Investigating the Role of Use Cases in the Construction of Class Diagrams

Bente Anda and Dag I.K. Sjøberg
Simula Research Laboratory
P.O. Box 134
NO-1325 Lysaker
NORWAY
Tel.: +47 67828306
{bentea,dagsj@simula.no}

Abstract

Several approaches have been proposed for the transition from functional requirements to object-oriented design. In a use case-driven development process, the use cases are important input for the identification of classes and their methods. There is, however, no established, empirically validated technique for the transition from use cases to class diagrams. One recommended technique is to *derive* classes by analyzing the use cases. It has, nevertheless, been reported that this technique leads to problems, such as the developers missing requirements and mistaking requirements for design. An alternative technique is to identify classes from a textual requirements specification and subsequently apply the use case model to *validate* the resulting class diagram. This paper describes two controlled experiments conducted to investigate these two approaches to applying use case models in an object-oriented design process. The first experiment was conducted with 53 students as subjects. Half of the subjects used a professional modelling tool; the other half used pen and paper. The second experiment was conducted with 22 professional software developers as subjects, all of whom used one of several modelling tools. The first experiment showed that applying use cases to validate class diagrams constructed from textual requirements led to more complete class diagrams than did the derivation of classes from a use case model. In the second experiment, however, we found no such difference between the two techniques. In both experiments, deriving class diagrams from the use cases led to a better structure of the class diagrams. The results of the experiments therefore show that the technique chosen for the transition from use cases to class diagrams affects the quality of the class diagrams, but also that the effects of the techniques depend on the categories of developer applying it and on the tool with which the technique is applied.

Keywords: UML, Use cases, Object-oriented design, Modelling tool, Controlled experiment, Replicated experiment

1. Introduction

The functional requirements of a software system can be captured and documented in use cases, which determine the functional scope of the objects in the system. A use case driven process in which the use case model is a primary artefact in the identification of system classes, is frequently recommended together with UML [3,6,7,15,16,18,19,25]. Nevertheless, there is no established technique for the transition from use cases to class diagrams. One technique is to derive class diagrams directly from the use cases [7,15]. We call this *the derivation technique*. It is claimed that such a development process may lead to missing classes because the use case model is insufficient for deriving all necessary classes, and to the developers mistaking requirements for design [11,31]. Alternatively, an initial class diagram can be created independently of the use case model, for example using grammatical analysis of textual requirements documents as a mechanism for identifying classes [30]. Subsequently, the use case model can be applied as a means of validating and improving the class diagram [22,24,28]. We call this technique *the validation technique.*

Even though use cases are important in the creation of a class diagram, there is, to our knowledge, little empirical research on how to extract class diagrams effectively from use case models. Our goal was therefore to investigate empirically advantages and disadvantages of the two alternative ways of applying a use case model in an object-oriented design process.

We previously conducted a pilot experiment with 26 students to compare the derivation technique with the validation technique [32]. Based on the results, we formulated hypotheses to test whether the two techniques result in differences in the quality of the resulting class diagrams with respect to their *completeness* and *structure,* and the *time* required for constructing them.

The hypotheses were tested in an experiment with 53 students (called Experiment 1 in this paper) and another experiment with 22 professional software developers (called Experiment 2). The task was to construct a class diagram for a library system.

In the pilot experiment, the subjects used only pen and paper. To increase the realism of the context [13,28], we introduced the use of professional modelling tools in Experiments 1 and 2. In Experiment 1 half of the subjects used a modelling tool, in Experiment 2 all the subjects used such tools. Another way of making experiments more similar to an industrial context is to use professionals instead of students as subjects. Hence, we replicated the student experiment with an experiment with professionals as subjects.

The results from Experiment 1 showed that the validation technique led to more complete class diagrams in that they implemented more of the requirements than did the derivation technique. In Experiment 2 we found no difference between the two techniques with respect to completeness. However, the derivation technique resulted in class diagrams with slightly better

structure in both experiments, although the difference was not significant in Experiment 2. There was no difference with respect to time for the two techniques.

The remainder of this paper is organized as follows. Section 2 gives an overview of different techniques for the transition from requirements to design models and describes the two techniques evaluated in this experiment in detail. Section 3 presents the experimental design. Sections 4 and 5 report the detailed design and results of Experiments 1 and 2, respectively. Section 6 discusses the results of both experiments. Section 7 discusses some threats to the validity of the results. Section 8 presents some experiences from organising the experiments. Section 9 concludes and suggests further work.

2. Transition from Use Case Model to an Object-Oriented Design

There are several approaches to identifying classes from requirements specifications [22]:

1. In grammatical analysis the requirements specification is searched for nouns, which are candidate classes or attributes of classes, and services to be delivered by the system, which are candidates for methods. If the emphasis is on searching the requirements specification for nouns, the approach can be characterized as data-driven [27]; if the emphasis is on services, however, the approach can be classified as responsibility-driven [34].
2. The approach termed “common class pattern” is based on the identification of various kinds of classes, of which a system will typically consist [4]. Examples are physical classes, business classes, logical classes, application classes, computer classes and behavioural classes.
3. CRC cards are specially prepared for use in brainstorming sessions. Each developer plays one or more cards. New classes are identified from the message passing between the players. This is a responsibility-driven approach [34] and requires developers/participants who know the system requirements well.
4. In the use case-driven approach, the use cases are important in the identification of classes. A use case specifies the sequences of actions that the use case should be able to perform, that is, changes of state and communications with the environment. This implies that a use case contains a state and behaviour from which classes, attributes and methods can be derived, or against which class diagrams can be validated. Sequence and/or collaboration diagrams, which detail the requirements, are made for each use case scenario. The objects used in these diagrams contribute to the discovery of analysis classes. The analysis class diagram is then elaborated upon to produce a design model, from which code can be developed. A variant of the use case-driven approach, in which the classes are identified from the goals of each use case instead of from the scenarios, is suggested in [20].

