University of Oslo
Department of Informatics

Comparing two Ways of Applying Use Case Models in Object-Oriented Design with UML

Erik Syversen

Hovedoppgave

October 7, 2002

Contents

List of Tables

List of Figures

Abstract

In software engineering most proposed technology have not been subject to a thorough investigation. Knowledge of the effects, strenghts, and weaknesses of technology is essential to ensure a successful transfer to industry and identify areas of improvement. For example, the Unified Modeling Language (UML) is becoming the de-facto standard for object-oriented software analysis and design. Constructing analysis- and design models of a software system is beneficial to gain a higher-level understanding of the system domain prior to implementation. Models provide blueprints of possible solutions and are beneficial to comprehending a complex system in its entirety, ensure a sound architecture, and to support communication among project developers and teams. However, UML does not *prescribe* a development process, nor has it undergone a thorough investigation to identify how it is best applied in a software development process. A use case driven development process, where a use case model is the basis for *constructing* an object-oriented design, is *recommended* with UML. An alternative way of applying a use case model is to use it to *validate* a design model made as part of another development process. We are interested in the differences imposed on design models constructed between such processes. In order to investigate the differences, we conducted a controlled experiment with 26 students as subjects. One half of the subjects were given guidelines on a use case driven process whereas the other half on a validation process. The resulting class diagrams were evaluated with regards to how well they were suited as a basis for automatic code generation. The results show that the validation process produced class diagrams better suited for automatic code generation than those produced in the use case driven process. In particular, the subjects applying the use case driven process primarily identified methods that were mapped directly from the use case descriptions, something that resulted in design models with fewer appropriate methods than those produced with the validation process. The results also show that the use case driven process resulted in more erroneous classes, and the validation process led to a larger variation in the *number* of classes in the class diagrams. This indicates that the use case driven process gave more, but not always more appropriate, guidance on how to construct a class diagram. In summary, our study implies that following a validation process is the better choice in similar conditions to our experiment.

Acknowledgement

I would like to express gratitude to my supervisor, PhD Student Bente Anda, and Simula Research Laboratory for giving me the opportunity to conduct an experiment. This gave me the chance to design and carry out my own little empirical study. I am also grateful for the guidance I have received from my supervisor.

I would like to thank professor Dag I.K. Sjøberg for his involvement and valuable contributions.

I acknowledge all the students at University of Oslo who took part in the experiment.

I would like to thank my student colleagues and friends for valuable discussions and support throughout my student carreer.

Last but not least, my sincerest gratitude goes to my familiy for their support, encouragement, and inspiration. Thanks!

1 Introduction

Technologies (processes, methods, programming languages, tools, etc.) are introduced to the software industry claiming to improve software production and the quality of the resulting products. However, little or no evidence exists to support the claims [4,17]. As software researchers and practitioners recognize a direct correlation between the quality of the software process and the quality of the resulting software product, proposed technology need to be investigated thoroughly. Knowledge of the effects, strengths, and weaknesses of technology is essential to ensure a successful transfer to industry and identify areas of improvement [4,17,29]. Such knowledge may also lead organizations to adopting technology they otherwise would not have considered because any benefits will be realized [17].

Empirical studies are a means to increase our understanding of the phenomena under study. In software engineering, empirical studies are essential in order to assess, validate and provide evidence on the usefulness and effects of technology, improve existing technology, and develop new technology [28,29]. An overview of empirical research and results in software engineering is provided in [4]. Among future research directions it is suggested that software development processes, such as the Unified Process, need to be investigated and compared to alternatives in order to be successfully transferred into industrial practice, and that the use of the Unified Modelling Language (UML) need to be investigated more thoroughly since it is now becoming a de-facto standard for object-oriented analysis and design.

A use case driven development process is recommended when applying UML [2,3,15,16,32]. In a use case driven process the use case model, possibly in combination with a domain model, serves as the basis for deriving all subsequent models. Use case driven development processes have, however, been criticized for not providing a sufficient basis for the construction of a design model. For example, in [22] it is claimed that:

- The use case model alone is insufficient for deriving all necessary and appropriate classes.
- Focusing on the use case model leads the developers to mistake requirements for design.

It may therefore be more beneficial to construct a design model using alternative processes. In [22], a combination of use case driven, responsibility-driven, and data-driven processes is recommended. In [24], a responsibility-driven process, where a use case model is applied subsequently to validate the design model, is suggested as an equal alternative to a use case driven process. In the following, the term "validation process" is used to denote such a development process.

We are interested in differences between a use case driven process and a validation process in constructing design models. This may influence the choice of development process, even though the choice of development process in a software project is typically determined by characteristics of that project, such as the experience and skill of the developers, the problem domain and existing architecture, and may also influence how to teach object-oriented design. To investigate the differences, we conducted an experiment with 26 undergraduate students as subjects where the task was to construct a design model of a library system consisting of class and sequence diagrams. One half of the subjects were given guidelines on a use case driven process whereas the other half was given guidelines on a validation process. The processes investigated were based on the use case driven process and the validation process described in [24]. The resulting design models were evaluated relative to how well they specified a class diagram suited for automatic code generation. The class diagrams should implement the requirements, and should be specified at an appropriate level of syntactic granularity. Suitability for automatic code generation is important because it helps ensure consistency between models and code. We also investigated differences in size and number of errors in the class diagrams.

The results show that the validation process resulted in design models that better described the requirements and contained class diagrams better suited for automatic code generation. The subjects following the use case driven process mostly mapped the steps of the use case descriptions directly onto methods in the class diagram, while those following the validation process were more successful in deriving appropriate methods from the written requirements document. The results also show that the use case driven process led to more erroneous classes, and the validation led to a larger variation in the *number* of classes in the class diagrams. This indicates that the use case driven process provides more, but not necessarily better, guidance. In our opinion, the results support the claims that a use case model is insufficient for deriving the appropriate classes and may lead the developers to mistake requirements for design. The results also indicate that it may be more appropriate to consider a use case as a behavioural feature of a system against which class diagrams can be validated, rather than consider a use case as having state and behaviour from which the design can be derived.

