Experimentally investigating effects of defects in UML models

Experimentally investigating Effects of Defects in UML Models

Christian F.J. Lange, Michel R.V. Chaudron
Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, P.O. Box 513
5600 MB Eindhoven, The Netherlands
{*C.F.J.Lange, M.R.V.Chaudron*}*@tue.nl*

March 14, 2005

Abstract

The Unified Modeling Language (UML) is the de facto standard for designing and architecting software systems. With a large variety of diagram types it offers a large degree of freedom. Mainly due to this there is a potential risk for consistency defects and lack of completeness in UML models. Previous research has shown that in industrial UML models a large number of defects occur. The purpose of this study is to investigate to what extent implementers detect defects and to what extent undetected defects cause different interpretations by different readers. Therefore we performed two controlled experiments with a large group of students and a smaller group of industrial practitioners respectively. The experiment's results show that defects often remain undetected and cause misinterpretations. As a result of this study we present a ranking of defect types according to detection rate and risk for misinterpretation. Additionally we observed effects of using domain knowledge to compensate defects and effects of the order in which the diagrams are presented. The results are generalizable to industrial UML users.

Contents

1 Introduction

This study is performed within the EmpAnADa project [Tec]. The EmpAnADa project develops techniques to assess and improve the quality of models for the design and architecture of software systems. Within the empirical component of the EmpAnADa project the techniques are applied and validated. The Unified Modeling Language (UML) is the de facto standard for designing and architecting software systems. Therefore the focus of the project is on the UML. An integral part of the project is the empirical investigation of problems arising during the use of the UML.

The UML lacks a formal semantics and allows many degrees of freedom by offering powerful extension mechanisms like stereotypes and profiles. As a result, the UML is applied in many different ways by its users. Additionally the UML offers various diagram types to describe the system from different views. UML 1.x [Obj03b] offers 9 diagram types and in the new UML 2.0 [Obj03a] the user can choose from 13 diagram types. Since diagrams from all types in a model essentially describe the same system there is overlap [BBD+00] between different diagrams (from the same type as well as from different types). Views (i.e. diagrams) are not created from a single source, but are independently constructed and edited. This causes the risk of completeness and consistency defects in overlapping diagrams. A consistency defect is a mismatch between overlapping diagrams. There are several approaches that classify these defects. Additionally there are several methods [KHR+03, MCB05] that aim at finding, removing and preventing defects in UML models. Despite these methods and modern UML case tools that assist in preventing defects in UML models the number of defects in practice is alarmingly large. Empirical studies on industrial UML models [LC04] have shown a large amount of defects.

UML models are used for finding solutions and analyzing their properties, to document them and as a means for communication between stakeholders. It has not yet been addressed whether the presence of defects in UML models affects these applications.

The goal of this study is to investigate the effects of defects in UML models in the context of Master's students and industrial professionals. The following research questions are addressed:

- Are defects in UML models detected by readers?
- (If defects are not detected) how do they effect the interpretation of the model?

Additionally the effect of availability of domain knowledge and the order in which the diagrams are presented to the reader on the interpretation of the model are addressed by this study. The goal of this study is summarized by the GQM template [BCR94] (Table 1).

To answer these questions we have designed and performed two controlled experiments. Section 2 explains the defect types that we have investigated in the experiment. In Section 3 the experimental design, its execution and the

Analyze	consistency and completeness defects in UML models
for the purpose of	identifying risks
with respect to	detection and misinterpretation
from the perspective of	the researcher
in the context of	graduate students at the TU Eindhoven and professionals

Table 1: Goal according to GQM template

two groups of subjects are described. The results and major findings of the experiment are presented in Section 4. Observations made are discussed in Section 5. Concluding remarks and future work are given in Section 6. In the appendix we present the developed agreement measure and the raw results.

2 Consistency Defects

A UML model consists of several diagrams and these diagrams have a certain overlap. When a mismatch between overlapping parts of diagrams occurs a consistency defect is present. A formal definition can be found in [Lan03]. In this section the defect types that are analyzed in the described experiment are described.

2.1 Prerequisites

We start with giving some definitions that are needed for the formal specification of the defects. The formal specifications are based on a relational meta model which contains the following sets:

$$
\begin{aligned}
U &= \textit{set of all use cases} \\
S &= \textit{set of all sequence diagrams} \\
C &= \textit{set of all classes} \\
O &= \textit{set of all objects} \\
M &= \textit{set of all methods} \\
E &= \textit{set of all messages} \\
N &= \textit{set of all names} \\
L &= U \cup S \cup C \cup O \cup M \cup E
\end{aligned}
$$

Based on these sets we consider relations of the following types:

$$
\begin{aligned}
US &\subseteq U \times S \\
&\quad \langle u, s \rangle \in US \textit{ means } s \textit{ illustrates } u \\
SE &\subseteq S \times C \times C \times E
\end{aligned}
$$

$$
\begin{array}{rcl}
 & & \langle s, c_1, c_2, e\rangle \in SM \textit{ means in context of } s, \\
 & & \quad c_1 \textit{ sends } e \textit{ to } c_2 \\
EM & \subseteq & E \times M \\
 & & \langle e, m\rangle \in EM \textit{ means } e \textit{ is a call of } m \\
CM & \subseteq & C \times M \\
 & & \langle c, m\rangle \in CM \textit{ means } c \textit{ provides } m \\
OC & \subseteq & O \times C \\
 & & \langle o, c\rangle \in OC \textit{ means } o \textit{ instantiates } c \\
OS & \subseteq & O \times S \\
 & & \langle o, s\rangle \in OS \textit{ means } o \textit{ is part of } s
\end{array}
$$

Functions. Here we list the functions that are used when defining the defect types. The function $name(x)$ returns the name of the model element x ($x \in L$). The function $class(o)$ returns the class that is instantiated by object o. The functions $callee(e)$ and $caller(e)$ return the object that receives message e and the object that sends message e, respectively.

For this experiment the correspondence between a message and a method is expressed by equality of names, i.e.

$$name(e) = name(m) \equiv \langle e, m\rangle \in EM \tag{1}$$

2.2 Consistency Defect Types

Now we are ready to give the set of defects that are analyzed in the experiment. The defect types analyzed in this study take into account class diagrams, sequence diagrams and use case diagrams. Each defect has a name, an abbreviation (which is used in the sequel), a brief description that informally describes the defect, and a formal specification. Note that the formal specification is a boolean expression that is true, when an instance of the defect type is present, it does not describe the set of defects.

Message without Name (EnN)
Objects are passing messages in sequence diagrams. The arrows representing the messages should be annotated with a name that describes the message. In case the message is not described by a name, this defect is present.

$$(\exists e \in E : \neg(\exists\ string\ s : s = name(e)))$$

Message without Method (EcM)
A message from one object to another means that the first object calls a method that is provided by the second object. The name annotating the message ideally corresponds to name of the called method. In case there is no correspondence between the message name and a provided method name, this defect is present.

$$
\begin{array}{rcl}
(\exists e \in E & : & \neg(\exists m \in M : \\
 & & name(m) = name(e) \\
 & & \wedge \langle class(callee(e)), m\rangle \in CM))
\end{array}
$$

Message in the wrong direction(ED)
This inconsistency occurs when there is a message from an object of class A to an object of class B but the method corresponding to the message is a member of class A instead of class B. This is a special instance of "Message name does not correspond to Method".

$$\begin{aligned}(\exists e \in E \quad : & \\ \neg \quad & (\exists m \in M : name(m) = name(e) \\ \wedge \quad & \langle class(callee(e)), m\rangle \in CM) \\ \wedge \quad & (\exists m \in M : name(m) = name(e) \\ \wedge \quad & \langle class(caller(e)), m\rangle \in CM))\end{aligned}$$

Class not instantiated in SD (CnSD)
Sequence diagrams describe the interactions between instantiations of classes. When a class that is defined in a class diagram of the model does not occur as class instantiation in a sequence diagram, this inconsistency is existent.