Our goal was to investigate the two different techniques of approach (4), in which use cases are applied in the creation of class diagrams; namely (i) the derivation technique, in which class diagrams are directly derived from the use cases, and (ii) the validation technique, in which the use cases are applied as a means for validation. The independent variable in the experiments described in this paper is the technique for applying use cases in the design of class diagrams. The two techniques were refined in order to compare them in a controlled setting. Figure 1 and Table 1 describe the steps of the derivation technique, while Figure 2 and Table 2 describe the steps of the validation technique. There are two main differences between the two techniques:

- *The construction of an initial class diagram.* In the derivation technique, a domain model is constructed from the use cases, and this domain model is the basis for a complete class diagram. In the validation technique, an initial class diagram is constructed from a textual requirements specification.
- *The identification of methods*. In the derivation technique, sequence diagrams are used to identify the message passing between the classes; whereas in the validation technique, methods are identified from grammatical analysis of the requirements specification, and the method composition is subsequently validated using sequence diagrams.

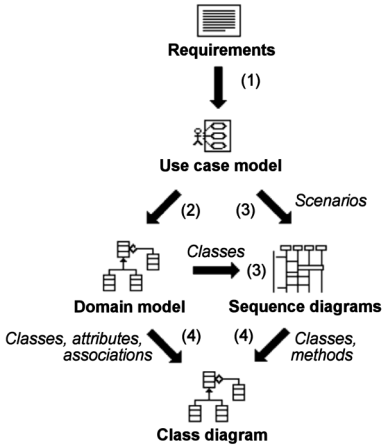

Fig. 1. The derivation technique

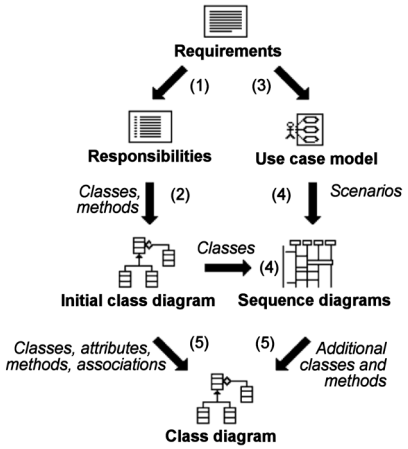

Fig. 2. The validation technique

Table 1. Steps of the derivation technique

1. Identify the system use cases.
2. Construct a domain model, with classes, attributes and associations, from the use case model. Note that no methods are added at this stage.
3. Construct sequence diagrams for each use case. Use classes from the domain model and add new classes where necessary. Identify the methods necessary for the realization of the use cases.
4. Extend the domain model to a complete class diagram by adding the classes and methods from the sequence diagrams.

Table 2. Steps of the validation technique

1. Identify the classes in the system using grammatical analysis of the textual requirements. Make a list of the responsibilities of each class.
2. Construct an initial class diagram with classes, attributes, associations and methods.
3. Identify the system use cases.
4. Construct a sequence diagram for each use case. Use classes from the initial class diagram. Identify the methods necessary for the realization of the use cases.
5. The initial class diagram is validated, and possibly updated, using sequence diagrams. The result is the final class diagram.

3. Design of Experiments

This section describes the experimental design of the experiments. Figure 3 shows the independent variable, the dependent variables and the context variables. The independent variable was described in the previous section. Here we describe the dependent variables and how they are evaluated, the context variables, the hypotheses tested and the tools used in the experiments. This section also gives an overview of the progress through the pilot experiment and the two experiments.

3.1. Dependent Variables

The goal of the experiments described in this paper is to give recommendations with respect to which technique is most likely to support the efficient construction of a good object-oriented design in a specific context. A good design is complete and well-structured, that is, easy to understand [5]. The evaluation scheme thus includes three scores:

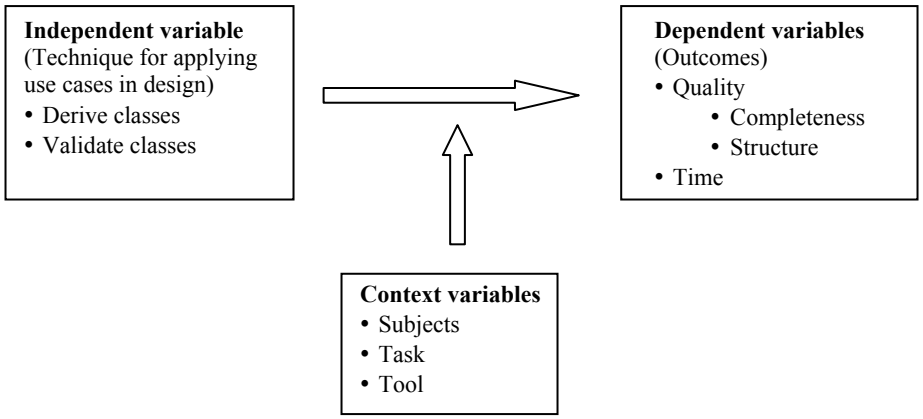

Figure 3. Experimental design

1. *Completeness*, measured in terms of how much of the functionality described in the requirements specification was actually implemented in the class diagram. The following aspects should be satisfied:
    1. All services described in the requirements specification are implemented.
    2. The services are allocated to all and only the correct classes; that is, the class diagram contains all required correct classes and no superfluous classes.
    3. The classes contain the necessary information in terms of attributes and associations.

    Each of these three aspects were given a score between 0 (very poor) and 5 (very good), and the three scores were combined into one according to the following formula

    ```
    (Aspect 1 * 2 + Aspect 2 + Aspect 3)/4.
    ```

    The formula reflects that we consider the first aspect to be the most important.

2. *Structure*, measured in terms of cohesion and coupling. Cohesion and coupling were measured subjectively because the class diagrams were too small to apply established metrics, such as the high-level design metrics described in [8,9]. Each class diagram was given a score between 0 (very poor) and 5 (very good) on structure.
3. *Time* spent on obtaining class diagrams of acceptable quality, measured in minutes. The time was compared only for those subjects who produced satisfactory solutions, that is, those who obtained a score of at least 3 on both completeness and structure.

Three people were involved in determining what would be a correct solution to the experimental task. One of them was the first author of this paper, and one was an independent consultant from another research institute. The consultant marked the class diagrams based on that solution. He was not involved in the design or conduct of the experiments, and did not know the research questions. He was not involved in the teaching of the students in Experiment 1, and he did not know any of the professionals who participated in Experiment 2. He could see which class diagrams in Experiment 1 were made with pen and paper and which were made using a modelling tool, but he could not distinguish between class diagrams made with different tools because he received the class diagrams as pdf-documents. He knew that the subjects of Experiment 1 were students, and that the subjects of Experiment 2 were professionals. Approximately 20% of the class diagrams were also scored by the first author, and it was verified that these scores were consistent with those made by the consultant.