The remainder of this paper is organized as follows:

- Section 2 discusses software process improvement and empirical software engineering with an emphasis on software experimentation.

- Section 3 gives a brief description of UML and the models relevant to this paper, and discusses how it is currently recommended applied in a development process. An elaboration of the two process investigated in this paper is also given.

- Section 4 presents the framework for the analysis of the experiment.
- Section 5 describes the experimental design, analysis and results, and suggests areas of improvement.
- Section 6 concludes and suggests future work.

2 Software Process Improvement

A *software process* is defined as the set of activities, methods, and practices developers use to produce and maintain a software product [9,14,30]. A *software process model* outlines the framework of a software process from a certain perspective [27]. From a workflow perspective, a process model outlines the sequence of activities needed to produce and maintain a software product. Generic activities in all software process models are [27]:

- Specification - what the system should do, and its development constraints.
- Development - production of the software system.
- Validation - checking that the system is coherent with customer requirements
- Evolution - changing the software system in response to changing demands.

>From other perspectives, process models may show how developers are coordinated on activities and how information flows within an organization during development [27]. A software process thus includes any aspect related to software production and maintenance.

A software product is often behind schedule, over budget, non-conforming to requirements, and of poor quality. Controlling and improving the process used to produce and maintain software products has been proposed as the solution to these problems [6,9,14,30]. The software industry is therefore becoming increasingly aware of the competitive advantages gained from conducting *software process improvement* [18]. This means continuously assessing, refining, and changing the process to increase its ability to meet the requirements and expectations of company stakeholders and of the market [9]. For example, organizations may have strategic goals, such as to develop reusable code, reduce development effort, users may demand usability or reliability, and the market may expect an organization to be up to date with current technology.

In industry, software process improvement can be considered a process in its own right. The software process needs to be assessed in order to determine its current state relative to strategic goals. Based on this assessment an improvement plan, that outlines the changes that will be introduced in order to improve the process according to the strategic goals, needs to be created. Then, the changes need to be implemented and the effects of change need to be monitored and evaluated. Software process *improvement methods*, such as SPICE and IDEAL, specify generic activities in a cyclical process for conducting software process improvement within an organization [19,30]:

1. Initiating - to start the improvement effort.
2. Diagnosing - to assess the current state of practice.
3. Establishing - to determine the improvement plan.
4. Acting - to implement the actions in the improvement plan.
5. Leveraging - to analyse the lessons learned and results of the improvement effort.

Quality models such as The SEI Capability Maturity Model (CMM) and the ISO 9001 standard can be used as references to assess an organization's state regarding overall process quality based on the requirements of an ideal company [9]. For example, CMM evaluates a software process relative to five levels of maturity. Process maturity implies the capability of a process: to what extent it is explicitly defined, managed, controlled, and effective [30]. The maturity levels differentiate potential for growth in capability. When evaluating the CMM level of an organization, a process improvement proposal is produced based on questionnaires or inspecting tangible process artefacts [6]. The CMM may thus be used as a roadmap for process improvement within an organization.

2.1 Empirical Software Engineering

Empirical studies are essential to support software process improvement [9]. The three main empirical research methods applied in software engineering are highlighted in [18,31]:

- Case studies. A study focusing on a single project.
- Formal experiments. A study focusing on multiple projects or a single project replicated several times.
- Surveys. A survey is similar to a formal experiment. If the selection of projects and teams is post hoc it is a survey.

Organizations usually conduct software process improvement through case studies. In a case study, the measurement framework is derived from a specific project setting and specific goals, thus the results are of value to *that* particular type of project. Projects are due regardless of data-collection, thus case studies are also the least expensive for an organization [31].

Organizations constantly seek technology that improves software production and maintenance [18]. When deciding on an improvement strategy that involves introducing new technology, organizations will benefit from any available evidence on the usefulness and effects of the technology. This will aid organizations in selecting the *appropriate* technology and reduce the risks concerning introducing *inappropriate* technology [29]. This may also lead organizations to adopting technology they otherwise would not have considered because the benefits are proven [17,29]. However, the usefulness and effects of a technology needs to be proven valid in many different settings, due to different use in different organizations. The evidence must therefore be based on a study that produces results that are applicable across many different types of projects and organizations. Experiments may produce such results.

2.2 Experimentation in Software Engineering

Although no amount of experimentation can provide absolute evidence on the validity of a theory as it always stand in danger of being falsified by a counter-example, experiments test theories, explore factors of interest, and may trigger ideas on new theories [29]. Research in other scientific fields has shown that when an extensive base of empirical evidence supports a theory it will gradually be accepted as a *valid* theory in that field [29]. For these reasons, experimentation is considered essential to making progress in any scientific field, including software engineering [4,29].

With a correlation between the quality of the software process and the quality of the resulting product, *processes* in particular need to be fully understood and improved to advance in the quality of softare products. Experiments may contribute by exploring critical factors and effects of a process. The knowledge derived and collected from experimental studies of a process may lead to a higher understanding of that process which may contribute to identifying areas of improvement in that process.

In order to make claims on their results, experiments must be well designed by having internal- and external validity. Internal validity means the experimental design must be based on valid theory and carried out through correct scientific practice [9]. For example a valid measurement programme must be used, data must be collected in an objective manner, and conclusions must have scientific support. There are problems identified with achieving internal validity, such as biased researchers aiming to prove one alternative superior to another [29], an invalid measurement programme [4], or conclusions drawn without scientific support [29]. External validity means results that are valid in other settings [9,18]. The results of an experiment do not generalize outside the controlled experimental conditions [18], thus it is important to determine the external validity of an experiment by precisely specifying its conditions.