$$(\exists c \in C : \neg(\exists o \in O : class(o) = c))$$

under the assumption

$$(\forall o \in O : (\exists s \in S : (o, s) \in OS))$$

Object has no Class in CD (CnCD)
This inconsistency occurs if there is an object in a sequence diagram and no corresponding class is defined in any class diagram.

$$(\exists o \in O : \neg(\exists c \in C : class(o) = c))$$

under the assumption

$$(\forall o \in O : (\exists s \in S : \langle o, s\rangle \in OS))$$

Use Case without SD (UCnSD)
Sequence diagrams illustrate scenarios of use cases. Hence, the classes instantiated by a particular sequence diagram contribute to the functionality needed for the corresponding use case. This incompleteness exists when there is a use case that is not illustrated by any sequence diagram.

$$(\exists u \in U : \neg(\exists s \in S : \langle u, s\rangle \in US))$$

Multiple definitions of classes with equal names (Cm)
This inconsistency occurs when in the same model more than one classes have the same name. The different classes may be defined in the same diagram or in different diagrams.

$$(\exists c_1, c_2 \in C : name(c_1) = name(c_2) \wedge \neg(c_1 = c_2))$$

Defect	Questions	Symb.
EnN	Q1.1	yes
	-	no
EcM	Q8, R5*, R1.1*	yes
	Q2, R2*	no
ED	-	yes
	Q3	no
CnSD	Q9.2	yes
	Q5.1	no
CnCD	-	yes
	Q6.1, R4.1*	no
UCnSD	-	yes
	Q7.1	no
Cm	-	yes
	Q10	no
MnSD	R3	yes
	-	no
no	Q1.2, Q9.1	yes
defect	R1.2*, R4.2*, Q5.1 Q6.2, Q7.2	no

Note: Questions marked with an asterisk (*) are variations of the questions in the same row with the diagrams reversed.

Table 2: Nested Design with one Factor

Method not called in SD (MnSD)
This incompleteness occurs when there is a method of a class that is not called as a message in any sequence diagram.

$$(\exists m \in M : \neg(\exists e \in E : name(m) = name(e)))$$

The presented defects are caused by a mistake of the designer(s) or by improper use of the tooling.

3 Experiment Design

We performed two similar experiments. The subjects of the first experiment were students and the subjects of the second experiment were professionals. In this section the experiments are described. Most parts are the same for both experiments. The differences are explained where applicable.

3.1 Design

For each of the eight selected defect types that are presented in the previous section we have constructed an UML model fragment containing an instance

of the defect type. In the experiment the subjects were given the UML model fragments and the subjects' task was to indicate how he interprets the diagrams from the perspective of someone who must implement a system according to the given diagrams. Accompanying the diagrams we gave the subjects a question that focusses on a specific aspect of the model fragment. For each question there were four possible answer options, since the experiment was set up as a multiple-choice test. For each question about a model fragment containing an injected defect we constructed a similar control question that focusses on the same aspect but that does not contain a defect. Since the goal of the experiment is to investigate the effects of defects in UML models, the situation with defects must be compared to the situation without defects. The treatment in this experiment is the *defect injection* and the different levels are "no defect" and the eight defects defined in the previous section.

3.2 Question Design

The four answer options provided with each question were designed according to the following schema:

- *For questions containing a defect*: A defect between two or more diagrams means that there is conflicting information between the diagrams. The answer options are therefore designed such that for each of the diagrams there is at least one answer that corresponds to the system as described in the diagram. If possible, one answer option is a combination of the given diagrams, and at least one answer option is absolutely wrong according to all given diagrams. This is illustrated in the example in Section 3.2.1. The critical call is message `open()` from object `atm` to object `a`. Answer option A corresponds to the sequence diagram, options B and D correspond to the class diagram and option C is not according to any of the diagrams.

- *For questions not containing a defect*: one correct answer option and the other being wrong answers.

All questions had a fifth answer option that indicated that the subject could not give one of the other four possible answers because of an error or defect in the model fragment. Furthermore the subjects were asked to give a brief (textual) motivation of their answer.

When designing the experiment we felt that for some defects it would make a difference whether the subject is familiar with a specific problem domain and hence compensates a defect by applying domain knowledge. To enable us when analyzing the effect of domain knowledge as compensation for model defects we designed pairs of model fragments such that one version represents a system from a domain that every participant is familiar with (ATM machine and train crossing) and contained *meaningful* names (e.g. `account`, `customer`...) and the equivalent version contained *symbolic* (or meaningless) names (e.g. `Class A`, `method3`...).

To compensate for the effects of the order in which the diagrams of each model fragment are presented to the subjects we have asked some questions in the second run of the (student) experiment with the order of the diagrams swapped.

We assume that the subjects are not influenced in successor questions by treatments of previous questions, therefore the experiment is designed as a nested same-subject design, i.e. all subjects are exposed to all treatment levels. Hence, the design is by definition balanced. An overview of the experimental design is given in Table 2. The first column contains the defect types (see Section 2), the second column contains the identifiers of the questions questions marked with an asterisk are variants of previous questions with the order of diagrams reversed), and the third column indicates whether the questions (in the same row) are designed using diagrams with or without symbolic names. Questions whose identifier begins with a Q were asked in the first run of the experiment, the identifier R indicates that the question stems from the second run.

3.2.1 An Example Question

To illustrate the design of the questions we give question Q2 as presented in the experiment as an example. Q2 is a representative question for the whole questionnaire. In Figure 1 and Figure 2 the diagrams of a model fragments are shown as they were exposed to the subjects in the experiment. The question and answer options are as follows:

Question: *Suppose you are developer in this banking software project. It is your task to implement class ATM. Please indicate on the next page how you would implement the ATM class given these two UML diagrams?*

- **Answer option A**

```
Class ATM{
    method getCardInserted(){
        c.requestPIN();
        dosomething;
        a.open()}
    method acknowledge (){
        dosomething;
        c.seeMenu()}
}
```

- **Answer option B**

```
Class ATM{
    method getCardInserted(){
        c.requestPIN();
        dosomething;
        a.lock()}
```

```
    method acknowledge (){
        dosomething;
        c.seeMenu()}
}
```

- **Answer option C**

```
Class ATM{
    method getCardInserted(){
        c.requestPIN();
        dosomething;
        a.acknowledge()}
    method acknowledge (){
        dosomething;
        c.seeMenu()}
}
```

- **Answer option D**

```
Class ATM{
    method getCardInserted(){
        c.requestPIN();
        dosomething;
        a.validate()}
    method acknowledge (){
        dosomething;
        c.seeMenu()}
}
```

- **Answer option E** No interpretation possible because of an error in the model.

3.3 Subjects

3.3.1 Students

In total 111 students participated in the experiment. The experiment was conducted within the course "Software Architecture" [Cha04] at the Eindhoven University of Technology (TUE). This course is taught in the first year of the Masters program in computer science, hence all subjects hold a bachelor degree or equivalent. 85% of the subjects have a background in computer science, 10% in electrical engineering and 5% in other disciplines. 52% of the students received their bachelor from the TU Eindhoven, 31% from other Dutch institutions, 5% from universities in India and 12% from other countries. In the self assessment on the scale between 1 (no experience) and 5 (applied in many industrial projects) the results are shown in Table 3.