An example of how one class diagram was marked is shown in Appendix B. The X's mark the elements that were correct in that specific class diagram.

3.2. Context Variables

The effects of a specific technique will depend on the context in which it is used. The important context variables in these experiments are subjects, task and tool.

3.2.1. Subjects

In our opinion, students and junior professionals, with only basic knowledge of object-oriented modelling with UML, are more in need of support from a technique than are professionals with much experience from object-oriented modelling. These groups of subjects consequently represent our target population. The subjects in Experiment 1 were students; the subjects in Experiment 2 were professional consultants. The background of the subjects is described in detail in Sections 4 and 5.

3.2.2. Task

The task of these experiments was to construct a class diagram for a simple library system that contains functionality for borrowing and returning books and videos. This task was chosen because the application domain is simple and well-known. This system is described in many books on UML, for example [24,31]. The task was clearly smaller than most software development projects, but since larger development projects typically are broken down into smaller parts, and individual parts are often modelled by different teams [1], the task of the experiment may be representative for small tasks in industrial development projects. The class diagrams created by the subjects of the experiments contained between four and thirteen classes.

The subjects received a textual requirements document and a use case model with the following use cases:

- borrow an item
- hand in an item
- check the status of an item

The use cases were described using a template format based on those given in [10]. The subjects were given detailed guidelines on the two techniques to apply. The guidelines are shown in Appendix A.

3.2.3. Tool

UML class diagrams are most often made using a modelling tool, although UML can be used with simple drawing tools or even with pen and paper. We do not believe that the tools used in this experiment differ much with respect to the approaches evaluated in the reported experiments, but some influence is likely. For example, modelling tools that assume that the classes are shared by all the diagrams in the model may to some extent contradict the validation technique. Such tools facilitate the use of already defined classes in the sequence diagrams, and do not encourage the development of sequence diagrams independently of the existing class diagram. A consequence of the classes being shared by all the diagrams in the model is that classes are updated when the sequence diagrams are developed. An explicit activity on updating the class diagrams, as prescribed in the validation process, may thus appear superfluous.

In Experiment 1 half of the subjects used the modelling tool Tau UML Suite from Telelogic [33], and the other half used pen and paper. Consequently, we blocked on tool in the analysis of that experiment. In Experiment 2 all the subjects used one out of the three different modelling tools Visio [36], Rational Rose [14] and MagicDraw [35]. These tools are described in more detail in Sections 4 and 5.

3.3. Hypotheses

The results from the pilot experiment [32] showed a difference, both in the quality of the resulting class diagrams and in the time spent on design for the two techniques. To compare the two techniques, we tested the following hypotheses in Experiments 1 and 2:

H1$_0$: There is no difference in the completeness of the class diagrams.
H2$_0$: There is no difference in the structure of the class diagrams.
H3$_0$: There is no difference in the time spent constructing the class diagrams.

3.4. Supporting Tool

All the subjects used a Web-based tool for experiment support, the Simula Experiment Support Environment (SESE) [2], which has the following functionality:

- It distributes experimental material.
- It uploads documents produced by the subjects. In our two experiments the UML documents created using the various modelling tools were uploaded to SESE when they were completed.
- It records the time on each task of an experiment for each subject.

SESE also includes a feedback collection tool, which was used in Experiment 1 [17]. This tool was used to check whether the subjects actually followed the guidelines describing the techniques. This tool contains a screen that pops up at pre-specified intervals. The subjects used the screen to comment on what they were doing every 15 minutes during the experiment. Examples of comments for one subject are given in Appendix C.

3.5. The Series of Experiments

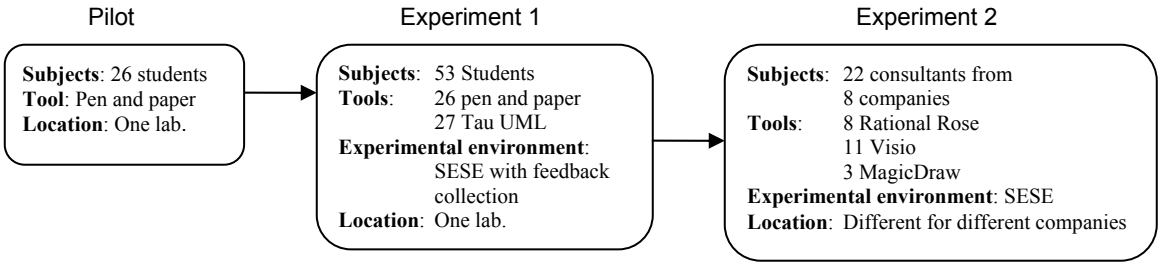

Figure 4. Progress through the experiments

Figure 4 shows the progress through the pilot experiment and the two subsequent experiments. The differences between the pilot experiment and Experiment 1 were that the experimental material and the evaluation scheme were modified slightly, realism was increased by introducing a modelling tool and feedback collection was used to ensure internal validity.

Experiment 2 is a *differentiated* replication of Experiment 1 with professional consultants as subjects using their usual modelling tool. A differentiated replication involves variations in essential aspects of the experimental conditions [21]. In Experiment 2 we decided not to use the feedback collection tool. A maximum of seven consultants were present at one time, and we therefore expected to be able without the tool to check that the guidelines were followed.

4. Experiment 1

The first experiment was run with students. In addition to comparing the derivation technique with the validation technique, we compared the use of pen and paper with the use of a commercially available modelling tool regarding the two techniques. This section describes the subjects, setting, assignment, analysis model and the results of Experiment 1. The last subsection discusses the explanatory power of the analysis model.

4.1. Subjects and Setting

The subjects were 53 students taking an undergraduate course in software engineering. The students were in their 3rd or 4th year of study. They had learned the basics of object-oriented programming and UML through this and previous courses in computer science. They had also used the Tau UML Suite in this and one previous course. Table 3 shows the background of the students regarding total study credits (20 credits equal one year of study), credits in courses that included UML and years of work experience. Four of the students had used UML in industry for 1-2 months.

