Effects of Defects in UML Models – An Experimental Investigation

Christian F.J. Lange
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven, The Netherlands
C.F.J.Lange@tue.nl

Michel R.V. Chaudron
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven, The Netherlands
M.R.V.Chaudron@tue.nl

ABSTRACT

The Unified Modeling Language (UML) is the de facto standard for designing and architecting software systems. UML offers a large number of diagram types that can be used with varying degree of rigour. As a result UML models may contain consistency defects. Previous research has shown that industrial UML models that are used as basis for implementation and maintenance contain large numbers of defects. This study investigates to what extent implementers detect defects and to what extent defects cause different interpretations by different readers. We performed two controlled experiments with a large group of students (111) and a group of industrial practitioners (48). The experiment's results show that defects often remain undetected and cause misinterpretations. We present a classification of defect types based on a ranking of detection rate and risk for misinterpretation. Additionally we observed effects of using domain knowledge to compensate defects. The results are generalizable to industrial UML users and can be used for improving quality assurance techniques for UML-based development.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*object-oriented design methods, computer-aided software engineering*

General Terms

Experimentation, Design, Documentation, Measurement

Keywords

Completeness, consistency, defect detection, UML

1. INTRODUCTION

The Unified Modeling Language is the de facto standard for designing and architecting software systems. It lacks a formal semantics and allows a large degree of freedom in modelling. As a result, the UML is applied in many different ways. Additionally the UML offers various diagram types to describe different views of a system. UML 1.x [25] offers 9 diagram types and in the new UML 2.0 [24] the designer can even choose from 13 diagram types. A UML model consists of a set of diagrams that together describe a single system. Hence, there may be overlap between different diagrams. The overlapping parts between diagrams contain the risk of consistency defects. A consistency defect is a mismatch between overlapping diagrams. There are several approaches that classify these defects [10]. Additionally there are several methods that aim at finding, removing and preventing defects in UML models. These include reading techniques [18, 8, 27] and techniques based on formalization [17, 23].

Notwithstanding advances in these areas as well as in modern UML case tools, the number of defects that remain undetected in practice is alarmingly large. This is reported by empirical studies such as [20] where a large amount of defects was found in several industrial UML models.

UML models are used for describing solutions, analyzing their properties, and as a means for communication between stakeholders. Defects in UML models are likely to affects these uses, but no reseach on such effects is known. This paper describes our investigation into the effects of defects in UML models. We consider two main research questions:

- **RQ1:** Are defects in UML models detected by implementers?
- **RQ2:** How do undetected defects impact the interpretation of the model by different implementers?

The goal is summarized by the GQM template [5] (Table 1).

To answer these questions we have conducted an experiment with 111 students and replicated the experiment with 48 professionals. Section 2 discusses related work. Section 3 explains the defect types that we have investigated. In Section 4 the experimental design, its execution and the two groups of subjects are described. The results and major findings of the experiment are presented in Section 5. Additional observations are discussed in Section 6. Concluding remarks and future work are given in Section 7.

2. RELATED WORK

There exists only limited research on the effects of defects that occur in UML models at the stage of implementing the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICSE'06, May 20–28, 2006, Shanghai, China
Copyright 2006 ACM 1-59593-085-X/06/0005 ...$5.00.

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** masters students at the TU Eindhoven and professionals

Table 1: Goal according to GQM template

system. Therefore we have widened our literature review to defect detection techniques and defect classification approaches. This enabled us to compare our work to previous research.

Software inspection [12] is an efficient and effective means for quality improvement in software engineering. Defect detection is besides planning, defect collection and defect correction a central activity in software inspection [19]. In inspections so called reading techniques are applied for defect detection. The most applied reading techniques in industry are ad hoc reading and checklist-based reading (CBR) [12, 19]. Ad hoc reading is not structured and does not provide the inspector with any advice on how to proceed, whereas in CBR checklists of yes/no questions are used to guide the inspector. Fagan [13] and Gilb and Graham [14] claim that inspection leads to the detection of 50% to 90% of the defects in a software document. These results stem from the time before the UML existed.

Few results exist where inspection techniques for UML and related modeling techniques are analyzed. Since UML is a modelling language based on object-oriented concepts, the more recent object-oriented reading techniques (OORT) such as scenario-based reading (SBR) [3] seem more applicable for inspection of UML models. Scenario-based reading techniques provide the inspector with so-called *scenarios* that describe what must be checked and how to perform a check. One of the specializations of scenario-based reading techniques is perspective-based reading (PBR) [4] where the scenarios are based on a particular stakeholder's perspective.

