this paper. Moreover, we thank the students from the University of Kaiserslautern and the Technical University of Munich for their willingness to participate in this experiment. Finally, we gratefully acknowledge the constructive and detailed comments of the anonymous referees on the previous version of this paper.

REFERENCES

[ABL96] J.-R. Abrial, E. Börger, and H. Langmaack (Eds.). *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*. Lecture Notes in Computer Science 1165. Springer, 1996.

[ACJ+96] M.A. Ardis, J.A. Chaves, L.J. Jagadeesan, P. Mataga, C. Puchol, M.G. Staskauskas, and J.von Olnhausen. A framework for evaluating specification methods for reactive systems: Experience report. *IEEE Transactions on Software Engineering*, 22(6):378-389, June 1996.

[AKZ96] Maher Awad, Juha Kuusela, and Jurgen Ziegler. Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT and Fusion. Prentice Hall, 1996.

[BAB+98] A.T. Bahill, M. Alford, K. Bharathan et al. The Design Methods Comparison Project. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 28(1): 80-103, 1998.

[Ban97] Bandai. *Das Original Tamagotchi Buch*. Tamagotchi & Bandai, 1997.

[BSH86] V.R. Basili, R.W. Selby, and D.H. Hutchens. Experimentation in software engineering. *IEEE Transactions on Software Engineering*, SE-12(7):733–743, July 1986.

[CP93] P.-J. Courtois and D.L. Parnas. Documentation for safety critical software. In *Proceedings of the 15th International Conference on Software Engineering*, pages 315–323, Baltimore, Maryland, USA, May 1993. IEEE Computer Society Press.

[EC97] S. Easterbrook and J. Callahan. Formal methods for V&V of partial specifications: An experience report. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, pages 160–168, Annapolis, Maryland, USA, January 1997.

[EQM96] K. El Elmam, S. Quintin, and N.H. Madhavji. User participation in the requirements engineering process: An empirical study. *Requirements Engineering Journal*, 1(1):4–26, 1996.

[FFFL97] M.S. Feather, S. Fickas, A. Finkelstein, and A. van Lamsweerde. Requirements and specification exemplars. *Automated Software Engineering*, 4(4):419–438, October 1997.

[FT89] G.A. Ferguson and Y. Takane. *Statistical Analysis in Psychology and Education*. McGraw-Hill, 1989.

[Har87] D. Harel. Startecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[HBGL95] C. Heitmeyer, A. Bull, C. Gasarch, and B. Labaw. SCR*: A toolset for specifying and analyzing requirements. In *Proceedings of the 10th Annual Conf. on Computer Assurance*, pages 109–122, Gaithersburg, MD, USA, June 1995.

[Hen80] K.L. Heninger. Specifying software requirements for complex systems: New techniques and their application. *IEEE Transactions on Software Engineering*, SE-6(1):2–13, January 1980.

[HJL96] C.L. Heitmeyer, R.D. Jeffords, and B.G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, July 1996.

[HLN+90] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.

[HMR+98] F. Huber, S. Molterer, A. Rausch, B. Schätz, M. Sihling, and O. Slotosch. Tool supported specification and simulation of distributed systems. In B. Krämer, N. Uchihira, P. Croll, and S. Russo, editors, *Proceedings International Symposium on Software Engineering for Parallel and Distributed Systems*, pages 155–164, 1998.

[ITU93] *Recommendation Z.100, Specification and Description Language (SDL)*. ITU, 1993.

[Jon90] C. B. Jones. *Systematic Software Development Using VDM*. Prentice Hall, New Jersey, USA, 1990.

[Kah74] G. Kahn. The Semantics of a Simple Language for Parallel Programming. *Information Processing*, 74:471–475, 1974.

[Kam01] E. Kamsties, *Surfacing Ambiguity in Natural Language Requirements*, PhD Thesis, Univ. of Kaiserslautern, 2001.

[KKR+98] A. von Knethen, E. Kamsties, R. Reussner, C. Bunse, and B. Shen. A comparative case study with industrial requirements engineering methods. In *Proceedings of the 11th International Conference on Software Engineering an its Applications*, Paris, France, December 8–10 1998. Vol.3: Preprints.

[LL94] C. Lewerenz and T. Lindner (Eds.). Case study "production cell". FZI-Publication 1/94, Forschungszentrum Informatik, Universitaet Karlsruhe, Germany, 1994.

[MT90] J. Martin and W. T. Tsai. N-fold inspection: A requirements analysis technique. *Communications of the ACM*, 33(2):225–232, February 1990.

[SGW94] B. Selic, G. Gullekson, and P.T. Ward. *Real–Time Object–Oriented Modeling*. Wiley, 1994.

[Spi92] J. M. Spivey. *The Z Notation – A Reference Manual*. Prentice-Hall, 1992.

[SS97] I. Sommerville and P. Sawyer. *Requirements Engineering – A good practice guide*. Wiley, 1997.

[UML99] OMG Unified Modeling Language. Technical report, Rational Software Corporation, June 1999. Version 1.3.

[WGW97] Y. Wand, A. Gemino, and C. Woo. An Empirical Comparison of Object Oriented and Data Flow Modeling. *Proceeding of the CAiSE/IFIP 8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'97)*, Barcelona, Spain, June, 1997.

[Win88] J.M. Wing. A study of 12 specifications of the library problem. *IEEE Software*, pages 66–76, July 1988.

[Win90] J.M. Wing. A specifier's introduction to formal methods. *IEEE Computer*, 23(9):8–24, September 1990.