Table 3. Background of subjects in Experiment 1

	Min	Average	Max
Credits	32	63	135
UML Credits	0 (6 subjects)	7	15
Work experience (years)	0 (41 subjects)	2	13

The students were present in the same laboratory during the experiment. They worked until they were finished, the time taken varying from 2.5 to 4.5 hours. In addition to the authors of this paper, two more persons were present during the experiment to help the subjects with both understanding the experiment and the tools.

The experiment was voluntary, but the students were informed that it was relevant for their course. They were also paid for their participation.

4.2. Assignment

A randomized block experimental design was used; each subject was assigned to one of two techniques by means of randomization and blocking. The two techniques were *derivation* and *validation*. The blocks were *pen and paper* and *Tau UML Suite*. Table 4 shows the distribution of the categories of subject in the different groups. The design is uneven because five of the students who had registered did not present themselves for the experiment.

Table 4. Distribution of subjects in Experiment 1

Technique\Tool	Pen and paper	Tau	Total
Derivation	14	15	29
Validation	12	12	24
Total	26	27	53

Six of the subjects had major problems during the experiment; they either had misunderstood the experiment or possessed too little knowledge of UML to perform the required tasks. This was revealed based on their comments in the feedback collection tool. Those six subjects were removed from the analysis. Three of them had been assigned to the derivation technique; the other three to the validation technique.

4.3. Analysis Model

To test the hypotheses, a regression-based approach was used on the unbalanced experiment design. The dependent variables were **completeness**, **structure** and **time,** as defined in Section 3.1. The independent variable was **technique** (the main treatments of this experiment were the two techniques). The blocking factor was **tool**. The model is summarised in Table 5.

Table 5. Specification of analysis model

Model	Response	Model Term	Primary use of model term
(1)	Completeness	Technique	**Test $H1_0$**
		Tool	Assess the effect of tool on completeness
		Technique* Tool	Assess the interaction of the technique and effect of tool on completeness
(2)	Structure	Technique	**Test $H2_0$**
		Tool	Assess the effect of tool on structure
		Technique* Tool	Assess the interaction of the technique and effect of tool on structure
(3)	Time	Technique	**Test $H3_0$ for subjects with score 3 or higher on both completeness and structure**
		Tool	Assess the effect of tool on time
		Technique* Tool	Assess the interaction of the technique and effect of tool on time

4.4. Descriptive Statistics and Tests of Hypotheses

The hypotheses were tested using an unbalanced *analysis of variance* (ANOVA) by means of the GLM (General Linear Model) procedure of the SAS statistical analysis system [26]. This section describes the results of the experiment in terms of descriptive statistics and tests of

hypotheses, using the GLM model in Table 5 for the three scores completeness, structure and time, respectively. A significance level of 5% (α=0.05) is chosen as the level of significance. Type III sums of squares are used in the GLM model. They are preferred in testing effects in unbalanced cases because they test a function of the underlying parameters that is independent of the number of observations per treatment combination, see discussion in [12].

To produce exactly valid p-values, GLM assumes that the dependent variable(s) are continuous and that the true residual errors are independent and identically normally distributed. In our model, time is obviously continuous. Completeness and structure are measured on a scale from 0 to 5, and we consider a step from (say) 0 to 1 on the measurement scale to indicate the same difference as a step from (say) 4 to 5. We will, in this context, treat completeness and structure as continuous variables. Residual analyses of the model indicate that the assumption of normality is not violated.

4.4.1. Completeness

Table 6 shows higher median and mean scores for the validation technique than for the derivation technique for those who used pen and paper. The mean for the validation technique is only slightly higher for those who used Tau UML Suite.

The Technique*Tool interaction is not significant, although it is very close to (F=3.77, p=0.0586). (As described above, most of the difference between the two techniques occurred for those who used pen and paper.) Therefore, the tests for the individual effects may be considered valid, and show a significant Technique effect (F=9.26, p=0.0040) but no significant Tool effect (F=0.60, p=0.4414). Hence, the validation technique gave a significantly higher score on completeness than did the derivation technique, so hypothesis $H1_0$ is rejected.

4.4.2. Structure

For structure, the results point in the opposite direction. The derivation technique gave a higher median and mean for both those who used pen and paper, and for those who used Tau UML Suite. Table 7 shows a low Technique*Tool interaction (F=3.60, p=0.4415). Hence, the individual effects are valid. The difference between the techniques is significant (F=4.24, p=0.0455), so hypothesis $H2_0$ is also rejected.

4.4.3. Time

The time spent on performing the tasks was compared only for those subjects who produced satisfactory solutions, that is, those who obtained a score of at least 3 on both completeness and structure. Sixteen of the subjects who applied the derivation technique, and nine of those who applied the validation technique, obtained such a score.

Both in the pen-and-paper group and in the tool group, the descriptive statistics for time show very little difference between the two techniques. There is no Technique*Tool interaction. The

test of difference between the two techniques gave F=0.28 and p=0.6037, so hypothesis $H3_0$ is not rejected.

However, there is a relatively large difference between those who used pen and paper (median 172 minutes) and those who used Tau UML Suite (median 224 minutes). The difference is significant (F=6.77, p=0.0167).

Table 6. Descriptive statistics for Experiment 1

Score	Block	Technique	N	Min	Q1	Median	Mean	Q3	Max
Completeness	Pen and paper	Derivation	12	1.0	1.0	2.5	2.5	3.5	5.0
		Validation	9	3.0	3.0	5.0	4.3	5.0	5.0
		Total	21	1.0	2.0	3.0	3.3	5.0	5.0
	Tau UML Suite	Derivation	14	0.0	3.0	3.0	2.9	3.0	5.0
		Validation	12	2.0	2.5	3.0	3.3	4.5	5.0
		Total	26	0.0	3.0	3.0	3.1	4.0	5.0
	Total	Derivation	26	0.0	2.0	3.0	2.7	3.0	5.0
		Validation	21	2.0	3.0	3.0	3.8	5.0	5.0
Structure	Pen and paper	Derivation	12	1.0	2.5	4.0	3.3	4.0	5.0
		Validation	9	1.0	2.0	2.0	2.3	3.0	3.0
		Total	21	1.0	2.0	3.0	2.9	4.0	5.0
	Tau UML Suite	Derivation	14	0.0	2.0	3.5	3.3	4.0	5.0
		Validation	12	1.0	2.0	2.5	2.8	4.0	5.0
		Total	26	0.0	2.0	3.0	3.1	4.0	5.0
	Total	Derivation	26	0.0	2.0	4.0	3.3	4.0	5.0
		Validation	21	1.0	2.0	2.0	2.6	3.0	5.0
Time	Pen and paper	Derivation	6	124	137	175	174	204	226
		Validation	4	161	165	171	182	200	227
		Total	10	124	161	172	177	204	227
	Tau UML Suite	Derivation	10	136	202	220	216	247	263
		Validation	5	180	185	231	224	262	264
		Total	15	136	185	224	219	258	264
	Total	Derivation	16	124	171	207	200	231	263
		Validation	9	161	172	185	206	231	264