The literature review shows that initial empirical research has been done on applying reading techniques to UML models. Laitenberger et al. [18] conducted an experiment with 18 practitioners as subjects to compare checklist-based reading with perspective-based reading for UML (both reading techniques applied in teams). The results show that perspective-based reading leads to detecting 41% more defects than checklist-based reading and perspective-based reading is less cost-intensive than checklist-based reading. The defect detection effectiveness using perspective-based reading is between 45% and 75%. Wohlin and Aurum [27] conducted an experiment with 486 students as subjects to evaluate checklist-based reading for entity-relationship diagrams, which are comparable to UML class diagrams. The results show that the median effectiveness is 46%. Conradi et al. [8] investigated cost-efficiency of reading techniques in an industrial experiment. Since their measurements are based on industrial UML models and the total number of defects is not known they cannot report about defect detection effectiveness like the other studies. However they report cost-effectiveness in terms of defects detected per person hour (d/ph). For individual readers the cost-effectiveness is around 1.7 d/ph and for team meetings it is less than 1 d/ph. The range of the reported values for the effectiveness of defect detection is large. However, even in the best reported case, 25% of the defects remain undetected and propagate to later phases. The experiences from our industrial case studies [20] show that still a large number of defects exist in industrial UML models and not all organizations are using defect detection techniques for UML models. Therefore an investigation like described in this paper of the effects of defects in UML models that propagate to later phases is needed. Our study does not investigate inspection or OORT's, but the effect of defects that outlive inspection.

Defect classification is a means to assign priorities to defects to enable cost-effective resource-usage to fix defects ('most severe defects first'). Defect classifications are subjective [11] and many organizations use simple categories such as Minor, Major or Severe [15]. Since classifications are subjective, there are approaches that provide guidelines to enable repeatable defect classification such as IBM's Orthogonal Defect Classification (ODC) [7] and an improvement variant of it by El Emam et al. [11]. These approaches are not specifically validated for UML defects. The aforementioned study by Wohlin and Aurum [27] is the only work found reporting a defect classification for ER-diagram defects. However, the classification reported there is subjective and no guideline is given on how it can be repeated. There is no literature known to the authors describing a defect classification based on empirical validation.

3. DEFECT TYPES

A UML model consists of several diagrams and these diagrams may have a certain overlap. If overlapping parts of diagrams conflict, then the model contains a consistency defect. The defect types analyzed in this study take into account a subset of the UML diagrams: class diagrams, sequence diagrams and use case diagrams.

We describe the defects that are analyzed in this paper in this section. Each description consists of a name, an abbreviation (which is used in the sequel), a brief description that informally describes the defect. A formal specification of the defect types based on a meta-model can be found in [21].

Message without Name (EnN)
In sequence diagrams objects exchange messages. The arrows representing the messages should be annotated with a name that describes the message. In case no name describes the message, this defect is present.

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 the name of the called method. In case there is no correspondence between the message name and a provided method name, this defect is present.

Message in the wrong direction(ED)
This inconsistency occurs if 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".

Class not instantiated in SD (CnSD)
The objects in sequence diagrams should be instantiations of classes. If there is no class instantiation in a sequence

diagram of a class that is defined in a class diagram of the model, this defect is present.

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.

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 if there is a use case that is not illustrated by any sequence diagram.

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

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

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

4. EXPERIMENT DESIGN

We conducted 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 according to Wohlin et al. [28]. Most parts are the same for both experiments. The differences are explained where applicable.

4.1 Design

The treatment in this experiment is the *defect injection* and the different levels are "no defect" and the eight defects defined in Section 3.

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.

4.2 Objects and Task

The subjects were given fragments of UML models. A fragment consists of two or three diagrams. For each fragment the subjects had to answer a multiple-choice question that asked how they would implement the system given the UML model fragments. Subject's were not asked to inspect the model according to a particular reading technique, but asked to answer the question from the perspective from a person who has to implement the system. Therefore they implicitly followed an ad hoc reading approach.

For each of the eight selected defect types that were presented in Section 3 we constructed a UML model fragment that contains an instance of the defect type. With each fragment we presented a question that focusses on a specific aspect of the model fragment. For each multiple-choice question there were four possible answer options, that represent four interpretations. The questions about model fragments containing an injected defect were paired with a similar control question that focusses on the same aspect but that does not contain a defect in the model fragment. The control questions served in the analysis of the results to compare the answer behavior in case of a defect to the ideal case without a defect.

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

For questions about a defected model: 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 incorrect with respect to all given diagrams. This is illustrated in the example in Section 4.2.1. 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 control questions: One correct answer option and the other three answers being incorrect.