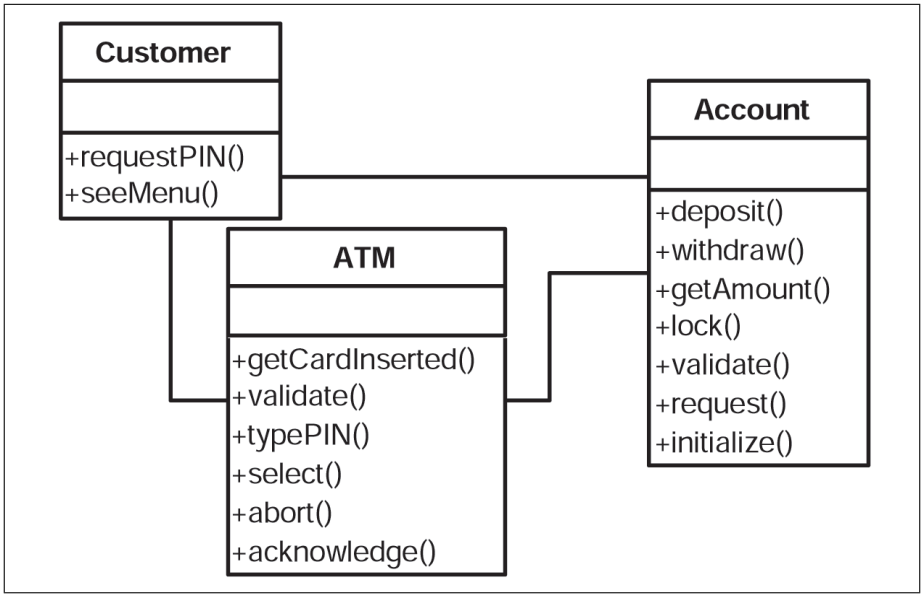

Figure 1: Example Class Diagram

3.3.2 Professionals

In total 48 professionals participated in the experiment by completing the online questionnaire. Some of the subjects did not complete all questions, but all questions were answered by at least 27 subjects. The background questions allowed us to get insight into the subjects' experience in relevant fields and, hence, to remove subjects with not enough experience to be regarded 'professional' in the context of this experiment. We removed subjects who entered 'student' as job description or who had less than two years of work experience. The average work experience of all remaining subjects is 10.7 years with a maximum of 35 years. The most frequent job descriptions (of all subjects who entered a job description) were 'architect', 'designer' and 'engineer'. In the self assessment on the scale between 1 (no experience) and 5 (applied in many industrial projects) the results are shown in Table 3.

3.4 Preparation

Prior to the experiment we conducted a pilot experiment to evaluate the experimental design and the experiment materials. Some colleagues and students participated in the pilot experiment, none of them was familiar with the goal of the experiment. The feedback of the pilot experiment led to improvements of the instruction sheet and to develop questions such that analysis with respect to compensation by domain knowledge was possible. Minor mistakes in the diagrams and text were found and corrected. The subjects took between 20

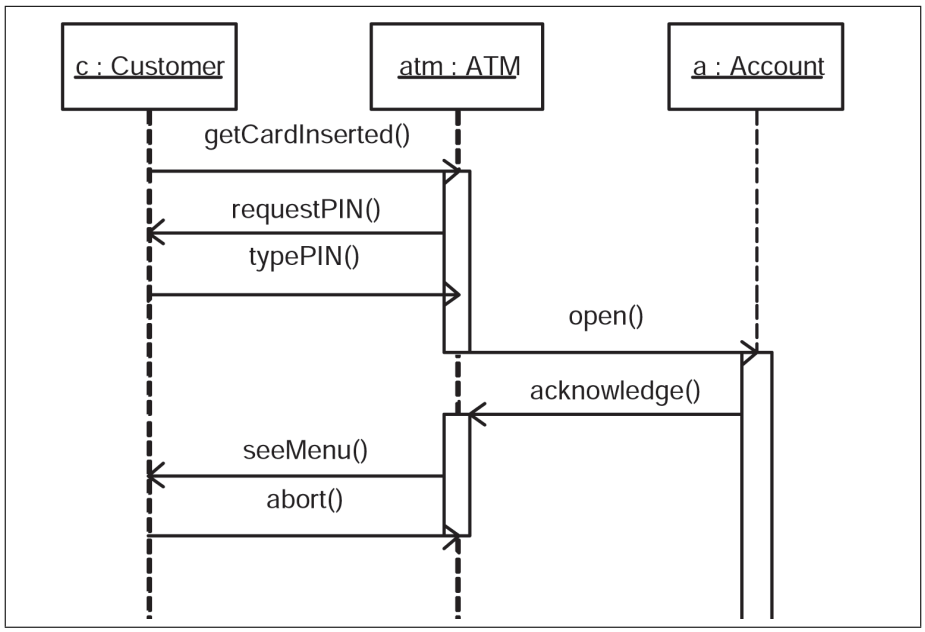

Figure 2: Example Sequence Diagram

minutes and 55 minutes to complete all questions.

All subjects of the student experiment were students of the course "Software Architecture" at the TUE. In this course UML was presented and explained to the students. In the five weeks before the experiment was conducted the students had to develop and evaluate a UML model as part of a design assignment and familiarized themselves with the UML (in case they had not been familiar with it before).

The professionals' experiment was conducted as an online questionnaire. Therefore besides setting up the website and the database to collect the results no preparation was needed.

Task	**Stud.**	**Prof.**
Designing	2.7	4.5
UML	2.4	4.3
Implementing	3.0	4.0
Code Review	2.5	3.8
Inspections	1.7	3.7
Design Review	2.2	3.4

Table 3: Subject Background according to Self Assessment (Averages)

3.5 Operation

3.5.1 Student Experiment

The student experiment was conducted in two runs. The first run contained questions Q1 to Q10, the second run was conducted five weeks later and contained questions R1 to R5. The procedure of operation was equal for both runs and is explained in the sequel.

Both runs were held as part of a test of the course. The incentive for the subjects was to gain bonus points for their grade by participating in the experiment. The subjects' achievement for the experiment questions had no influence on the grade. The experiment is according to the ethical issues addressed by Carver et al. [CJMS03].

The experiment was conducted in a classroom with the subjects spread out. The subjects were given the experiment material containing instructions, the diagrams of the model fragments, questions and answers options. For the test and the experiment the subjects had three hours available, the actual test was expected to take between 60 and 90 minutes, such that there was enough time to complete the experiment (which took less than 60 minutes in the pilot experiment) without time pressure. In addition to the written instructions we gave instructions at the beginning of the run. During the run the subjects were allowed to ask questions for clarification. The subjects were not familiar with the goal of the experiment to avoid biased results.

After the first run the subjects had to complete a questionnaire to assess their background with questions about their academic background, work experience, experience with UML and other relevant software engineering related topics.

3.5.2 Professionals Experiment

Because of the time constraints of professionals we performed only one experiment run in this group. The run contained questions Q1 to Q10. Since we intended to allow professionals from all organizations and from all over the world to participate in the experiment we executed the professionals' experiment as an online questionnaire. Subjects who prefer pen and paper for the experiment could download a printable version of the experiment material from the experiment website and fax or mail it to us. We announced the URL of the experiment website on several related newsgroups and asked industrial contacts to participate in the experiment and to forward the request to colleagues in their organization. The professionals' experiment contained the same diagrams and questions as the first run of the student experiment. The professionals' questionnaire also contained background questions to gain insight into the subjects' experience (which enabled us to remove results from subjects that could not be regarded as "professionals" in the sense of this experiment).

3.6 Threats to Validity

The threats to the validity [CC79] of the experiment include learning effects, fatigue, lack of generalizability of the student experiment and the size of the model fragments, and bias because of the multiple-choice design. These threats will be discussed in this section.

In both experiments described in this paper the order of the questions is the same for all subjects, hence there is a potential for order effects. Order effects occur when there is interaction between the objects of the study. To avoid interaction we constructed the objects, i.e. the model fragments, such that all fragments are in different domains, and the naming of elements (classes, methods...) is chosen such that each pair of model fragments has no common names for its elements. As described in Section 3.1 each question has three types of possible answers. To avoid that the subjects could predict the correct answer the order of the answers is chosen absolutely random. In all runs, there were no two questions with the same combination of injected defect and domain knowledge available, hence, we assume that there were no learning effects. The second run of the first experiment contained questions that were almost equal to questions from the first run The results were almost the same, hence there were no learning effects.

Fatigue during completion of the questionnaire is a possible threat to validity. The number of obvious wrong answers is almost the same for questions at the end of the questionnaire as it was for questions at the beginning of the questionnaire. Therefore there is no decrease in performance present.

The first experiment deploys students as subjects. This could be a threat for the external validity of the experiment. The subjects in the first experiment are all masters students with experience in UML. According to Kitchenham et al. [KPP+02] students can be used as subjects. Additionally we have performed the second experiment with professionals. The experiment is smaller in the number of subjects, but confirms the results of the larger first experiment (Section 5.4).