Table 7. Hypotheses testing in Experiment 1

Dependent Variable: Completeness

R-Square	*Coeff Var*	*Root MSE*	*Completeness Mean*
0.219068	38.99366	1.244478	3.191489

Source	*DF*	*Type III SS*	*Mean Square*	*F Value*	*Pr > F*
Technique	1	14.34415584	14.34415584	9.26	0.0040
Tool	1	0.93506494	0.93506494	0.60	0.4414
Technique*Tool	1	5.84415584	5.84415584	3.77	0.0586

Dependent Variable: Structure

R-Square	*Coeff Var*	*Root MSE*	*Structure Mean*
0.100140	39.76367	1.192910	3.000000

Source	*DF*	*Type III SS*	*Mean Square*	*F Value*	*Pr > F*
Technique	1	6.04058442	6.04058442	4.24	0.0455
Tool	1	0.58603896	0.58603896	0.41	0.5245
Technique*Tool	1	0.85876623	0.85876623	0.60	0.4415

Dependent Variable: Time

R-Square	*Coeff Var*	*Root MSE*	*Time Mean*
0.260015	19.02129	38.43823	202.0800

Source	*DF*	*Type III SS*	*Mean Square*	*F Value*	*Pr > F*
Technique	1	410.403488	410.403488	0.28	0.6037
Tool	1	9998.543023	9998.543023	6.77	0.0167
Technique*Tool	1	0.170930	0.170930	0.00	0.9915

4.5. Explanatory Power of the Model

R^2 is usually defined as the proportion of variance of the response that can be explained by the regressor (independent) variables of a regression model. Note that, in this context, we use R^2 to explain the existing results, not to forecast future results. Table 7 shows R^2 values of 0.22, 0.10 and 0.26 for completeness, structure and time, respectively. Given the few regressor variables (only Tool and Technique) and a narrow range of the regressor values (only two values each), the R^2 values of 0.22 and 0.26 are reasonable, while 0.10 is low. Quoting from [26]: "Whether a given R^2 value is considered to be large or small depends on the context of the particular study. A social scientist might consider an R^2 of 0.30 to be large, while a physicist might consider 0.98 to be small." Few of the authors who report controlled software engineering experiments with humans as subjects discuss the explanatory power of their statistical models. Hence, there are no conventions for low/high R^2 (or related measures) in our area. Nevertheless, in addition to reporting p-values one should also discuss the explanatory power of the studied variables. For example, in the study reported here, there are, of course, factors other than technique and tool that may explain the results, such as the subjects' competence and motivation. This is an issue for future work.

5. Experiment 2

Experiment 2 was a replication of Experiment 1, with two differences: (i) students were replaced with professionals and (ii) all the participants used a professional modelling tool.

5.1. Subjects, Setting and Assignment

The subjects were 22 consultants from eight companies (one to seven persons from each company). All of them had used UML on software development projects. More than half of them had also studied UML as part of their education. We believe that they represent a typical sample of UML consultants. Table 8 shows the background of the consultants.

Table 8. Background of subjects in Experiment 2

	Min	Average	Max
Credits	50	94	140
UML Credits	0 (9 subjects)	10	30
Work experience (years)	1	5	12
UML experience (months)	1	13	50

The companies were paid normal consultancy fees corresponding to four hours for each consultant, independently of how long the consultants actually spent on the experimental tasks. As in the student experiment, they worked until they were finished.

During the experiment, the consultants from three of the companies were placed at Simula Research Laboratory, while those from the other five companies stayed in their usual work offices. All of the experiment sessions were run in an open concept office. As in the student experiment, this experiment was conducted using the Simula Experiment Support Environment. Two people were present during the experiments to help the consultants with understanding SESE and the experimental material. The experiments were conducted on five days within a one-month period.

The consultants were randomly assigned to one of the two techniques; 11 to the *derivation* technique and 11 to the *validation* technique. The consultants were asked to use the UML tool with which they were familiar. In this experiment, there was no blocking on tool. Table 9 shows the distribution of subjects on tool, technique and company.

Table 9. Distribution of subjects in Experiment 2

	Technique	Company	No. of subjects
Visio	Derive	A	2
		C	3
		D	2
	Validate	C	4
MagicDraw	Derive	B	2
	Validate	D	1
Rational Rose	Derive	F	1
		G	1
	Validate	E	1
		F	1
		G	3
		H	1

The project manager of two of the subjects had, just before the experiment, installed a new version of Rational Rose with which the subjects were not acquainted. This resulted in too much time spent on fumbling with the tool, so they gave up the experiment, and were removed from the analysis. One of them had been assigned to the derivation technique; the other to the validation technique.

5.2. Descriptive Statistics and Tests of Hypotheses

Table 10 shows the descriptive statistics and the hypothesis testing for this experiment. For completeness, both the median and the lower and upper quartiles are the same for both techniques. For structure, all of these measures are lower for the validation technique. Also for time, they are lower for the validation technique. The corresponding hypotheses (Section 3.3) were tested using a two-sample Kruskal-Wallis non-parametric test, but none of them was rejected (α=0.05).