All questions had a fifth answer option where the subjects could indicate that they detected a defect. The subjects were asked to give an explanatory motivation of their answer.

4.2.1 A Question about a defected Model

As an example we present question Q2 from the experiment. Q2 is a representative question for the whole questionnaire. The model fragment we presented consist of a class diagram and a sequence diagram (Figure 1). 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 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.seeFromMenu()}}
```

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();
```

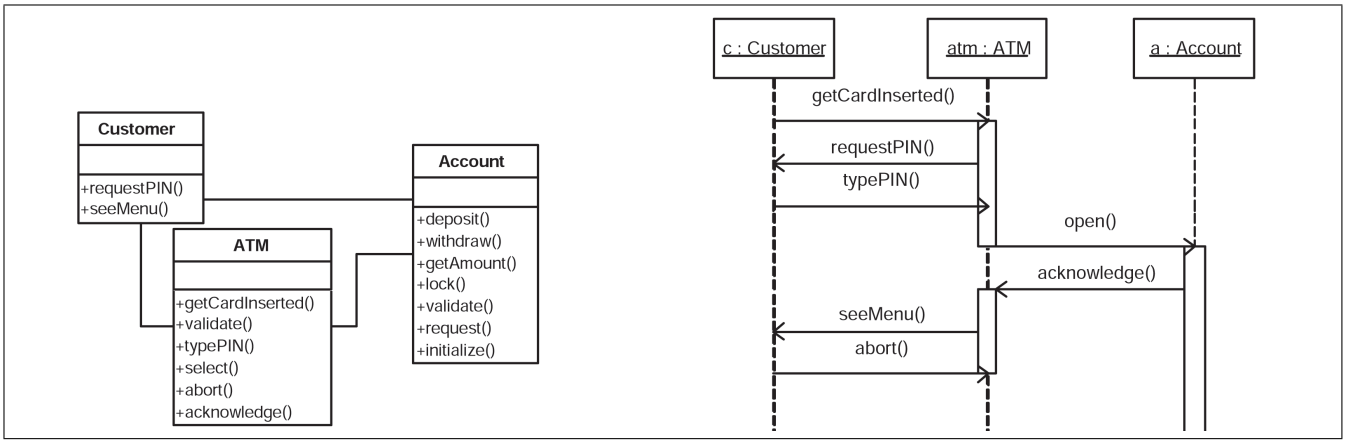

Figure 1: Example Class Diagram (left) and Sequence Diagram (right)

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

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

4.3 Subjects

4.3.1 Students

In total 111 students participated in the experiment. The experiment was conducted within the course "Software Architecting" 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. Most students have some experience in using the UML and object oriented programming through university courses and industrial internships.

4.3.2 Professionals

In total 48 professionals from 18 companies in 10 countries participated in the experiment by completing the online questionnaire [1]. Some of the subjects did not complete all questions, but all questions were answered by at least 27 subjects. 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. The most frequent job descriptions (of all subjects who entered a job description) were 'architect', 'designer' and 'engineer'. In a self assessment on a scale from 1 (no experience) to 5 (several years of experience) the professionals indicated the following: designing (average: 4.5), UML, (4.3), implementing (4.0), code review (3.8), inspections (3.7) and design review (3.4).

4.4 Preparation

Prior to the experiment we conducted a pilot run to evaluate the experimental design and the experiment materials. The subjects of the pilot experiment did not participate in the actual experiment.

UML was presented and explained to the students as part of the "Software Architecting" course. 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. Those who had only limited UML experience familiarized themselves with the UML during the assignment.

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

4.5 Operation

4.5.1 Student Experiment

The student experiment was conducted in two runs. The first run contained questions Q1 to Q10, the second run (five weeks later) contained questions R1 to R5. The procedure of operation was the same for both runs.

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 proposed by Carver et al. [6].

The experiment was conducted in a classroom with the subjects spread out in an exam-like setting. The subjects were given the experiment material containing instructions, the model fragments, questions and answers options. For the experiment the subjects had 90 minutes available. The average time for completing the first run was 67 minutes, hence there was no time pressure for the subjects. For the second run the time was not collected. 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 academic background, work experience, experience with UML and other relevant software engineering related experience.

4.5.2 Professionals Experiment

Because of professionals' time constraints they performed only one experiment run. 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 ex-

periment 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).

4.6 Variables

The factor of interest in this study is the defect type. This variable has nine levels: the eight defect types and 'no defect' for control questions. Additionally we controlled the variable 'domain knowledge' by designing some questions with symbolic names instead of meaningful names to make observations about the effect of domain knowledge. This is not related to our main research questions and is further discussed in Section 6.