The size of the model fragments ranges between three and nine classes. In industrial models we have found between fifty and several hundred classes. Therefore the size can be considered as a threat to validity concerning the generalizability of the outcomes. For actions on industrial models (e.g. reading, modification) the designer focusses only on a subset of the model, which decreases the size gap for this experiment on cognitive effects. Since the complexity in industrial models is in general larger, the effects of defects in industrial models will therefore be at least as severe as the effects reported in this paper. In terms of number of classes industrial models are larger than our fragments by a factor ranging between ten and hundred. Similar experiments show the same size factor, e.g. Deligiannis et al.[DSA+04] have source code fragments of 17 and 18 classes and Purchase et al.[PCM+01] has a model fragment of ten classes. Multiplying the sizes of these experiments by a factor between ten and 100 yields sizes that are common source code sizes in industrial projects.

Since the experiment is designed as a multiple-choice test, four possible

Number of wrong answers	0	1	2	3	4 or more	
Run A	72	30	8	1	0	subjects
	64,9%	27,0%	7,2%	0,9%	0%	percentage
Run B	99	7	3	2	0	subjects
	89,2%	6,3%	2,7%	1,8%	0%	percentage

Table 4: Cumulative number of wrong answers per subject.

interpretations are explicitly stated to the subject. This situation is different from the situation in a real software development process, where the subject is not *guided* by a set of predefined interpretations. In practice, the subject has to choose from an infinite set of interpretations. Therefore the results in the experiment might differ from practice. Since the set of possible interpretations is in practice much larger and the subject is not guided and will most likely not *guess* the interpretation (which is possible in the multiple-choice test by a chance of .25), we expect the values for detection rate and agreement measure are even worse than in the experiment.

4 Results

The results of this experiment concerning defect detection are shown in table 5. The first column shows the identifier of the question, the second column shows the type of the defect (according to Section 2). The following columns give the results for the student experiment (S) and the professionals experiment (P). The column N shows the number of subjects participating in the question (since in both experiments some questions are not answered by all subjects the number of participants is not the same for all questions). The actual results are given in the columns *d-rate* (detection rate) and *AgrM* (Agreement Measure), both are explained in the sequel. In the column *Type* is indicated whether the question is a control question without defect (c), a symbolic question not allowing the subject to apply domain knowledge (s) and whether the question is a repetition of another question with the same defect with the diagram reversed (r). All questions (except for Q10) are paired with a control question that does not contain a defect. Note that some control question are used for more than one question. For a better readability the results of the control questions are repeated in the table after each question they are compared to.

In the Tables 6, 7 and 9 ten results are listed (the eight defect types plus two variants with symbolic names). The results of questions with reversed diagrams are omitted, because of the similarity to the results of questions with the same defect (see Section 5.3).

4.1 Data Purification

The results of this experiment might be biased by subjects with lack of motivation or with insufficient expertise to answer the questions. Therefore the answers of such subject should be excluded from the results. We analyzed the subjects' answer behavior to identify subjects with the mentioned characteristics.

The answers to the questions were designed such that questions concerning a defect have one obviously wrong answer and control questions have three obviously wrong answers. Hence, by giving a random answer, the probability to give an obviously wrong answer is .25 for questions concerning a defect and .75 for control questions. In the first run of the student experiment and in the professionals' experiment with 10 defected questions and 8 control questions subjects who are giving answers randomly would give on average 47% wrong answers, in the second student run with 7 defect questions and 2 control questions the average would be 36%. Table 4 shows the number of incorrect answers per subject for both runs of the student experiment. In both runs the maximum number of incorrect answers per subject is three, i.e. 16.7% of the 18 questions in the first run and 33.3% of the nine questions in the second run. Since for the first run this is below the expected average we conclude none of the completed questionnaires suffered from lack of motivation or insufficient knowledge. The same conclusion was drawn for the professionals' experiment (after subjects with insufficient experience were removed). In the second student run two subjects had three wrong ansers (33,3%). Such a high percentage might indicate, that the subjects were guessing. Therefore we removed the results of those subjects.

4.2 Defect Detection

One of our primary questions was to investigate whether developers detect defects in UML models. The detection rate of a question is the number of subjects that indicate that they cannot give an implementation due to a defect that is present divided by the total number of subjects that answered a question.

$$detection\ rate = \frac{\#\ of\ option\ E\ answers}{\#\ of\ subjects}$$

Ideally, if a defect is present, it should be detected and no implementation should be given. Given the motivations of the subjects' answers, we regard the case where multiple answers are given also as defect detection (by giving multiple answers the subject indicates that the underspecification or ambiguity has been detected). In the case where no defect is present, all subjects should ideally give the same implementation.

The data of all defect types from Table 5 is summarized in Figure 3. The boxplot shows the detection rate for questions containing a defect compared to control questions. In case of a control question the vast majority of the subjects gives an implementation and only a small fraction wrongly detects a (non present) defect. The figure shows that even in case of a model defect, more

Quest.	Defect	N		d-rate		AgrM		Type		
		S	P	S	P	S	P	s	r	c
Q1.1	Message without Name	111	48	.69	.60	.47	.44	√		
Q1.2		110	48	.08	.16	.97	.89	√		√
Q2	Message without Method	111	40	.39	.38	.84	.90			
Q1.2		110	48	.08	.16	.97	.89			√
Q8	Message without Method	111	30	.49	.33	.86	.94	√		
Q4		111	34	.05	.13	.95	.91	√		√
R2	Message without Method	109		.36		.89			√	
Q4		111	34	.05	.13	.95	.91			√
R5	Message without Method	108		.34		.95		√	√	
Q1.2		110	48	.08	.16	.97	.89	√		√
R1.2	Message without Method	110		.49		.69		√		
R1.1		110		.06		.92		√		√
Q3	Message in the wrong direction	111	34	.60	.58	.47	.95	√		
Q1.2		110	48	.08	.16	.97	.89	√		√
Q6.1	Object has no Class in CD	111	31	.18	.11	.83	.93			
Q6.2		111	31	.08	.18	.77	.70			√
R4.1	Object has no Class in CD	108		.21		.86			√	
R4.2		108		.14		.77			√	√
Q7.1	Use Case without CD	109	30	.50	.52	.83	.44			
Q7.2	Use Case without CD	110	30	.05	.22	.95	.92			√
Q5.2	Class not instantiated in SD	111	34	.47	.68	.49	.64			
Q5.1		111	34	.05	.13	.94	.95			√
Q9.2	Class not instantiated in SD	109	30	.95	.96	.54	.14	√		
Q9.1		110	30	.04	.07	.99	1.0	√		√
Q10	Multiple Class defs.	107	27	.10	.33	.92	.68			
R3	Method not called in SD	109		.14		.67		√		
Q1.2		110	48	.08	.16	.97	.89	√		√

Table 5: Complete Detection Rate and AgrM Results

Defect	S	P	Quest.
Multiple Class Defs	.10	.33	Q10
Method not in SD (s.)	.14	n/a	R3
Object has no Class in SD	.18	.11	Q6.1
Msg. without Method	.39	.38	Q2
Class not in SD	.47	.48	Q5.2
Msg. without Method (s.)	.49	.33	Q8
UC without SD	.50	.52	Q7.1
Msg. in the wrong Dir. (s.)	.60	.48	Q3
Message without Name (s.)	.69	.60	Q1.1
Class not in SD (s.)	.95	.96	Q9.2

Table 6: Defects sorted by detection rate

subjects indicate to give a implementation than detect the error. Defect types marked with an '*s.*' are based on model fragments with symbolic names.

Table 6 shows the defect types ordered by detection rate (of the student experiment), defect types at the top of the table remain undetected in most cases.