If the goal of an experiment is to transfer results from research to industry it is therefore essential that there is a close relationship between the experimental conditions and real industrial situations [28]. Without such a relationship, organizations may perceive the results as irrelevant and ignore them [28]. However, as most experiments are conducted by researchers in a laboratory setting, this relationship might be difficult to achieve [18]. As a solution, it is suggested that experiments should be designed and carried out as realistic as possible [28]. An experiment is considered realistic if it motivates and appears meaningful to the subjects and resembles "real" industrial situations [28]. The latter is referred to as mundane realism and is achieved by having:

- A realistic task - the task should resemble real industrial projects.
- Realistic subjects - subjects should be representative of software developers.
- Realistic environment - the experimental environment should resemble a real industrial development environment.

2.3 Measurements

When investigating the effects of technology in empirical studies, measurements on both the process and the resulting product are needed. Specifically what needs to be measured is relevant to the goal and nature of the individual study, but three main dimensions for measurement related to software production and maintenance are highlighted in [19]:

- Cost - by devlopment effort, maintenance effort, training costs, cost of tools ets.
- Quality - by usability, reliability, maintainability, etc.
- Schedule - by project duration, maintenance duration, etc.

Internal structural attributes such as inheritance, coupling, cohesion, etc., are assumed closely related to external quality attributes, such as fault proneness, reusability, maintainability, etc. [4]. Therefore, many software metrics for internal structural attributes have been developed, with perhaps the most commonly referenced those by Chidamber and Kemerer [5]. Such metrics collects data on code. Recent research has identified the need for metrics for higher-level design in order to evaluate quality early on in the process lifecycle [4]. Metrics especially developed for UML class diagrams are for example presented in [10], and [23] presents a model for refining traditional object-oriented metrics to be applicable on higher-level design (e.g. UML).

3 Applying the UML in Object-Oriented Software Development Processes

The UML is a notation for object-oriented software analysis and design. Its notation is specified by a set of grammatical rules for constructing analysis- and design models of software systems, and defined by a meta-model [32]. The UML itself does not prescribe or give advice on how to use the notation in a software development process [3,8,32].

3.1 UML Models

The UML provides a set of diagrams for modelling the requirements, structure, state, and behaviour of a software system. When applied in a development process, diagrams are grouped together forming analysis- and design models of the system [2,15,16,25]. This section elaborates on the diagrams relevant to this paper, and how they are used in the respective models.

3.1.1 Use Case Diagrams and Use Case Modelling

A use case model captures interactions between the system and its users. A use case represents one complete service required by a system. For example, registering a loan in a library system is a use cases in such a system. A use case is related to its actor(s), which are the human users or external systems that require the use cases. A use case diagram shows the relationship between the actors and use cases of the system. Written descriptions of use cases, following some guideline such as the formats in [7], specifies each use case and the flow of events necessary for realizing each use case scenario. A use case scenario is one particular path through a use case. Graphical descriptions of use cases, using for example activity or sequence diagrams, may also describe use cases. However, these are less appropriate for validating requirements in collaboration with users and customers since they cannot be expected to be familiar with UML. A use case model is the collection of the use case diagram and written or graphical descriptions of the use cases.

The common approach to constructing use case diagrams is to first identify the actors, then the use cases required by the actors [2,3,8,24]. The identification of actors can be made by abstraction in the problem domain, investigating nouns that represent actors in written requirements documents, or by consulting the customer(s). The use cases are identified by imagining services required by actors, investigating verb sentences that represent use cases in requirements documents, or consulting human actors.

3.1.2 Sequence Diagrams

Sequence diagrams show how entities of the system interact in order to execute a use case scenario. When applied in an analysis model, the entities are conceptual classes, separated into boundary, control, and entity objects [2,15,16,25]. The interaction is shown by how these objects delegate responsibilities between them. The responsibilities are the high-level operations that the system must be able to perform to realize use cases. When applied in a design model, the entities are system class instances and the interaction is shown by the flow of method calls between the class instances.

3.1.3 Class Diagrams

A class diagram shows the static structure of a system's concepts, types, or classes. A class diagram is constructed in one of three different *perspectives* [8]. When applying class diagrams in analysis models, the perspective is *conceptual*. Such a class diagram is referred to as a domain model, and shows the entities, which are class candidates, in the domain under study, their high-level- attributes, operations, and relationships [2,15,16,25]. When applied in a design model, the perspective is either *specification* or *implementation*. In a specification perspective interfaces of software components are modelled. In an implementation perspective the system classes, their attributes, methods, and associations are shown.

3.2 Recommended Use

The UML meta-model defines a use case as a subclass of the meta-model element Classifier [32]. This implies that a use case has a state and behaviour from which classes, attributes, methods, and associations can be derived. In our opinion a use case driven development process implicitly assumes this definition and that the use case model contains all information necessary for constructing analysis- and design models.

The Unified Process, which is considered to be the process that best complements UML [2], is the primary example of a use case driven process. It is developed by the authors of UML and covers their recommended application of all UML models. In the Unified Process, an analysis model is derived from the use case model, possibly in combination with a domain model, and a design model is a refinement of the analysis model. The design model contains the same diagrams as the analysis model, but with a higher-level perspective. This stepwise process of transitioning a use case model to a design model is claimed to ensure traceability from requirements to design [2,16]. The result of implementing the system according to the class diagram in the design model is thus a system architecture that to a great extent will be determined by the information retrieved and contained in the use case model and how this is refined to a design model.

The extensive framework provided by the Unified Process allows for several interpretations and adaptations to different development projects. For example, the ICONIX process described in [25] is a derivate of the Unified Process, and the use case driven process described in [24] may be considered a simplification of the Unified Process. The steps of the use case driven process, as described in [24], are presented in Figure 1. The actual use case driven process investigated in this paper is outlined in Figure 2.

---

1. Identify the use cases of system behaviour
2. Describe each use case in detail, for example using a template format.
3. Define a scenario for each interesting path through the use case, for example using an activity diagram.
4. Draw a sequence diagram for each scenario.
5. Identify the methods needed in every scenario, thus all the methods needed for the realization of the use cases.
6. Transfer the objects and methods in the sequence diagrams to a class diagram.