The subject's answers to the multiple choice questions are summarized in a vector with five fields. The fields contain the frequencies of answers for the four answer options and the fifth option, that indicates an error. Based on this data we measure two dependent variables that relate to our research questions stated in Section 1.

In RQ1 we are interested in whether implementers detect a defect for a given model fragment. The corresponding dependent variable is the *detection rate* for each question. The detection rate of a question is the ratio 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.

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). When no defect is present, all subjects should ideally give the same implementation.

In RQ2 we are interested in whether undetected defects impact the interpretation of a model. An undetected defect is not necessarily problematic. In case the writer(s) of a model and all readers have the same interpretation 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 developed a so called agreement measure (short *AgrM*). AgrM maps the four fields of the answer vector representing the answer frequencies to a scalar. The agreement measure is 0 if answers are distributed equally over all options (maximal disagreement), and 1 if only one option receives all answers (everyone agrees). AgrM is informally explained in Figure 2 as the opposite of entropy. AgrM is described formally in the Appendix. We use AgrM for descriptive statistics and defect classification in Section 5.

4.7 Hypotheses

Ideally, when a defect is present, it is detected by everyone (detection rate = 1) and when no defect is present, no one reports a defect (detection rate = 0). We expect that

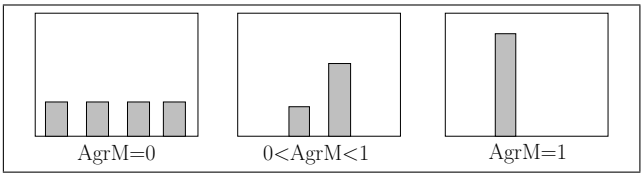

Figure 2: Informal explanation of AgrM measure

defects are not detected by everyone. This would lead to the null hypothesis: 'the detection rate for defected models = 1' and the alternative hypothesis 'the detection rate for defected models < 1'. However, in practice false negatives and false positives appear. False negatives reduce the maximum measured defect rate of 1. We assume that the likelihood of false positives equals the likelihood of false negatives. This justifies hypothesis **H1** for RQ1, which is tested for each defect type (d-rate$_{def}$ and d-rate$_{control}$ denote the detection rate for defected questions and for control questions, respectively):

- $H1_0$: d-rate$_{def}$ = 1−d-rate$_{control}$
- $H1_{alt}$: d-rate$_{def}$ < 1−d-rate$_{control}$

In RQ2 we address the risk for misinterpretation caused by a defect. For RQ2 we only consider the four fields of the answer vector that represent the answer frequencies for the four options that correspond to some choice of implementation. The field containing the frequency of indicated errors is discarded. Ideally, everyone interprets a given model in the same way (no misinterpretation). In this case 100% of the subjects give the same answer. The results of the control questions (Table 2) show that even without a defect not all subjects agree upon the same interpretation (i.e. due to human imperfection some subjects make errors). The results of the control questions show that it is reasonable to expect that 98% of the subjects agree on one interpretation, and 2% are spread over the other three options. This leads to the hypothesis **H2** for RQ2, which is tested for each defect type:

- $H2_0$: for a defected model fragment ≥ 98% of the implementers agree upon the same interpretation.
- $H2_{alt}$: for a defected model fragment < 98% of the implementers agree upon the same interpretation.

4.8 Analysis Techniques

To test H1 we consider for each defect type the number of subjects that indicated a defect and the number of subjects that did not indicate a defect and the same data for the corresponding control question. We applied McNemar's test [2, 22] for marginal homogeneity. This test is similar to the χ^2 test, but does not require independent samples. We have a same subject design, hence our samples are dependent.

To test H2 we consider the distribution (l, r) where l is the frequency of the most prominent interpretation and r combines the frequencies of the three other interpretations. More formally: l is the maximum frequency of the four interpretations options, and r is the sum of the frequencies of the three other interpretation options. The sum s of l and r is the total number of answers of subjects, not indicating an error. We construct a reference distribution $(0.98 \cdot s, 0.02 \cdot s)$ which represents the ideal case, where 98% of the readers

agree on one interpretation. Since these distributions are independent we can use the χ^2 test to test the two distributions for equality.

We used Microsoft Excel for hypothesis testing.

4.9 Threats to Validity

In this section we discuss the threats to validity in order of decreasing priority: internal, external, construct and conclusion validity (according to [28]).

4.9.1 Internal Validity

Threats to internal validity are influences on the causal relation between the controlled factors and the independent variable.

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 4.1 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.