4.3 Variation of Interpretations

The effect of an undetected defect is not necessariliy negative, in case everybody has the same interpretation of the model there is no problem. But since defects are in most cases mismatches between diagrams, it is possible that conflicting information leads to different interpretations. To measure the degree of spread over the four possible options of the answers we have develeped a so called agreement measure ($AgrM$). The agreement measure is 0 if all options receive the same number of answers (multiple interpretations) and 1 if only one option receives all answers (one interpretation). AgrM is intuitively explained in Figure 5 and is explained in more detail in the Appendix.

The data from Table 5 is summarized in Figure 4. In the boxplot the agreement measure of questions containing a defect compared to control questions is shown. The control questions have a high AgrM value, which indicates that most subjects have the same interpretation of the model. The AgrM value of defected models are widely spread between .14 and 1.00. The results show that in general defected models cause a larger variety of misinterpretations and, hence, contain a higher risk for misinterpretation and miscommunication.

Table 7 shows the defect types ordered by AgrM (of the student experiment), defect types with the largest spread over different interpretations are at the top of the table.

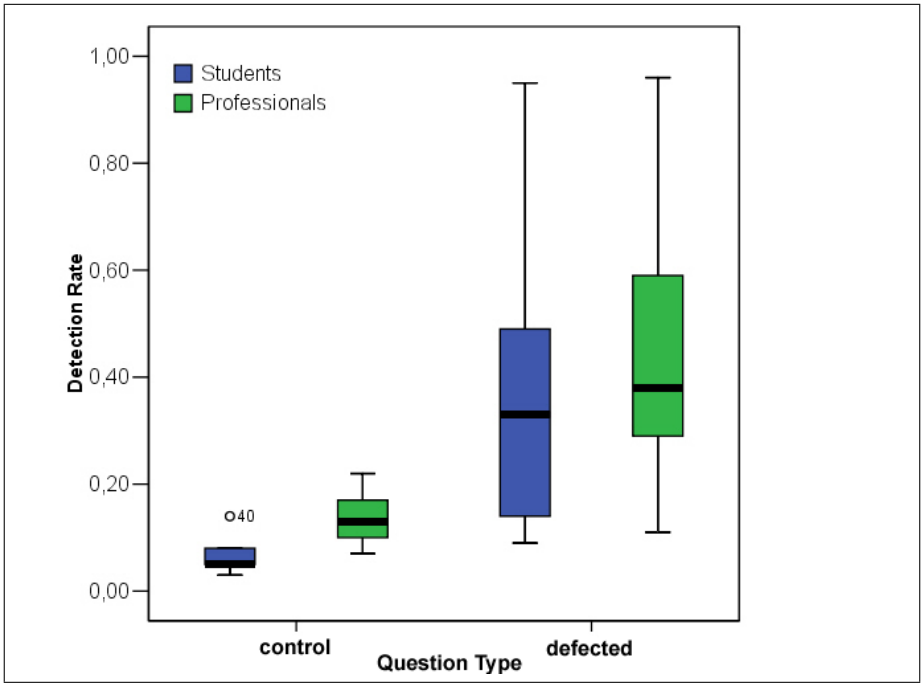

Figure 3: Defect detection boxplot

4.4 Severity

The overall severity of a defect type depends on the likelihood to which a defect is detected and the degree of misunderstanding that it possibly causes. Table 8 shows four risk categories:

1. defect types with low detection rate and AgrM cause the highest risk, because they are unlikely to be detected and they cause a large amount of misinterpretation,
2. defect types with low detection rate and high AgrM cause medium risk, because they are unlikely to be detected, but do not cause a lot of misinterpretations,
3. defect types with high detection rate and low AgrM cause medium risk, because they are likely to be detected, but if they are not detected a large amount of misinterpretation is caused,
4. defect types with high detection rate and high AgrM cause hardly any risk, because they are likely to be detected and if they remain undetected, there is agreement upon the interpretation.

The cases where only one measure has a high value cause less risk, since either the defect is likely to be detected or the defect does not cause misunderstandings.

Defect	S	P	Quest.
Class not in SD (s.)	.34	.14	Q9.2
Msg. without Name (s.)	.47	.44	Q1.1
Msg. in the wrong Dir. (s.)	.47	.95	Q3
Class not in SD	.49	.64	Q5.2
Method not in SD (s.)	.67	n/a	R3
Object has no Class in SD	.83	.93	Q6.1
UC without SD	.83	.44	Q7.1
Msg. without Method	.84	.90	Q2
Msg. without Method (s.)	.86	.94	Q8
Multiple Class Defs	.92	.68	Q10

Table 7: Defects sorted by AgrM

	d-rate low	d-rate high
AgrM low	high risk (1)	medium risk (3)
AgrM high	medium risk (2)	no risk (4)

Table 8: Severity - Risk Matrix

We define the overall severity of a defect as the product of the detection rate and AgrM. In table 9 the defects are ranked according to their severity value (of the student experiment). A scatterplot of the severity in the student experiment is in Figure 6 and of the professionals' experiment is in Figure 7.

5 Discussion

5.1 Domain Knowledge

When reading a text that contains errors it is often possible to understand the intended meaning of the text. Understanding the right meaning is sometimes even possible if the errors introduce ambiguity or change the meaning. This is based on the fact, that the reader knows the language and he is supported by the fact, that he is familiar with the context, and, hence, can infer the correct meaning from the context.

In this study we investigate the effects of defects in UML models. Hence we are also interested whether context knowledge enables the reader to infer the right interpretation from the defected model. To be able to analyze the use of context (or domain) knowledge, we designed pairs of models for the defects *Message name does not correspond to method name* (EcM) and *Class from SD not in CD* (CnSD) such that one model was taken from a familiar domain (ATM machine and train crossing) and the other model is essentially equal, but the elements have symbolic names without a particular meaning (e.g. `class A`,

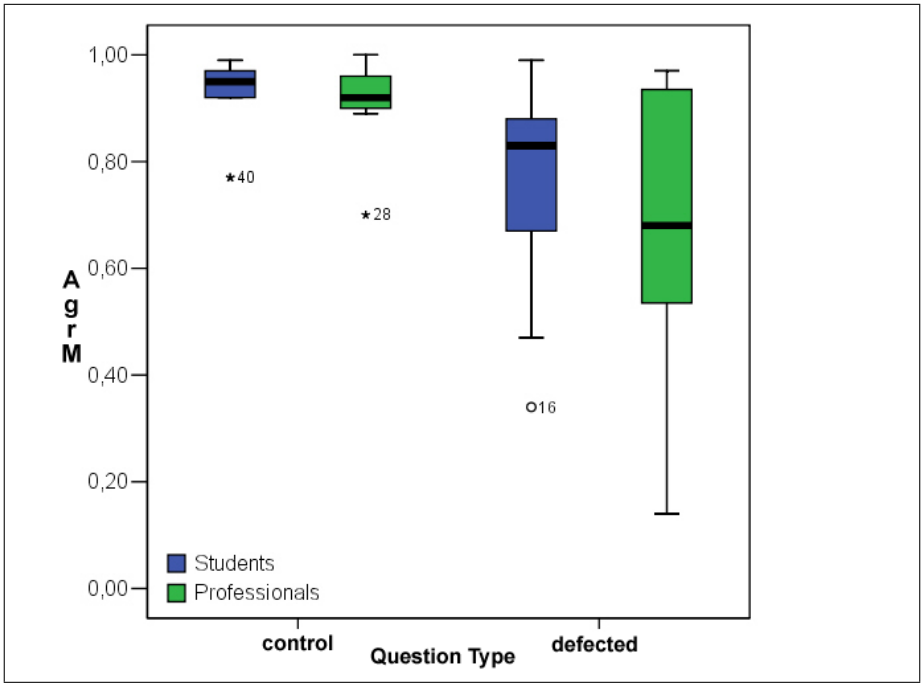

Figure 4: Agreement boxplot

`method3`).

For the defect EcM the results of students and professionals are almost the same in the cases with and without domain knowledge (see Table 5). For the defect CnSD there is a large difference between the model with and without context for the detection rate as well as for AgrM in both groups of subjects (Figure 8). When the reader cannot use domain knowledge to compensate the defect CnSD the detection rate is higher, i.e. 95% of the students and 96% of the professionals detect the defect. The subjects that compensate the defect using their domain knowledge do have different interpretations of the model, resulting in low scores for AgrM.