Figure 1: The steps in a use case driven process

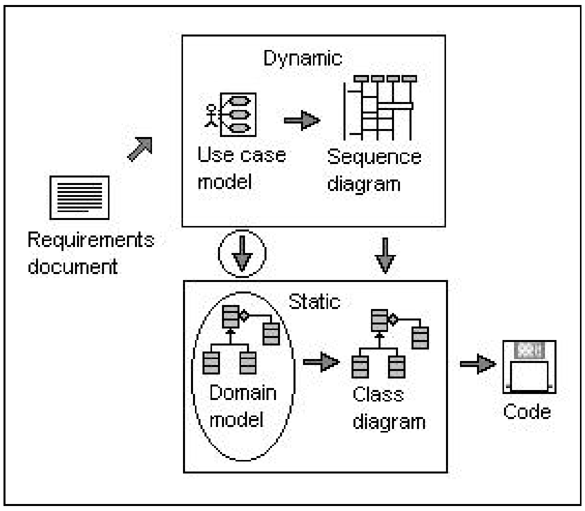


Figure 2: Outline of the use case driven process

3.2.1 Problems and Proposed Solutions

It is often difficult to ensure a consistent transition of models, due to their different aim and different perspectives of diagrams [12,20,25]. Solutions to this problem have been suggested. For example:

- In [20] the refinement of activity diagrams are used to ensure a traceable transition and proper coupling of use cases and class diagrams.
- In [12] semi-formal models are introduced as means to systematically identify corresponding or redundant information in models depicting use case scenarios and class diagrams.
- A set of reading techniques for inspecting the quality and consistency of diagrams and other artefacts such as class descriptions, are used as a means to validate a design model according to requirements and ensure consistency in [26].
- The process in [25] places a stronger emphasis on the analysis phase by constructing a more elaborate analysis model in the attempt to bridge the gap between the use case model and the class diagram.

In [22] it is claimed that a use case driven development process may lead to an unsuitable architecture. For example:

- The use case model does not contain the information necessary for deriving all or the appropriate classes.
- Focusing too much on the information contained in the use case model may lead developers to mistake requirements for design.
- Focusing on realizing the use cases in the current iteration may cause developers to loose sight on structural quality aspects, thus prohibiting maintainability, especially if the schedule to complete the iteration is tight.

The proposed solution to these problems in [22] is to apply a process that combines and includes aspects of use case driven, data-driven, and responsibility driven processes. Data-driven and responsibility-driven processes derive a class diagram by investigating nouns and verb phrases in written requirements documents, and possibly by abstraction in the problem domain. They differ in their starting points: A data-driven process starts by identifying classes and their state (attributes and associations) whereas a responsibility-driven process starts by identifying classes and their responsibilities [22]. In such a combination the design model is derived from written requirements documents, and possibly abstraction, in addition to the use case model. The use case model will thus less determine the system's architecture.

In [24], a validation process is suggested as an equal alternative to a use case driven process. In such a process the use case model only contributes to detecting required functionality overlooked in validating the class diagram. Thus, the use case model will not determine classes or the class composition, and in our opinion a validation process supports the definition of a use case as a behavioural feature of a system, rather than as a classifier.

The steps of a validation process as described in [24], is presented in Figure 3. The actual validation process investigated in this paper is outlined in Figure 4. The differences between Figures 2 and 4 are encircled.

1. Identify the classes in the system using abstractions or noun phrases in requirements documents.
2. Make a list of the responsibilities of the objects of each class.
3. Draw a class diagram to show the classes and their responsibilities, that is their methods.
4. Identify interesting scenarios that will be used to validate the class diagram. To do this, identify and model the use cases.
5. Draw a sequence diagram for each scenario.
6. Use the sequence diagrams to validate that the class diagram contains all the necessary methods. Add or rearrange methods if necessary.

Figure 3: The steps in a validation process

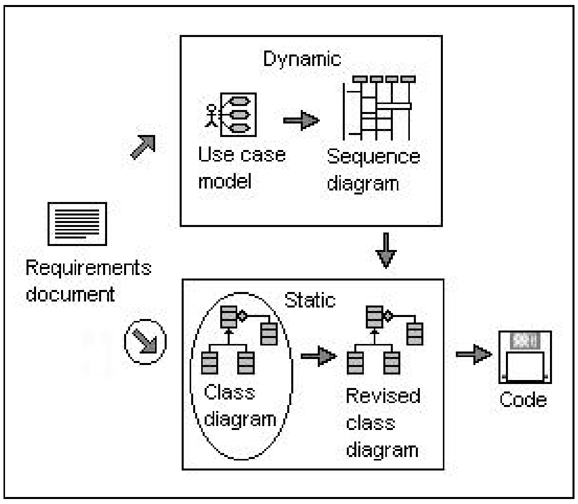


Figure 4: Outline of the validation process

3.3 Tool Support

Most UML CASE tools support automatic generation of code from class diagrams to common object-oriented programming languages, such as Java and C++. The resulting code is a framework of the system where ideally only the contents of methods need to be implemented. This provides a controlled transformation of design models to code, ensuring consistency.

4 Analysis Framework

The design models produced in the experiment were evaluated relative to both qualitative and quantitative metrics. The qualitative metrics describe the class diagrams' suitability for automatic code generation and to what extent the subjects were successful in following the guidelines for the two different processes. The quantitative metrics describe the size and number of errors in the class diagrams. The time spent on creating the design models was also measured.

4.1 Qualitative Metrics

We propose a set of metrics for measuring the level of detail of the class diagrams, the extent to which the design models implement the requirements, and the correspondence between class and sequence diagrams. Our metrics are based on the marking scheme for evaluating quality properties of a use case description presented in [1].

The level of detail was measured relative to the syntactic elements that were used in the resulting class diagrams. To support code generation, the class diagrams should show the:

- Visibility of attributes and methods.
- Type and names of attributes, methods and parameters.
- Navigation and cardinality of associations.