Communication between the subjects influences the subjects' answer behavior. This threat can be ruled out for the student subjects, since the experiment was executed in an observed exam session where the students were not allowed to communicate. There is a risk, that the professionals communicated. After participating in the experiment we interviewed some of the subjects, who indicated that they did not communicate about the experiment during execution.

A difference between the students and the professionals is, that the professionals volunteered to participate in the experiment and may be more motivated for a task. We cannot completely exclude this threat, but since the results of both subject groups are very similar we assume that the volunteer effect has no significant influence on our observations.

Subject's knowledge of an application domain can influence the dependent variable. Since there are differences in the cultural and educational background of the subjects this effect is a threat to the internal validity of this experiment. To compensate for this effect we have chosen objects from various application domains and some objects unrelated to any application domain (because symbolic names are used).

4.9.2 External Validity

Threats to external validity are concerned with whether the results of the experiment can be generalized to a professional software context.

Students as subjects could be a threat for the external validity of the experiment. The subjects in the first experiment are all MSc students with experience in UML. According to Kitchenham et al. [16] students can be used as subjects. Additionally we have performed the second experiment with professionals. The experiment has fewer subjects, but confirms the results of the larger first experiment (Section 6.3).

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. When acting (e.g. reading, modification) on industrial models, designers focuss on only a subset of the model, which decreases the size gap for this experiment on cognitive effects. As the complexity in industrial models is in general larger, the effects of defects in industrial models are expected to be at least as severe as the effects reported here. 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.[9] have source code fragments of 18 classes and Purchase et al.[26] 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.

4.9.3 Construct Validity

Construct validity is the degree to which the dependent and independent variables accurately capture the concepts that should be measured by this study.

Since the experiment is designed as a multiple-choice test, four possible 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. As the set of possible interpretations is in practice much larger and the subject is not guided and will most likely not *guess* the interpretation, we expect the detection rate and agreement measure to be even lower than in the experiment.

4.9.4 Conclusion Validity

Conclusion validity is the extent to which correct conclusions about the relations between the treatment and the outcome of the experiment can be drawn. We have carefully taken precautions to avoid issues threatening the conclusion validity. Particularly precautions guaranteeing homogeneity of the subject groups with respect to subjects' background, satisfaction of the assumptions of the statistical tests and accuracy of measurements are taken (described in in Section 4.

5. RESULTS

The results of this experiment are shown in Table 2. The first column shows the identifier of the question, the second column shows the defect type (as in Section 3). In the following columns, S and P indicate the results from the students experiment and the professionals experiment, respectively. The column N shows the number of subjects participating in the question, The following columns give the results for detection rate ($d\text{-}rate$) and agreement measure ($AgrM$). The

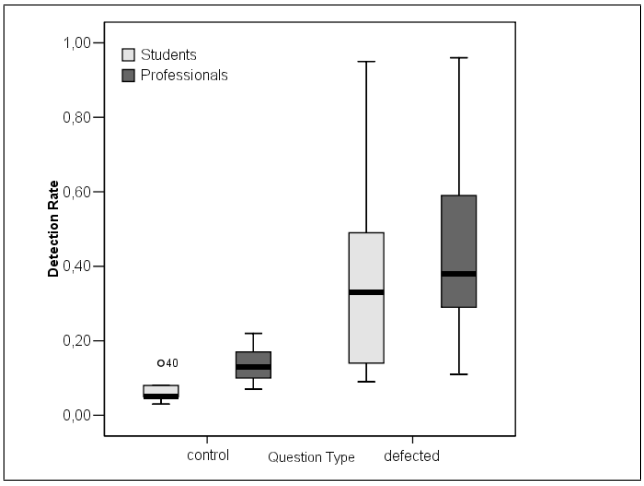

Figure 3: Defect detection boxplot

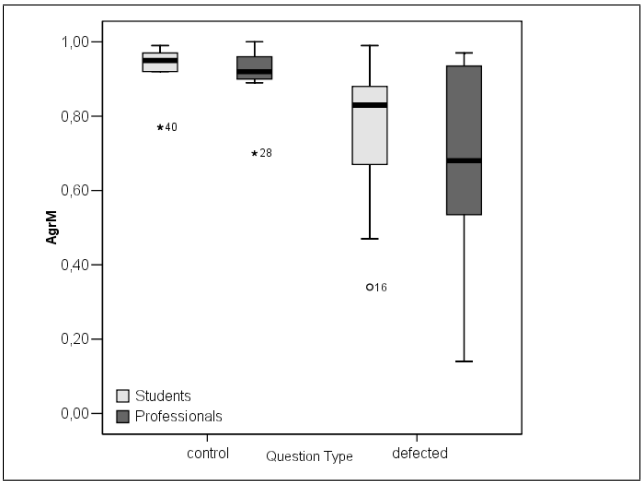

Figure 4: Agreement boxplot

column *s* indicates with a checkmark which questions used symbolic names in the model fragments. All questions (except Q10) are paired with a control question that does not contain a defect (the paired control question is given in the column *Ctrl. Qu.*). The results are discussed in this section.