The question for this defect (using domain knowledge) was to describe the behavior of the classes that control the traffic light events based on events from the gate sensors and the rail sensors. Because the order of events at traincrossings might be slightly different in different countries, we analyzed the results to detect whether subjects from the same country (i.e. having common domain knowledge) would have the same interpretation, but even this was not the case.

5.2 Leading Diagram and other Observations

The questions about defect types that represent an inconsistency or an incompleteness between a sequence diagram and a class diagram were designed in such

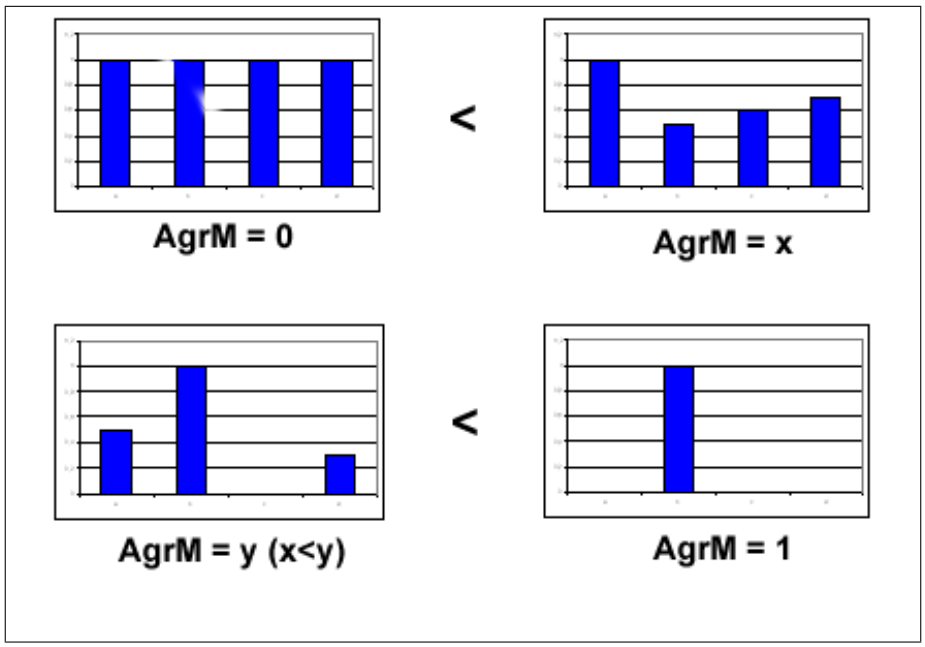

Figure 5: Intuition of AgrM measure

a way, that the answer options included at least one option that compensated the defect by regarding the sequence diagram as correct and at least one option that compensated the defect by regarding the class diagram as correct. Interestingly for all defect types that allow either way of interpretation, the option regarding the sequence diagram as correct (i.e. *leading*) received the largest amount of answers. These defect types are: ED, MnSD, CnCD and EcM.

The defect types Class not in SD (CnSD) and Message without Name (EnN) also address sequence diagrams and class diagram, but in these cases there is information missing in the sequence diagrams (instead of a mismatch). Therefore they cannot be compensated by using the sequence diagram. CnSD is discussed in 5.1. The majority of subjects compensates a defect of type EnN using the most straightforward interpretation of the class diagram (not taking into account inheritance). But the degree of misinterpretation induced by this defect type is rather high (i.e. low AgrM values: .47 resp. .44).

The defect type *Multiple Class Definitions under the same Name* involves only class diagrams. Most subjects compensate an instance of this defect by taking the union of all methods and classes of both definitions of the class.

The defect type *UC without SD* has a very large difference in AgrM between students and professionals (.83 vs. .44). We could not find a reason for this large difference.

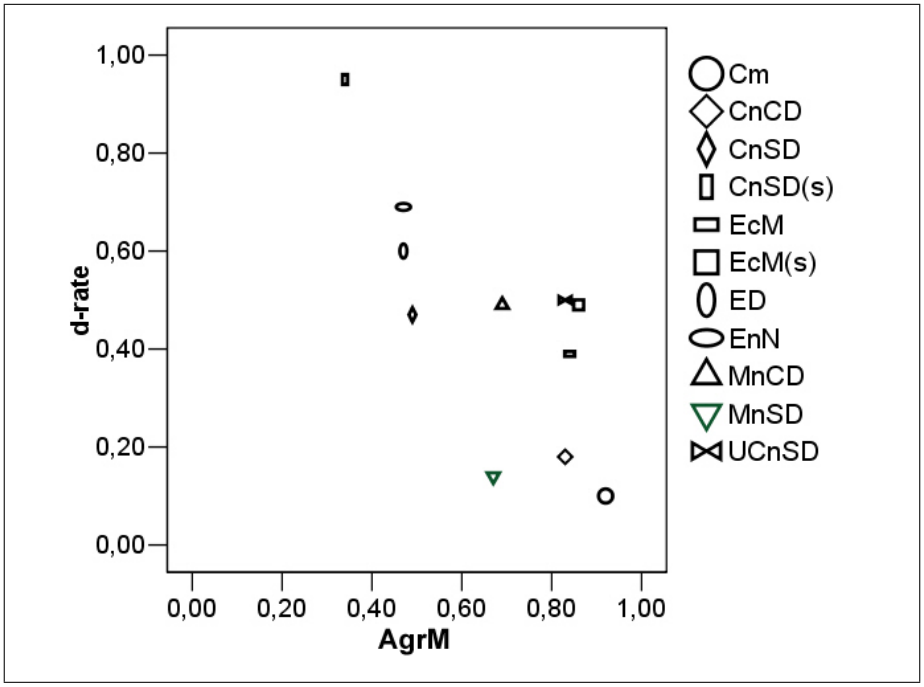

Figure 6: Scatterplot of Severity (Students)

5.3 Order of Diagrams

In the first run of the student experiment all questions addressing a model consisting of a sequence diagram and a class diagram were presented to the subjects such that the sequence diagram was the first (i.e. leftmost) diagram. To make sure that this order of presentation was not the cause of the leading-diagram-effect, we performed a second run with a subset of the questions. In the second run the order of presentation was changed, such that the class diagram was the leftmost diagram. The results of the second run confirmed the observation, that the sequence diagram was regarded as the leading diagram by the majority of the subjects. The Pearson correlation between the results of the first and second run are .913 for the detection rate (p-value .011) and .638 for AgrM (not significant). The Pearson correlation between the results of the second run and the professionals experiment are .824 (.044) for the detection rate and for .899 for AgrM (.015).

5.4 Generalizability

The presented results are based on a large group of students (111 subjects) and a smaller group of professionals (between 27 and 48 subjects, depending on the question). The larger amount of subjects in the student group results in a higher