Table 10. Descriptive statistics and hypothesis tests for Experiment 2

Score	Technique	N	Min	Q1	Median	Q3	Max	P-value	Reject
Completeness	Derivation	10	0.0	3.0	4.5	5.0	5.0		
	Validation	10	1.0	3.0	4.5	5.0	5.0		
	Total	20	0.0	3.0	4.5	5.0	5.0	0.9675	No
Structure	Derivation	10	2.0	2.0	3.0	4.0	5.0		
	Validation	10	1.0	1.0	2.5	3.0	4.0		
	Total	20	1.0	2.0	3.0	3.5	5.0	0.1841	No
Time	Derivation	6	144	194	243	269	271		
	Validation	4	163	164	197	246	263		
	Total	10	144	164	226	263	271	0.5224	No

6. Discussion

The results from Experiment 1 showed that the class diagrams constructed with the validation technique implemented more of the requirements. The difference between the two techniques was, however, larger for the subjects using pen and paper than for those who used the modelling tool (Tau UML). The hypothesis testing also indicated an interaction effect, although not significant, between technique and tool regarding completeness.

The class diagrams constructed with the derivation technique were structured better in both experiments, but the difference between the two techniques was larger for the subjects using pen and paper also in this case.

In our opinion, the use of a tool reduced the effects of the design techniques because the tool itself imposes a process on its users to a certain extent. In this case it may have been more difficult to follow the validation process than the derivation process when using a modelling tool because the tools facilitate the use of already defined classes and methods. Using such a tool, the changes in the classes in one diagram may propagate to other diagrams. (This feature is supported to a varying extent in different tools.)

The authors have found no other controlled experiment in the field of object-oriented design in which the subjects used professional modelling tools to support the design process. Hence, another goal of these experiments was to gain experience of conducting controlled experiments with such tools, including comparing the performance when using a tool with that when using pen and paper.

The subjects who used a modelling tool spent more time than did those who used pen and paper on obtaining similar quality. We believe that the subjects who used a tool instead of pen and paper spent some extra time on understanding how to perform the tasks with it, even though they were familiar with the tool used. Those who used the tool probably also spent more time on getting the syntax correct to avoid error messages. In Experiment 1, the subjects may also have

been hindered by some minor bugs in the Tau UML Suite, and by the fact that it was very slow in some periods due to the heavy load of 26 people working simultaneously.

The results from Experiments 1 and 2 differ to some extent, in particular regarding completeness. The professional subjects spent more time than did the students and obtained a higher score on completeness. A possible explanation is that as professionals representing companies, they may have been more motivated than the students to produce diagrams of high quality. However, the average performance, time taken into account, of the professional was not better than that of the students. This shows that experience of using UML on actual software projects does not necessarily lead to better performance than does the training received in university courses on UML, at least not for smaller tasks.

Spending more time on the experiment may in itself have led to the detection of more of the possible classes, and thus to reducing the effect of the techniques with respect to completeness.

The results showed that the use of a modelling tool required more effort from the subjects than did the use of pen and paper, and that the effects of the two techniques were smaller for the subjects who used a modelling tool.

7. Threats to Validity

This section discusses threats to the validity of the results of the two experiments.

7.1. Experimental Material

Use cases can be described with many different formats and with varying level of detail. The format of the use cases will affect how they can be applied. The format used in these experiments, "The fully dressed use case template" described in [10], is frequently used and rather detailed. Using another format or level of detail in the use cases may have an impact on the results from this experiment.

The same, small task, in which development was done from scratch, was used in the pilot experiment and the two experiments. This task may influence the results. In particular, the results may be different on a larger system, and if development is based on existing systems [1].

7.2. Tools Used in the Experiment

The subjects used different modelling tools in the two experiments, and as discussed in Sections 3.2.3 and 6, the choice of tools may influence the results. However, we were unable to get professional subjects who used Tau UML in their daily work. It was also infeasible to get a high

enough number of professional subjects using each of the different tools to compensate for possible differences between the tools.

7.3. Measuring Quality

It is difficult to define, and consequently to measure, the quality of a class diagram. The quality in terms of completeness is subjective, but the domain of the experiments was, in this case, based on a well-known example from textbooks. Moreover, three persons were involved in determining what would be a correct solution.

Well-defined metrics for measuring coupling and cohesion exist, for example, those described in [8,9], but since the class diagrams produced in these experiments were quite small and simple, these metrics could not be applied easily. Therefore, coupling and cohesion were also measured subjectively.

8. Lessons Learned from Organising the Experiments

An important motivation for using professional modelling tools was to gain experience of conducting experiments with such tools, because in our opinion traditional pen-and-paper based exercises are hardly realistic for dealing with relevant problems of the size and complexity of most contemporary software systems. We have found that it is a challenge to configure an experimental environment with an infrastructure of supporting technology (processes, methods, tools, etc.) that resembles an industrial development environment. Our experience from replicating several experiments with the use of professional tools is that the use of system development tools requires proper preparation [30]:

- Licences, installations, access rights, etc. must be checked.
- The tools must be checked to demonstrate acceptable performance and stability when many subjects are working simultaneously.
- The subjects must be, or become, familiar with the tools.

Hiring consultants as subjects may make these aspects simpler, because one can then agree with the companies that they should be responsible for the tool environment. This was the case in Experiment 2. However, in spite of part of the contract that the consultants should know UML and use a tool with which they were familiar, two of the of the consultants informed us that they were given a version of Rational Rose they did not know, and consequently had to give up the experiment.

9. Conclusions and Future Work

Several approaches have been proposed for moving from functional requirements to a design model, but these approaches have been subject to little empirical validation. A use case-driven development process, in which the use case model is the principal basis for a class diagram, is recommended together with UML, but no established, empirically validated technique for the transition from use case models to the construction of class diagrams exists. This paper identified two alternative approaches to this transition; the classes are either derived directly from the use cases (called the derivation technique), or the use case model are applied as a means of validating a class diagram constructed using another approach, for example, grammatical analysis of requirements documents (called the validation technique).

We conducted a pilot experiment with 26 students as subjects, an experiment with 53 students as subjects and finally an experiment with 22 professional software developers as subjects to compare these two techniques. The aim of the experiments was to investigate differences between the two approaches with respect to the quality of the resulting class diagrams in terms of completeness and structure, and with respect to the differences in time spent on obtaining a good design.

The results from the pilot experiment were confirmed in the first experiment, which showed that the validation technique resulted in class diagrams that implemented more of the requirements. The derivation technique resulted in class diagrams with a significantly better structure than did the validation technique in the student experiment and slightly better structure in the experiment with the professionals. There was no difference in time spent between the two techniques.