In the Tables 3 and 4 we give an objective classification of defect types based on the experiments.

5.1 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 answer options of the questions contained wrong answers (one wrong answer for defected questions, three wrong answers for control questions). We use the number of wrong answers per subject to identify subjects who are *guessing*. The first run of the student experiment and the professionals' experiment contained 18 questions (including subquestions). 81,9% of the students gave no or one wrong answer. No student gave more than three wrong answers. The results for the professionals experiment are similar. Therefore no subject was removed. The second run of the student experiment contained 9 questions, two students gave three wrong answers (which we regarded as suspect of guessing) and they were therefore removed from the results.

5.2 RQ1: Defect Detection

Research question RQ1 investigates whether implementers detect a defect in an UML model.

The data of all defect types from Table 2 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 subjects indicate to give a implementation than detect the error. Defect types marked with an *s* are based on model fragments with symbolic names.

We tested hypothesis H1 to investigate RQ1. The test statistic χ^2 of McNemar's test is given in Table 2. The threshold value at a significance level of $\alpha = 0.05$ and one degree of freedom is 3.84. We indicate rejection of the null hypothesis with *r*, failure of rejection is indicated with an *f*. The null hypothesis is rejected for all defect types except for 'Class not instantiated in SD'.

Table 3 presents a classification of defect types by detection rate (of the student experiment), defect types at the top of the table remain undetected in most cases. The least detected defects are *Multiple classes with the same definition*, *Method not in SD (symbolic)* and *Object has no class in CD*. The most detected defect is *Class not in SD (symbolic)*.

5.3 RQ2: Variation of Interpretations

With research question RQ2 we investigate the risk for misinterpretation caused by a defect. The data from Table 2 is summarized in Figure 4. The boxplot shows the agreement measure of questions containing a defect compared to control questions. The control questions have a high AgrM value (average: 0.91), which indicates that most subjects have the same interpretation of the model. The AgrM value of defected models is widely spread between .14 and 1.00 (average: 0.73). The results show that defected models have a larger variety of misinterpretations and, hence, contain a higher risk for misinterpretation and miscommunication.

We tested hypothesis H2 to investigate RQ2. The test statistic χ^2 of the χ^2 test is given in Table 2. The threshold value at a significance level of $\alpha = 0.05$ and one degree of freedom is 3.84. We indicate rejection of the null hypothesis with *r*, failure of rejection is indicated with an *f*. The null hypothesis is rejected for all defect types in the student experiment. In the professionals experiment we fail to reject the null hypothesis for four defect types. These defect types are inconsistencies between sequence diagrams and class diagrams. We found that the practitioners tend to regard the sequence diagram as leading. Amongst students this behavior was observed to a much lesser degree.

H2 compares the results of a question with a reference distribution (as described in Section 4.8). This allows us to test the hypothesis for defected questions as well as control questions. As expected, we failed to reject the null hypothesis for all control questions.

Qu.	Defect	N S	N P	d-rate S	d-rate P	AgrM S	AgrM P	s	Ctrl. Qu.	χ^2 (H1) S	χ^2 (H1) P	χ^2 (H2) S	χ^2 (H2) P
Q1.1	Msg. without Name	111	48	.69	.60	.47	.44	√	Q1.2	16,00 r	8,17 r	87,37 r	19,16 r
Q2	Msg. without Method	111	40	.39	.38	.84	.90		Q4	50,97 r	13,37 r	11,68 r	1,78 f
Q8	Msg. without Method	111	30	.49	.33	.86	.94	√	Q4	45,63 r	13,76 r	7,17 r	1,01 f
R1.2	Msg. without Method	110	♭	.49		.69		√	R1.1	42,12		51,56 r	
Q3	Msg. in wrong direction	111	34	.60	.58	.47	.95	√	Q1.2	73,96 r	4,26 r	25,11 r	0,42 f
Q6.1	Obj. has no Class in CD	111	31	.14	.21	.81	.97		Q6.2	73,09 r	21,16 r	13,09 r	0,20 f
Q7.1	Use Case without CD	109	30	.50	.52	.83	.44		Q7.2	47,08 r	6,25 r	9,17 r	10,62 r
Q5.2	Class not inst. in SD	111	34	.47	.68	.49	.64		Q5.1	84,05 r	4,76 r	75,40 r	6,52 r
Q9.2	Class not inst. in SD	109	30	.95	.96	.54	.14	√	Q9.1	1,00 f	0,67 f	85,17 r	7,26 r
Q10	Multiple Class defs.	107	27	.10	.33	.92	.68		♮			5,27 r	9,70 r
R3	Method not called in SD	109	♭	.14		.67		√	Q1.2	76,05 r		55,70 r	
Q1.2	Control question	110	48	.08	.16	.97	.89	√				0,18 f	1,50 f
Q4	Control question	111	34	.05	.13	.95	.91					1,10 f	1,66 f
R1.1	Control question	110	♭	.06		.92		√				3,42 f	
Q6.2	Control question	111	31	.05	.07	.96	.97					0,13 f	0,15 f
Q7.2	Control question	110	30	.05	.22	.95	.92					2,69 f	0,94 f
Q5.1	Control question	111	34	.05	.13	.94	.95					1,85 f	0,88 f
Q9.1	Control question	110	30	.04	.07	.99	1.0	√				0,40 f	0,50 f