Realism of the class diagrams, to what extent the processes are successful in constructing design models that implement the requirements, was measured in three ways:

- Realism in class abstractions - to what extent the class diagrams contain the necessary classes.
- Realism in class attributes - to what extent the necessary attributes are identified, and whether they are specified as attributes of the correct classes.
- Realism in class methods - to what extent the necessary methods are identified, and whether they are specified as methods of the correct classes.

The realism of sequence diagrams was measured similar to the realism of class methods. The advantage of measuring sequence diagrams is that it is easier to follow the flow of successive method calls and flow of parameters than it is in class diagrams.

Table 1: The qualitative metrics

Property	**Mark**	**Comment**
Level of Detail in Class Diagram	0-6	0 = all wrong, 6 = all correct
Realism in Class Abstractions	0-6	0 = all wrong, 6 = all correct
Realism in Class Attributes	0-7	0 = all wrong, 6 = all correct
Realism in Class Methods	0-8	0 = all wrong, 6 = all correct
Realism in Sequence Diagrams	0-8	0 = all wrong, 6 = all correct
Class and Sequence Diagram Correspondence	0-6	0 = all wrong, 6 = all correct

Table 2: The quantitative metrics

Property	**Description**
NC	Total Number of Classes
NA	Total Number of Attributes
NM	Total Number of Methods
NAssoc	Total Number of Associations
NFC	Total Number of False Classes
NFA	Total Number of False Attributes
NFM	Total Number of False Methods
NFAssoc	Total Number of False Associations
NSC	Total Number of Superfluous Classes
NSA	Total Number of Superfluous Attributes
NSM	Total Number of Superfluous Methods
NSAssoc	Total Number of Superfluous Associations
Time	Time used to develop a design model

A measure of the correspondence between class and sequence diagrams was in our experiment used to assess to what extent the subjects were successful in following the guidelines for the two processes. For the use case driven process, the objects used in the sequence diagrams should be derived from the domain model, and the class diagram should contain exactly the methods found in the sequence diagrams. For the validation process, the objects used in the sequence diagrams should be derived from the class diagram, and the class diagram should contain complementing methods to those found in the sequence diagrams. For both processes, the direction of method calls between objects in the sequence diagrams should be consistent with the way the methods are defined in the class diagrams.

Table 3 summarizes the qualitative metrics described above. The score of each metric is made according to a checklist[1] with yes/no questions for each property. The marking scales match the number of questions (between 6 and 9) in the checklists.

4.2 Quantitative Metrics

Many design metrics for object-oriented code have been proposed, but not all are applicable to high-level design [23]. The metrics suggested in [10] are developed for UML class diagrams and have been empirically validated [11]. To examine the size of the class diagrams, we use a subset of those metrics: total number of classes, attributes, methods and associations.

Errors in class diagrams were measured according to the number of classes, attributes, methods and associations that were wrong relative to the problem domain, *false*, or did not contribute to the implementation of the requirements, *superfluous*. *Time* measures the total time spent on constructing the design model. The quantitative metrics are summarized in Table 4.

[1]Appendix B.1

5 Evaluation of the two Processes

This section describes the experimental design, analysis, results and possible improvements to the experimental design. To the authors' knowledge, no empirical studies have been conducted to compare alternative ways of applying use case models in a development process. This study is therefore explorative; the goal of the evaluation was to investigate differences between the two processes.

5.1 Hypotheses

The aim of the experiment was to investigate differences in the design models resulting from applying two different development processes. The design models were evaluated relative to their suitability for code generation, the size of the class diagrams, the number of false and superfluous elements[2] in the class diagrams, and the time spent on creating them. We also attempted to assess differences in the subjects' ability to follow the guidelines of the processes. These aspects were separated into the following hypotheses:

$H1_0$: There is *no* difference in the level of detail in the class diagrams.
$H1_1$: There is *a* difference in the level of detail in the class diagrams.

$H2_0$: There is *no* difference in the realism of the design models.
$H2_1$: There is *a* difference in the realism of the design models.

$H3_0$: There is *no* difference in the correspondence between the class and sequence diagrams.
$H3_1$: There is *a* difference in the correspondence between the class and sequence diagrams.

$H4_0$: There is *no* difference in the size of the class diagrams.
$H4_1$: There is *a* difference in the size of the class diagrams.

$H5_0$: There is *no* difference in the number of false elements in the class diagrams.
$H5_1$: There is *a* difference in the number of false elements in the class diagrams.

$H6_0$: There is *no* difference in the number of superfluous elements in the class diagrams.
$H6_1$: There is *a* difference in the number of superfluous elements in the class diagrams.

[2]The terms false and superfluous elements are used as abbreviations for false and superfluous classes, attributes, methods, and associations.

$H7_0$: There is *no* difference in time spent creating the design models.
$H7_1$: There is *a* difference in time spent creating the design models.

5.2 Experimental Design

5.2.1 Subjects

The subjects were 26 undergraduate students following a course in software engineering. Half of the subjects received guidelines for the use case driven process whereas the other half received guidelines for the validation process. The experiment was voluntary, and the subjects were paid for their participation. The subjects had learned the basics of UML and had constructed an object-oriented design as part of a compulsory assignment in the course.

5.2.2 Procedure of the Experiment

The experiment lasted for three hours. The subjects used pen and paper. They wrote down the exact time they started and finished each exercise. The experiment consisted of two parts. The first part contained three exercises guiding the subjects in developing a design model with a class diagram and three sequence diagrams, modelling three functional services of a library system. The second part was not included in our analysis, but was part of the experiment in order to make sure that all the subjects had enough to do for three hours. The subjects had no former training in neither of the two alternative development processes so detailed guidelines on were given. The second part gave equal guidelines for both processes.

5.2.3 Experimental Material

The task of the experiment was to construct a design model for three functional services of a library system. This case is described in many books on UML, for example [24]. It was chosen because it is a well-known domain and simple enough for students just introduced to UML. The subjects were given a use case model with the following use cases:

- Checking out an item.
- Checking in an item.
- Checking the status on an item.