The results support the claims that deriving classes directly from a use case model may lead the developers to miss some requirements, but it does not support the claim that it leads them to mistake requirements for design. We also believe that, based on these results, it may be beneficial to derive classes directly from the use cases when the use case model contains many details and there is a strong need for good structure, but that otherwise it is better to apply the use case model in validation.

In the experiments, the use of the tool did have an effect on the results. Hence, there may be a threat to the external validity of the results of experiments on object-oriented design that are conducted using pen and paper only. Note, however, that the use of tools requires more effort from the subjects, but, on the other hand, in our experiments the subjects using the tool seemed more motivated.

We intend to conduct further studies to investigate how to apply a use case model in an object-oriented design process. In particular, we intend to

- increase the size and complexity of the task,
- use different use case formats,

- compare different tools by having a larger number of subjects using each tool,
- improve the collection of background data, as well as process information during the experiment, to study which process attributes and skills actually affect the quality of the object-oriented design, and
- extend the evaluation of the quality of the resulting class diagrams by combining several aspects.

Acknowledgements

We acknowledge the students at the University of Oslo and the consultants from the companies Bekk, Genera, Halogen, Machina, ObjectNet, Software Innovation, TietoEnator and Unified Consulting who participated in the experiment. We also acknowledge Eskild Bush for support on the use of the Tau UML Suite and Kjell Jahr from Telelogic for technical assistance with this tool; Gunnar J. Carelius for adapting the experiments for use with the Simula Experiment Support Environment (SESE) and for support during the experiments; Dag Solvoll, Yngve Lindsjørn and Wiggo Bowitz from KompetanseWeb for implementing the necessary changes to SESE; Per Thomas Jahr from the Norwegian Computing Center for analysing the class diagrams; Terje Knudsen from the Department of Informatics, University of Oslo and Arne Laukholm, Director of the Computer Centre, University of Oslo, for technical support; and Tanja Grutshke, Christian Herzog, Tor Even Ramberg and Sinan Tanilkan for debugging the experimental material. We would also like to thank Erik Arisholm, Chris Wright and the anonymous referees for their valuable contributions to this paper.

References

1. Anda, B., Hansen, K., Gullesen, I. and Thorsen, H.K. Experiences from Introducing UML-based Development in a Large Development Project. Submitted to *Empirical Software Engineering,* 2004.
2. Arisholm, E., Sjøberg, D.I.K., Carelius, G.J. and Lindsjørn, Y. A Web-based Support Environment for Software Engineering Experiments. *Nordic Journal of Computing*, 9: 231-247, 2002.
3. Arlow, J. and Neustadt I. *UML and the Unified Process. Practical Object-Oriented Analysis and Design*. Addison-Wesley, 2002.
4. Bahrami, A. *Object Oriented Systems Development*. McGraw-Hill, 1999.
5. Batra, D., Hoffer, J.A. and Bostrom, R.P. Comparing Representations with Relational and EER Models. *Communications of the ACM*, 33(2): 126-139, 1990.
6. Bennett, S., McRobb, S. and Farmer, R. *Object-Oriented Systems Analysis and Design using UML*. McGraw-Hill, New York, 1999.
7. Booch, G., Rumbaugh, J. and Jacobson, I. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

8. Briand, L.C., Daly, J. and Wüst, J. A Unified Framework for Cohesion Measurement in Object-Oriented Systems. *Empirical Software Engineering*, 3(1): 65-117, 1998.
9. Briand, L.C, Daly, J.W. and Wüst, J. A Unified Framework for Coupling Measurement in Object-Oriented Systems. *IEEE Transactions on Software Engineering*, 25(1): 91-121, 1999.
10. Cockburn, A. *Writing Effective Use Cases*. Addison-Wesley, 2000.
11. Dobing, B. and Parsons, J. Understanding the Role of Use Cases in UML: A Review and Research Agenda. *Journal of Database Management*, 11(4): 28-36, Oct-Dec 2000.
12. Freund, R.J., Littell, R.C., and Spector, P.C., SAS System for Linear Models, Cary, NC: SAS Institute Inc., 1991.
13. Harrison, W. N=1: An Alternative for Software Engineering Research? Beg, Borrow, or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research, Workshop, 5 June, 2000 at 22nd Int. Conf. on Softw. Eng. (ICSE), Limerick, Ireland, pp. 39-44, 2000.
14. IBM Rational XDE Developer Plus, V2003.06.00, http://www-306.ibm.com/software/awdtools/developer/rosexde/, 2003.
15. Jacobson, I., Christerson, M., Jonsson P. and Overgaard, G. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley, 1992.
16. Jacobson, I., Booch, G., and Rumbaugh, J. *The Unified Software Development Process*. Addison-Wesley, 1999.
17. Karahasanovic, A., Anda, B., Arisholm, E, Howe, S.E., Jørgensen, M., Sjøberg, D.I.K. and Welland, R. Collecting Feedback during Software Engineering Experiments. Accepted for publication in *Empirical Software Engineering*.
18. Larman, C. *Applying UML and Patterns—an Introduction to Object-Oriented Analysis and Design and the Unified Process*. Prentice-Hall, Englewood Cliffs, NJ, 2002.
19. Lethbridge, T.C. and Laganiere, R. *Object-Oriented Software Engineering: Practical Software Development Using UML and Java*. McGraw-Hill, New York, 2001.
20. Liang, Y. From Use Cases to Classes: A Way of Building Object Model with UML. *Information and Software Technology*, 45(2): 83-93, 2003.
21. Lindsay, R.M. and Ehrenberg, A.S.C. The Design of Replicated Studies. *The American Statistician*, 47(3): 217-228, August 1993.
22. Maciaszek, L. *Requirements Analysis and System Analysis*. Addison-Wesley, 2001.
23. Meyer, B. *Object-Oriented Software Construction*. Prentice-Hall, 1997.
24. Richter, C. *Designing Flexible Object-Oriented Systems with UML*. Macmillan Technical Publishing, 1999.
25. Rosenberg, D. and Scott, K. *Applying Use Case Driven Object Modeling with UML. An Annotated E-commerce Example*. Addison-Wesley, 2001.
26. SAS Institute Inc., SAS Version 8, Cary, NC, USA, 1999.
27. Sharble, R.C. and Cohen, S.S. The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods. *Software Engineering Notes*, 18(2): 60-73. 1993.
28. Shull, F., Travassos, G., Carver, J. & Basili, V. Evolving a Set of Techniques for OO Inspections. University of Maryland Technical Report CS-TR-4070, October 1999.
29. Sjøberg, D.I.K, Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A., Koren, E.F. and Vokác, M. Conducting Realistic Experiments in Software Engineering. ISESE'2002 (First