Table 2: Complete Results. (♭: not part of professionals experiment; ♮: no control question)

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	.14	.21	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 3: Classification by detection rate

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	.81	.97	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 4: Classification by AgrM

Table 4 presents a classification of defect types by AgrM (of the student experiment), defect types with the largest spread over different interpretations are at the top. Most misinterpretation is caused by the defect type *Class not in SD (symbolic version)*.

6. ADDITIONAL OBSERVATIONS

6.1 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, which enables to infer the correct meaning.

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`, `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 2). 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 5). 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. Subjects who compensate the defect using domain knowledge 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.

6.2 Prevailing Diagram

The questions about consistency defect between a sequence diagram and a class diagram were designed such 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

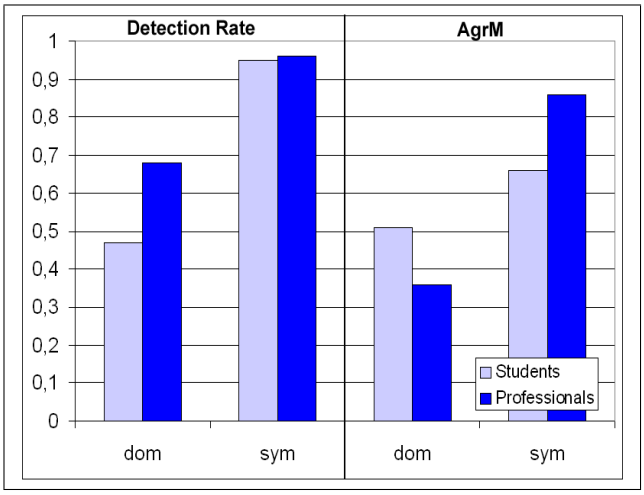

Figure 5: Results for Defect CnSD.

types that allow either way of interpretation, the option regarding the sequence diagram as correct (i.e. *prevailing*) 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 6.1. 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.

6.3 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 statistical reliability of the results obtained from this group. The drawback of experiments with students is that the results may not be generalizable to professionals because of the different experience level. Students are less experienced than professionals and their pressure and motivation differs from those of professionals.

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 students and professionals are related. The correlations are 0.929 (P-value<0.001) for detection rate and 0.779 (P-value<0.001) for AgrM. Hence, there is a strong correlation between detection rate and AgrM of students and practitioners. The results are statistically significant. Hence, the reliable results of the student group can be generalized to professionals without loss of validity.

7. CONCLUSIONS AND FUTURE WORK

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 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. The risks for misinterpretations are similarly alarming. Some defect types cause a large variation in interpretations amongst readers (e.g. *Class not in Sequence Diagram* has an AgrM of 0.14) and other defect types hardly cause any misinterpretations (e.g. *Message without Method* has a AgrM of 0.94).

We presented a classification of defect types based on detection rate and risk for misinterpretations. In contrast to most defect classifications found in literature and industrial practice this classification is objective and based on empirical evidence. 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. These results are generalizable to professional UML designers.

We observed that the presence of domain knowledge effects 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.

Our findings can be used to improve the practice of software modelling in the following ways: Defect prevention can be improved through the use of guidelines for creating UML models that minimize the risk of misinterpretation. Using the classification, defect removal activities can be improved by focussing on the most risky defects first.

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. We invite other researchers to replicate this experiment using other groups of subjects.

8. REFERENCES

[1] EmpAnADa UML Experiment Web Questionnaire. 2004. `http://www.win.tue.nl/empanada/survey`.