The use cases were described using a template format based on those given in [7]. Those following the validation process also received a textual requirements document of the system. The guidelines were based on the descriptions of the use case driven process and the validation process described in [24] and presented in Tables 1 and 2. In order to provide complete design processes, some additions were made to the original descriptions based on the reading techniques for object-oriented design described in [26]. For the same purpose, the construction of a domain model was added to use case driven process. Because of the time constraints on the experiment, one step from the original descriptions was removed from each process respectively. In addition, the subjects were given a use case model instead of being asked to construct one. In the use case driven process the explicit definition of the use case scenarios through activity diagrams was excluded. Table 5 shows the detailed guidelines for the use case driven process used in the experiment.

Exercise 1: Domain model

1. Underline each noun phrase in the use case descriptions. Decide for each noun phrase if 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 if these concepts should be represented as attributes in a domain model instead. (Not all attributes are necessarily found this way.)

Exercise 2: Sequence Diagrams

1. Read the use case descriptions carefully to understand the functionality they represent.
2. Mark the verbs or sentences in the descriptions that represent actions performed by the systems' classes. Decide if these actions should be represented by one or more methods in the sequence diagram. (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. For each method in the sequence diagram:
   (a) If an object of class A receives a method call M, the class A should contain the method M in the class diagram.
   (b) If an object of class A calls a method of class B, there should be an association between the classes A and B.

Figure 5: Guidelines for the use case driven process

In the validation process the listing of responsibilities for each class was excluded. Figure 6 shows the detailed guidelines for the validation process.

Exercise 1: Class Diagram

1. Underline all noun phrases in the requirements document. Decide for each noun phrase if 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 if these concepts should be represented as attributes in the class diagram instead. (Not all attributes are necessarily found this way.)

Exercise 2: Sequence Diagrams

Same as for the use case driven process.

Exercise 3: Validation of the Class Diagram

1. For each method in the sequence diagram, draw a circle around it. If several methods together form a system service, treat them as one service.
2. For each method or service circled out:
   (a) Validate that the class that receives the method call contains the same or matching functionality.
   (b) 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.
   (c) If the validation in the previous steps failed, make the necessary updates in the class diagram.

Figure 6: Guidelines for the validation process

5.3 Analysis and Results

The design models were evaluated according to the analysis framework presented in Section 3. The Kruskal-Wallis statistical test was performed on the results. This test was chosen because the data distributions were non-normal. A p-value of 0.1 was chosen as the level of significance for all the tests, because of the explorative nature of the experiment.

5.3.1 Assessment of level of detail, Hypothesis H1

We neither expected nor found a difference in the level of detail of the class diagrams, and we can therefore not reject $H1_0$. In our opinion, it is unlikely that this aspect relates to the choice of development process.

5.3.2 Assessment of realism, Hypothesis H2

The test on realism in class methods, figure 7, showed a difference in favour of the validation process. We therefore reject $H2_0$ at the selected level of significance, and assume with a 90% probability that there is a difference in the realism of the design models constructed by the two processes. We believe that this result is due to the fact that the subjects mostly mapped the steps in the use case descriptions directly onto methods when creating sequence diagrams. This resulted in both misleading method names and flaws is the method composition to realize the use cases. Figure 8 shows an example of how the steps of the use case "Check out item" typically were mapped to a sequence diagram. Every method in the sequence diagram is named and derived from a "corresponding" step in the use case description. For example, the initiating method is named "retrieveMember". This is either an innapropriate method or the name is misleading, since retrieving the member is probably just one of the things an initiating method needs to do in order to create a new loan. The order of method calls is equivalent to the order the steps are presented in the description. A problem caused by this, is for example that the due date on the loan is to be set before the loan is created. Also, how objects are associated is left out or not clear from assessing the sequence diagram. For example, how the member instance is associated with the new loan instance is left out or speculative.

```
Process            N    Median    Ave Rank         Z
Use case driven   10     4,000         8,0     -2,15
Validation        11     6,000        13,8      2,15
Overall           21                  11,0

H = 4,61  DF = 1  P = 0,032
H = 4,84  DF = 1  P = 0,028 (adjusted for ties)
```

Figure 7: Realism in class methods

The use case driven process used the sequence diagrams to identify all methods needed in the class diagram, while the validation process used the sequence diagrams only to detect any necessary functionality overlooked in the first attempt at a class diagram. Therefore, the problems concerning the direct mapping of methods had a much larger impact on the design models constructed with the use case driven process.

The tests on realism in class abstractions, class attributes, and sequence diagrams did, however, not show any difference with regards to realism between the two processes. This may be because of the small problem domain used in this experiment.