International Symposium on Empirical Software Engineering), Nara, Japan, October 3-4, pp. 17-26, IEEE Computer Society, 2002.
30. Sjøberg, D.I.K., Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A. and Vokác, M. Challenges and Recommendations when Increasing the Realism of Controlled Software Engineering Experiments, R. Conradi and A. I. Wang (Eds.): Empirical methods and studies in software engineering, ESERNET 2001-2002, LNCS 2765, pp. 24-38, Springer-Verlag, 2003.
31. Stevens, P. and Pooley, R. *Using UML. Software Engineering with Objects and Components*. Addison-Wesley, 2000.
32. Syversen, E., Anda, B. and Sjøberg, D.I.K. An Evaluation of Applying Use Cases to Construct Design versus Validate Design, Hawaii International Conference on System Sciences (HICSS-36), Big Island, Hawaii, January 6-9, 2003.
33. http://www.telelogic.com/products/tau/uml/
34. Wirfs-Brock, R., Wilkerson, B. and Wiener, L. Designing Object-Oriented Software. Prentice Hall, 1990.
35. http://www.magicdraw.com
36. http://www.microsoft.com/office/visio/

Appendix A. The exercise guidelines

Guidelines for the derivation technique	Guidelines for the validation technique
Exercise 1: Domain model 1. Underline each noun phrase in the use case descriptions. Decide for each noun phrase whether it is a concept that should be represented by a class candidate in the domain model. 2. For the noun phrases that do not represent class candidates, decide whether these concepts should be represented as attributes in a domain model instead. (Not all attributes are necessarily found this way.)	**Exercise 1: Initial lass diagram** 1. Underline all noun phrases in the requirements document. Decide for each noun phrase whether it is a concept that should be represented by a class in the class diagram. 2. For the noun phrases that do not represent classes, decide whether these concepts should be represented as attributes in the class diagram instead. (Not all attributes are necessarily found this way.) 3. Find the verbs or other sentences that represent actions performed by the system or system classes. Decide whether these actions should be represented by one ore more methods in the class diagram. (Not all methods needed are necessarily identified this way.)
Exercise 2: Sequence diagrams 1 Create one sequence diagram for each use case. 2. Study each use case description carefully, and underline the verbs or sentences describing an action. Decide for each action whether it should be represented by one or more messages in the sequence diagrams. (Not all methods needed are necessarily identified this way)	**Exercise 2: Sequence diagrams** 1. Create one sequence diagram for each use case. 2. Study each use case description carefully, and underline the verbs or sentences describing an action. Decide for each action whether it should be represented by one or more methods in the sequence diagrams. (Note! Not all methods needed are necessarily identified this way)
Exercise 3: Class diagram 1. Transfer the domain model from exercise 1 into a class diagram. 2. Use the three sequence diagrams from exercise 2 to identify methods and associations. For each method in the sequence diagram: o If an object of class *A* receives a method call *M*, the class *A* should contain the method *M* in the class diagram. o If an object of class *A* calls a method of class *B*, there should be an association between the classes *A* and *B*.	**Exercise 3: Validation of the class diagram** 1. Consider each method in the sequence diagram. If several methods together form a system service, treat them as one service. 2. For each method or service: o Confirm that the class that receives the method call contains the same or matching functionality. o If an object of class A calls a method of class B, there should be an association between the classes A and B in the class diagram. If the class diagram contains any hierarchies, remember that it may be necessary to trace the hierarchy upwards when validating it. If the validation in the previous steps failed, make the necessary updates.

Appendix B. Example of evaluation of one class diagram

ID	**49**
Completeness - All services described in the requirements specification are implemented (Aspect 1)	**Score: 5**
Lending item	X
Hand in item	X
Search for item based on id	X
Search for book based or title or author	X
Search for video based on title	X
Show status for an item	X
Completeness – The services are allocated to all and only the correct classes (Aspect 2)	**Score: 3**
Library	
Lend item	X
Hand in item	X
Search	X
Loan	
Binding between item and loaner	X
Time period for loan	X
Item	
Information about title, author and id	X
List of copies	
Copy	
Loan status	
Loaner	
Information about loaner	
Video	
Loan period for video	
Book	
Loan period for book	
Completeness - The classes contain the necessary information in terms of attributes and associations (Aspect 3)	**Score: 3**
Library	
All items	X
All loaners	X
All loans	X
Loan	
Period (start and end date)	X
Loaner	X
Copy	X
Item	
All it's copies	
Item id	X
Title	X
Copy	

It’s item	
It’s loan	
Copy id	
Status	
Loaner	
Loans	X
Loaner Id	X
Video	
Book	
Author	
Completeness – Total score	**Score: 4**
Structure – Coupling and cohesion	**Score: 4**
Is one class dependent on many other classes?	
How well do the methods in the classes correspond with the distribution of responsibilities identified above?	
How simple and focused are the methods?	

Appendix C. Examples of feedback comments for one subject in Experiment 1

Min.	Task	Comment
21.30	1	I have read through the specifications, and underlined the nouns.
35.68	1	I have almost finished the domain model, and I consider which associations and attributes should be included.
51.42	1	I consider the possibility of dividing the inheritance between article, book and film with respect to article number in articles and copy number in copies.
70.63	2	I look for verbs, etc. that can help me identify methods. At the same time I think about the calling structure between the classes that was identified in the domain model.
82.35	2	I draw sequence diagrams on paper, it is difficult to anticipate in advance so that I don't have to make too much clutter on the paper. This would have been much quicker with a modelling tool.
97.15	2	I am a bit uncertain about how to organize the information. Should I model a register or not regarding how to extract data?
112.02	2	Use case 2: Draw the sequence diagram according to the description.
126.17	2	Finished use case 3, it was easy according to the description. Started on task 3.
141.43	3	Draw the class diagram and add methods and attributes. Consider the validity of the methods.
158.00	3	Add variables to the methods and insert associations.