[2] A. Agresti and B. Finlay. *Statistical methods for the social sciences*. Prentice Hall, 3rd edition, 1997.

[3] V. Basili. Evolving and packaging reading technologies. *Journal of Systems and Software*, 38(1), 1997.

[4] V. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sorumgard, and M. Zelkowitz. The empirical investigation of perspective-based reading. *Empirical Software Engineering*, 1(2):133–144, 1996.

[5] V. R. Basili, G. Caldiera, and H. D. Rombach. The goal question metric paradigm. In *Encyclopedia of*

Software Engineering, volume 2, pages 528–532. John Wiley and Sons, Inc., 1994.

[6] J. Carver, L. Jaccheri, S. Morasca, and F. 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.

[7] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong. Orthogonal defect classification - a concept for in-process measurements. *IEEE Transactions on Software Engineering*, 18(11):943–956, November 1992.

[8] R. Conradi, P. Mohagheghi, T. Arif, L. C. Hedge, G. A. Bunde, and A. Pedersen. Object-oriented reading techniques for inspection of UML models – an industrial experiment. In *Proceedings of the European Conference on Object-Oriented Programming ECOOP'03*, volume 2749 of *LNCS*, pages 483–501. Springer, July 2003.

[9] I. Deligiannis, I. Stamels, L. Angelis, M. Roumeliotis, and M. Shepperd. A controlled experiment investigation of an object-oriented design heuristic for maintainability. *Journal of Systems and Software*, 2(72):129–143, 2004.

[10] M. Elaasar and L. Briand. An overview of UML consistency management. Technical Report SCE-04-18, Carleton University, Department of Systems and Computer Engineering, 2004.

[11] K. E. Emam and I. Wieczorek. The repeatability of code defect classifications. In *Proceedings of the 9th International Symposium on Software Reliability Engineering*, pages 322–333, 1998.

[12] M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211, 1976.

[13] M. E. Fagan. Advances in software inspections. *IEEE Tr. on Software Engineering*, 12(7):744–751, 1986.

[14] T. Gilb and D. Graham. *Software Inspection*. Addison Wesley Publishing Co., 1993.

[15] D. Kelly and T. Shepard. A case study in the use of defect classification in inspections. In *Proceedings of the IBM Centre of Advanced Studies Conference 2001*, pages 26–39. IBM, 2001.

[16] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions of Software Engineering*, 28(8):721–734, August 2002.

[17] L. Kuzniarz, Z. Huzar, G. Reggio, J.-L. Sourrouille, and M. Staron. *2nd Workshop on Consistency Problems in UML-based Software Development at the UML2003*. Blekinge Institute of Technology, 2003.

[18] O. Laitenberger, C. Atkinson, M. Schlich, and K. E. Emam. An experimental comparison of reading techniques for defect detection in UML design documents. Technical Report NRC/ERB-1069, National Research Council Canada (NRC), Ottawa, Canada, December 1999.

[19] O. Laitenberger and J.-M. DeBaud. An encompassing life-cycle centric survey of software inspection. *Journal of Systems and Software*, 2000.

[20] C. F. J. Lange and M. 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.

[21] C. F. J. Lange and M. R. V. Chaudron. Experimentally investigating effects of defects in UML models. CS-Report 05-07, Technische Universiteit Eindhoven, 2005.

[22] Q. McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12:153–157, 1947.

[23] J. Muskens, R. J. Bril, and M. R. V. Chaudron. Generalizing consistency checking between software views. In *Proceedings of the 5th IEEE/IFIP Working Conference on Software Architecture (WICSA)*, November 2005.

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

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

[26] H. C. Purchase, L. Colpoys, M. McGill, D. Carrington, and C. Britton. UML class diagram syntax: an empirical study of comprehension. In *Australian symposium on Information visualisation*, volume 9, pages 113–120, September 2001.

[27] C. Wohlin and A. Aurum. An evaluation of checklist-based reading for entity-relationship diagrams. In *Proceedings of the Ninth International Software Metrics Symposium*. IEEE CS, 2003.

[28] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlesson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2000.

APPENDIX

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.

First we define some abbreviations that we need for the explanation of the measure: K is the number of alternatives for a question; f_i is the number of times alternative i was selected, where $0 \leq i < K$ and $k_{[0..K)}$ is a sorted array such that $k_0 = (\max f_i : 0 \leq i < K)$ and $k_{K-1} = (\min f_i : 0 \leq i < K)$; N is the sum of answers over all alternatives: $N = \sum_{0 \leq i < K} k_i$.

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

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

AgrM is described in more detail in [21]