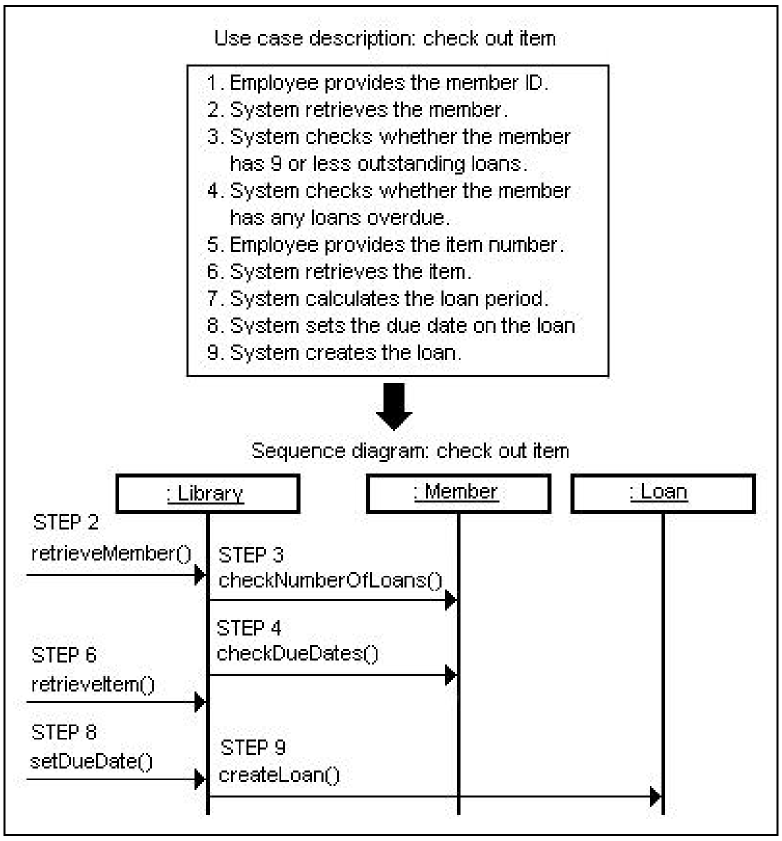

Use case description: check out item
1. Employee provides the member ID.
2. System retrieves the member.
3. System checks whether the member has 9 or less outstanding loans.
4. System checks whether the member has any loans overdue.
5. Employee provides the item number.
6. System retrieves the item.
7. System calculates the loan period.
8. System sets the due date on the loan
9. System creates the loan.
Sequence diagram: check out item
: Library
: Member
: Loan
STEP 2
retrieveMember()
STEP 3
checkNumberOfLoans()
STEP 4
checkDueDates()
STEP 6
retrieveItem()
STEP 8
setDueDate()
STEP 9
createLoan()

Figure 8: The mapping of methods from use case descriptions

5.3.3 Assessment of correspondence, Hypothesis H3

We expected a difference in correspondence in favour of the use case driven process as a strength of this process is claimed to be that it assures traceability between the diagrams in a design model, but the test, figure 9, did not show any difference, and we can not reject $H3_0$. However, there is a difference in median between the two processes, and the resulting p-values is close to showing a difference at the chosen level of significance. The subjects had little experience with creating and using sequence diagrams, and we believe this to be the main reason why those following the use case driven process did not achieve a better correspondence.

```
Process            N    Median    Ave Rank        Z
Use case driven   10     7,000        13,2     1,51
Validation        11     6,000         9,0    -1,51
Overall           21                  11,0

H = 2,29  DF = 1  P = 0,130
H = 2,41  DF = 1  P = 0,120 (adjusted for ties)
```

Figure 9: Class and sequence diagram correspondence

5.3.4 Assessment of size, Hypothesis H4

We expected a difference in the size of the class diagrams, with larger class diagrams produced with the validation process, because the use case driven process provides stricter guidance on how to identify classes, attributes, methods and associations. However, the tests regarding size did not show any difference, and we can not reject $H4_0$ at the chosen level of significance.

```
Process         N<  N>= Median Q3-Q1 ----------|---------|---------|-
Use case driven  3   7    6,00  1,25 (---------|--)
Validation       3   8    6,00  3,00 (---------|---------------------)
                                     ----------|---------|---------|-
                                              6,0       7,0       8,0
Overall median = 6,00

A 95,0% CI for median(Validation)-median(Use case driven):(-0,15;1,29)
```

Figure 10: Variance in number of classes

There is, however, a larger variance, figure 10, in the number of classes using the validation process as opposed to using the use case driven process. In our opinion this indicates that the use case driven process gives stricter guidance on how to identify classes. No difference in the variance in number of attributes, methods, or associations was found.

5.3.5 Assessment of superfluous elements, Hypotheses H5

We expected a difference in superfluous elements in the class diagrams created by the two processes for the same reason as we expected a difference in size. However, none of the tests on superfluous elements showed any difference, and we can not reject $H5_0$ at the level of significance. This may indicate that the guidelines on both processes prevented the subjects from including superfluous elements to an equal extent.

5.3.6 Assessment of false elements, Hypotheses H6

We did not expect a difference in false elements, due to the small problem domain and because the subjects had similar experience. However, the test on number of false classes, figure 11, showed a difference, and we reject $H6_0$ at the level of significance, and assume with a 90% probability that the use case driven process led to the inclusion of more false classes. The

```
Process            N     Median    Ave Rank          Z
Use case driven   10   1,00E+00        13,5       1,76
Validation        11   0,00E+00         8,7      -1,76
Overall           21                   11,0

H = 3,10  DF = 1  P = 0,078
H = 3,50  DF = 1  P = 0,061 (adjusted for ties)
```

Figure 11: Number of false classes

results of the tests on differences in size in Section 4.3.4 indicate that the use case driven process guided the subjects in a pattern when deriving classes. This result shows that the pattern resulted in the derivation of some classes that were wrong relative to the problem domain. For example the subjects following the use case driven process frequently derived a class "Employee" from the use case model. This is wrong relative to the problem domain, as an employee is a user and not an entity of the system.

5.3.7 Assessment of time spent, Hypothesis H7

A difference in time spent creating the design models was expected as exercises 1 and 3 were more comprehensive for the validation process. The tests on time on the individual exercises all showed differences. Those following the validation process spent most time on exercises 1 and 2, figure 12, while those following the use case driven process spent most time on exercise 3. The latter was surprising to us since exercise 3 was larger for the validation process. We believe that although the experiment consisted of two parts, many of the subjects did not proceed to the second part since they had been told that the first part was the most important. This influences the total time spent creating the design models, where no difference was shown. We can therefore not reject $H7_0$ at the selected level of significance.

```
Process            N    Median    Ave Rank        Z
Use case driven   10     78,00         5,7     3,73
Validation        11    108,00        15,8    -3,73
Overall           21                  11,0

H = 13,93  DF = 1  P = 0,000
H = 13,95  DF = 1  P = 0,000 (adjusted for ties)
```

Figure 12: Time spent on exercises 1 and 2

5.4 Improving the Experimental Design

This experiment was exploratory and there are several threats to the validity of our results. These threats and how they can be remedied are discussed below.

5.4.1 The Subjects

It is not clear how well student-based experiments generalize to software professionals [28]. Establishing the competence level of students compared to professional may be difficult based on number of credits earned and their amount of professional work experience [28].