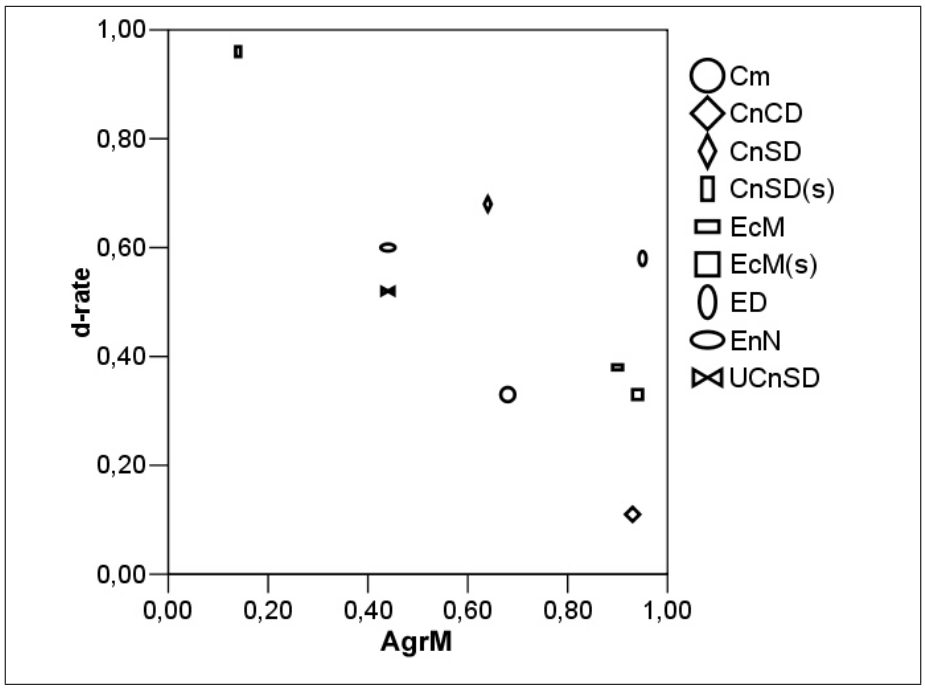

Figure 7: Scatterplot of Severity (Professionals)

reliability of the results obtained from this group. The drawback of experiments with students is that the results are possibly not generalizable to professionals because of the different experience level. Students are less experienced than professionals and in an experiment their pressure and motivation differs from those of professionals in their daily work.

To investigate whether the results obtained from the student group can be generalized to professionals we compared the results of both groups. We use Pearson's correlation to investigate to which degree the results of the professionals are related. The correlations are shown in table 10. There are strong correlations between detection rate and AgrM of students and practitioners. The results are statistically significant. Therefore the reliable results of the student group can be generalized to professionals without lost of validity.

6 Conclusions and Future Work

6.1 Conclusions

In this study we investigated the effects of defects in UML models. The two major contributions are the investigations into defect detection and misinterpretations caused by undetected defects. The results show that some defect types are detected by almost all subjects (e.g. 96% of the subjects detect *Class*

Defect	S	P	Quest.
Method not in SD (s.)	.09	n/a	R3
Multiple Class Defs	.09	.23	Q10
Object has no Class in SD	.15	.10	Q6.1
Class not in SD	.23	.44	Q5.2
Msg. in the wrong Dir. (s.)	.29	55	Q3
Class not in SD (s.)	.32	.14	Q9.2
Msg. without Method	.32	34	Q2
Msg. without Name (s.)	.33	.26	Q1.1
UC without SD	.42	.23	Q7.1
Msg. without Method (s.)	.42	.31	Q8

Table 9: Defects sorted by Severity

	r	p-value
Detection Rate	.929	$< .001$
AgrM	.779	$< .001$

Table 10: Pearson's Correlation between Student and Professional Results

not in Sequence Diagram) whereas other defect types are hardly detected (e.g. *Multiple Definitions of the same Class* is detected by 10% only). Most of the analyzed defect types are detected by less than 50% of the subjects. For the risk for misinterpretations results are similarly alarming. Some defect types cause a distribution over several interpretation amongst readers (e.g. *Class not in Sequence Diagram* has an AgrM of 0.14) and other defect types do hardly cause any misinterpretations (e.g. *Message without Method* has a AgrM of 0.94). We have presented a ranking of defect types according to detection rate and risk for misinterpretations. The results show that most defect types are hardly detected and that there is no implicit consensus about the interpretation of undetected defects. Therefore defects are potential risks that can cause misinterpretation and, hence, miscommunication. The obtained results are generalizable to professional UML users. Because of the described problems, that are caused by defects, it is recommended to identify and remove defects in UML models. Defect prevention is an even better approach.

In this study we present a ranking of defects according to detection rate and risk for causing misinterpretation.

Furthermore we observed that the presence of domain knowledge has effect on the interpretation of UML models. We found an instance of a defect type where the presence of domain knowledge strongly decreased the detection rate. This observation gives rise to the assumption that domain knowledge supports implicit assumptions that might be wrong and cause misinterpretations. The validity of this assumption should be investigated in further studies.

We observed the strong tendency that defect types that involve sequence

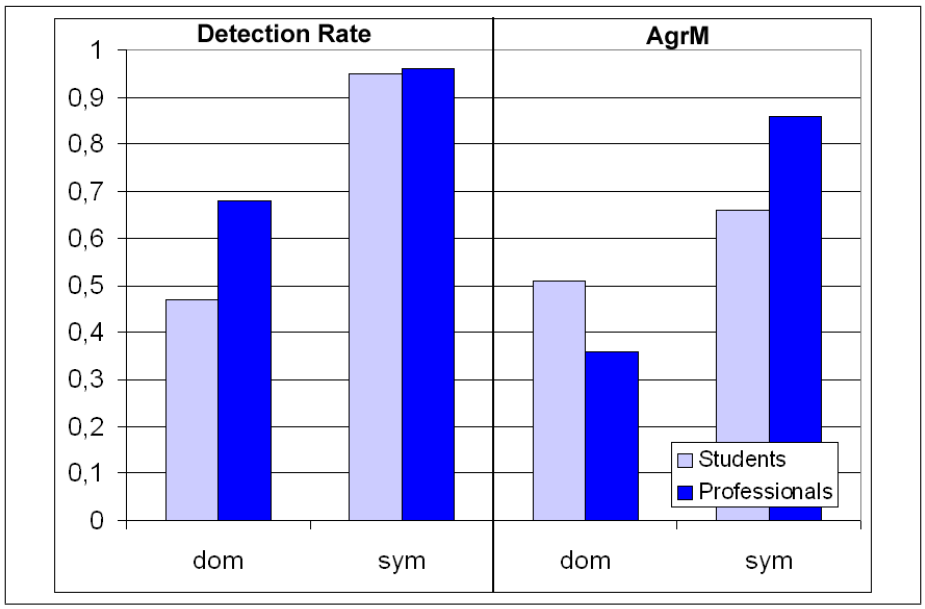

Figure 8: Results for Defect CnSD

diagrams and class diagrams are often compensated by regarding the sequence diagram as leading. Our results show that the leading diagram does not depend on the presentation order of the diagrams.

6.2 Future Work

In this study we focus on defects in class diagrams, sequence diagrams and use case diagrams. In these diagrams there exist more defect types and a large set of defect types exist when taking other diagram types into account. Therefore, a larger set of defect types should be analyzed in quasi replications of this study. In our research we focus on the likeliness of detection and the potential for misinterpretation. In further studies the impact of misinterpretations should be investigated. For example questions like "which implementation errors will be caused by model defects?" and "when will errors caused by model defects be detected and what is the cost of repairing them?" should be addressed. In this project the effect of instances of a single defect type in isolation were addressed. In practice defects do not occur in isolation. Further studies should investigate which combinations of defects are likely and whether there is interaction between defects such that the effects might be amplified or even decreased. We invite other researchers to replicate this experiment using other groups of subjects.

6.3 Acknowledgements

We thank the students of the course "Software Architecting" (2004/2005) at the TU Eindhoven and the anonymous professionals for participating in the experiments. Fruitful discussions with our colleagues Reinder Bril, Johan Muskens and Teade Punter and their reviews contributed to the improvement of this paper.

References

[BBD+00] Eerke Boiten, Howard Bowman, John Derrick, Peter Linington, and Maarten Steen. Viewpoint consistency in ODP. *Comput. Networks*, 34(3):503–537, 2000.

[BCR94] Victor R. Basili, G. Caldiera, and H. Dieter Rombach. The goal question metric paradigm. In *Encyclopedia of Software Engineering*, volume 2, pages 528–523. John Wiley and Sons, Inc., 1994.

[CC79] T. D. Cook and D. T. Campbell. *Quasi-Experimentation - Design and Analysis Issues for Field Settings*. Houghton Mifflin Company, 1979.

[Cha04] Michel R. V. Chaudron. Software architecting. `http://www.win.tue.nl/~mchaudro/sa2004/`, 2004. TU Eindhoven.

[CJMS03] Jeffrey Carver, Letizia Jaccheri, Sandro Morasca, and Forrest Shull. Issues in using students in empirical studies in software engineering education. In *Proceedings of The Ninth International Software Metrics Symposium*, pages 239 – 249, September 2003.

[DSA+04] Ignatios Deligiannis, Ioannis Stamels, Lefteris Angelis, Manos Roumeliotis, and Martin Shepperd. A controlled experiment investigation of an object-oriented design heuristic for maintainability. *Journal of Systems and Software*, 2(72):129–143, 2004.

[KHR+03] L. Kuzniarz, Z. Huzar, Gianna Reggio, Jean-Louis Sourrouille, and Miroslav Staron. *2nd Workshop on Consistency Problems in UML-based Software Development at the UML2003*. Blekinge Institute of Technology, 2003.

[KPP+02] Barbara A. Kitchenham, Shari Lawrence Pfleeger, Lesley M. Pickard, Peter W. Jones, Dafic C. Hoaglin, Khaled El Emam, and Janett Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions of Software Engineering*, 28(8):721–734, August 2002.

[Lan03] Christian F. J. Lange. Empirical investigations in software architecture completeness. Master's thesis, Technische Universiteit Eindhoven, Den Dolec 2, 5600MB Eindhoven, The Netherlands, September 2003.

[LC04] Christian F. J. Lange and Michel R. V. Chaudron. An empirical assessment of completeness in UML designs. In *Proceedings of the 8th International Conference on Empirical Assessment in Software Engineering (EASE'04)*, pages 111–121, 2004.

[MCB05] Johan Muskens, M. R. V. Chaudron, and R. J. Bril. Finding inconsistencies between views using relation partition algebra. CS Report 05/01, TU Eindhoven, Dep. of Mathematics and Computing Science, 2005.

[Obj03a] Object Management Group. *Unified Modeling Language, Adopted Final Specification, Version 2.0*, ptc/03-09-15 edition, December 2003.

[Obj03b] Object Management Group. *Unified Modeling Language, Specification, Version 1.5*, formal/03-03-01 edition, March 2003.

[PCM+01] Helen C. Purchase, Linda Colpoys, Matthew McGill, David Carrington, and Carol Britton. UML class diagram syntax: an empirical study of comprehension. In *Australian symposium on Information visualisation*, volume 9, pages 113–120, September 2001.

[Sha48] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, July 1948.

[Tec] Technische Universiteit Eindhoven, http://www.win.tue.nl/empanada. *The EmpAnADa Project.*

A The Agreement Measure

For the analysis of the multiple-choice questions we were interested in the amount of misinterpretation that was caused by the presence of a model defect. Therefore we needed a measure that captures the degree of agreement in the distribution of answers to each question. In our experiment each question has four answer alternatives. Essentially we want to measure a distributions' magnitude of discrimination.

The measure should have the following properties:

Property 1 *It should have its maximum value, when only one alternative received answers.*

Property 2 *It should have its minimum value, when all alternatives received an equal number of answers.*

Property 3 *It should have the range* $[0, 1]$.

Property 4 *It should be larger for distributions where X alternatives received 'many' answers than for distributions where X+1 alternatives received 'many' answers.*

Our agreement measure captures the attribute of a distribution that is the inverse of entropy. We also considered to use Shannon's entropy measure [Sha48]. The main difference between Shannon's entropy and our agreement measure is that Shannon's entropy measure behaves exponential and is not normalized between 0 and 1. The linear behavior and the range between 0 and 1 of AgrM is similar to the detection rate. Therefore we have chosen to develop a new measure as described below.

First we define some abbreviations that we need for the explanation of the measure (see Table 11).

Name	Description
K	the number of alternatives for a question
k_i	the number of times alternative i was selected, where $0 \leq i < K$ and $(\forall i : 0 \leq i < K - 1 : k_i \geq k_{i+1})$
N	the sum of answers over all alternatives: $N = \sum_{0 \leq i < K} k_i$

Table 11: Definitions for the Agreement Measure

For simplicity's sake we begin by developing a function that behaves opposed to the desired function, i.e. it has its maximum value when all alternatives receive an equal number of answers (opposed to Property 1).

The function must be such that it is larger, the broader the distribution of alternatives is. This means the value must increase, if the number of alternatives that received answers increases and if difference in number of answers between the alternative with the most answers and the other alternatives decreases. We accomplish this by multiplying the number of answers per alternative by a factor and adding the products. The factors are chosen such that alternatives with fewer answers receive a larger factor. This leads to the following equation for the weighted sum S, where we choose 0, 1... K-1 as factors:

$$S = \sum_{0 \leq i < K} k_i i \tag{2}$$

In the optimal case only one alternative receives answers ($k_0 > 0$) and all other alternatives receive no answers (for $i > 0$ is $k_i = 0$)), hence the sum $S = 0$.

In the worst case, the answers are equally distributed over all alternatives k_i. In this case the sum reaches its maximum S_{max} which can be calculated as follows:

$$S_{max} = \frac{N}{K} \sum_{0 \leq i < K} i = \frac{N}{K} \frac{K(K-1)}{2} = \frac{N(K-1)}{2} \tag{3}$$

We normalize the range of our measure such that it satisfies Property 3 by dividing S through S_{max}. This yields:

$$F = \frac{S}{S_{max}} \tag{4}$$

Now we have F which has Property 4 and Properties 1 and 2 opposed. We simply have to subtract F from 1 to obtain our measure which also has Properties 1 and 2.

Hence, our agreement measure (called $AgrM$) for $K > 1$ alternatives is:

$$AgrM(k_0, ..., k_{K-1}) = 1 - 2\frac{\sum_{0 \leq i < K} k_i i}{N(K-1)} \tag{5}$$

Question	Defect Type	A	B	C	D	detected
Q1.1	Message without Name	2	38	31	63	77
Q1.2	Control	2	100	1	0	9
Q2	Message without Method	71	6	0	9	43
Q3	Message in the wrong Direction	34	7	8	8	67
Q4	Control	2	106	2	1	6
Q5.1	Control	1	103	3	2	5
Q5.2	Class not instantiated in SD	18	53	4	39	52
Q6.1	Class not in CD	3	96	1	23	20
Q6.2	Control	0	78	42	0	9
Q7.1	UC not in SD	1	1	43	9	55
Q7.2	Control	0	100	1	6	5
Q8	Message without Method (s.)	58	2	7	1	54
Q9.1	Control	1	105	0	0	4
Q9.2	Class not instantiated in SD (s.)	30	33	1	31	104
Q10	Multiple class defs.	4	6	100	0	11

Table 12: Raw Results of the Student Experiment (First Run)

B Raw Result Data

The raw results of the student experiment can be found in Tables 12 (first run) and Table 14 (second run). Table 13 shows the raw results of the professionals' experiment.

Question	Defect Type	A	B	C	D	detected
Q1.1	Message without Name	2	7	7	16	27
Q1.2	Control	1	32	1	1	7
Q2	Message without Method	25	1	0	2	14
Q3	Message in the wrong Direction	13	9	1	0	18
Q4	Control	2	28	1	0	4
Q5.1	Control	0	25	2	0	4
Q5.2	Class not instantiated in SD	1	9	1	4	21
Q6.1	Class not in CD	1	26	0	1	3
Q6.2	Control	0	17	4	0	5
Q7.1	UC not in SD	0	5	5	3	14
Q7.2	Control	0	23	1	1	6
Q8	Message without Method (s.)	21	0	2	0	9
Q9.1	Control	0	25	0	0	2
Q9.2	Class not instantiated in SD (s.)	2	2	1	2	26
Q10	Multiple class defs.	2	7	14	0	6

Table 13: Raw Results of the Professionals' Experiment

Question	Defect Type	A	B	C	D	detected
R1.1	Control	102	4	3	1	7
R1.2	Message without Method (s.)	22	2	22	61	54
R2	Message without Method	71	2	1	6	36
R3	Method not instantiated in SD	41	68	6	3	15
R4.1	Class not in CD	5	93	0	14	23
R4.2	Control	0	76	39	0	15
R5	Message without Method (s.)	1	0	5	81	33

Table 14: Raw Results of the Student Experiment (Second Run)