The subjects were novices to modelling with UML. The measure of class and sequence diagram correspondence showed that neither process resulted in a perfect correspondence. This indicates that the subjects were not entirely able to follow the guidelines for the processes. Therefore, professional developers, more experienced subjects, or training in the processes in advance is an identified improvement. However, the guidelines were comprehensive and strict, which may compensate for lack of experience [13].

5.4.2 The Task

The task was small because of the time limit on the experiment. This resulted in small design models, leaving little room for variations in structure. Therefore, we were not able to investigate differences in external quality attributes such as maintainability and flexibility. The task of an experiment should be large enough to resemble real industrial projects in order to assess all factors relevant to industry [28]. Increasing the task and applying a more extensive measurement programme would thus represent an improvement.

5.4.3 The Environment

Pen and paper based experiments are not representative for real industrial situations. The experimental environment should resemble the environment in real software development projects if results are to be transferred to industry [28]. A great improvement would thus be to provide the subjects with computers and any necessary supporting technologies.

5.4.4 The Use Case Model

The subjects were provided with a use case model instead of letting them construct one for themselves. The analysis showed that the use case descriptions were important in the use case driven process because the subjects often assumed a direct mapping from the steps in the use case descriptions to methods in the class diagram. The results may therefore have been different if the order and textual presentation of the steps in the use case descriptions had been different, or if a use case model on another format had been used. The format used is a well-known format, but an improvement may be to let the subjects decide for themselves on which use case format to apply.

5.4.5 Interpretation of the two Approaches

The use case driven process and the validation process are not precisely defined in [24]. The guidelines for the two processes used in this experiment are therefore two particular interpretations of those processes. Due to the time limit on the experiment, we did not ask the subjects to construct any other models that might have served beneficial, such as activity diagrams recommended in [24].

5.4.6 The Guidelines

The exercise guidelines specified creating one sequence diagram for each use case. This was done because the use cases in this experiment are simple, but with more complex use cases it may be difficult to model all scenarios contained in one use case by one sequence diagram [21]. An improvement may therefore be to provide only an explanatory outline of the two approaches as in [24], letting the subjects decide for themselves which additional diagrams, such as activity and collaboration diagrams, to construct in the design models, and demand separate sequence diagrams for every scenario in the use cases.

5.4.7 The Measurement Framework

We used qualitative measurements in order to evaluate realism in the design models. This means the results of these measurements may have been affected by the experience, skill, and preference of the author. In order to claim internal validity, an identified improvement would be to only apply metrics that are empirically validated and may be collected automatically.

6 Conclusions and Future Work

Technologies are introduced to the software industry often without evidence on their usefulness and effects. However, in order to ensure a successful adaptation of technology in industry and be able to identify areas of improvement in a technology, knowledge and evidence of its effects, strenghts, and weaknesses is essential. Empirical research is identified as a means to explore, validate, and provide evidence on the effects of technology. In particular, experimentation is essential in order to build a *base* of knowledge and evidence.

In [4], an assessment and comparison of proposed software development processes and an investigation on the use of UML is in particularly great demand. We conducted an experiment to compare two processes with UML where the use case model is applied differently. One process was use case driven. In such processes a use case model serves as a basis for constructing a design model. The other process was a validation process where a use case model is applied in validating the design. The aim of the experiment was to investigate differences between the two processes with regards to the quality of the resulting design models defined in terms of suitability as a basis for automatic code generation as well as size and number of errors in the class diagrams.

The results show that the validation process led to design models better suited for automatic code generation because of greater realism in the method composition of the class diagrams. No significant difference regarding size was found, but there was a larger variance in the number of classes of the class diagrams constructed with the validation process than with the use case driven process. The use case driven process led to more erroneous classes relative to the problem domain than did the validation process. This indicates that the use case driven process gives more, but not necessarily better, guidance on how to identify classes and their attributes and methods. In our opinion, the results support the claims that a use case model is insufficient for deriving all necessary and appropriate classes, and may lead the developers to mistake requirements for design [22]. The results also indicate that it may be more appropriate to consider a use case as a behavioural feature of the system against which class diagrams can be validated, rather than consider a use case as having a state and methods from which the design can be derived. In summary, our study supports claimed weaknesses of use case driven processes and implies that following the validation process is the better choice in similar conditions to our experiment.

This study was exploratory. We are going to replicate this experiment with modifications discussed in section 4.3 incorporated. We also plan on conducing further studies to investigate how to best apply a use case model in an object-oriented design process. In particular we plan to interview project managers and experienced software developers on how use case models are actually applied in the software development process.

Bibliography

1. Anda, B., Sjøberg, D., Jørgensen, M. Quality and Understandability in Use Case Models. 13th European Conference on Object-Oriented Programming (ECOOP2001), June 18-22, Budapest, Hungary, LNCS 2072, Springer Verlag, pp. 402-428, 2001.

2. Arlow, J., Neustadt I. *UML and the Unified Process. Practical Object-Oriented Analysis and Design.* ISBN: 0-201-77060-1, Addison-Wesley, 2002.

3. Booch, G., Rumbaugh, J.,Jacobson, I. *The Unified Modeling Language User Guide.* ISBN: 0-201-57168-4, Addison-Wesley, 1999.

4. Briand, L., Arisholm, E., Counsell, S., Houdek, F., Thevenod-Fosse, P. Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of The Art and Future Directions. Empirical Software Emgineering: An International Journal, 2000.

5. Chidamber, S.R., Kemerer C.F. A Metrics Suite for Object-Oriented Design. IEEE Transactions on Software Engineering, Vol.20, No.6, pp.476-493, 1994.

6. Clark, C.B. The Effects of Software Process Maturity. USC Center for Software engineering PhD Dissertations, University of Southern California, 1997.

7. Cockburn, A. *Writing Effective Use Cases.* ISBN: 0-201-70225-8, Addison-Wesley, 2000.

8. Fowler, M., Scott, K. *UML Distilled Second Edition.* A Brief Guide to the Standard Object Modeling Language. ISBN: 0-201-65783-X, Addison-Wesley, 2000.

9. Fuggetta, A. Software Process: A Roadmap. In Anthony Finkelstein, ed. The future of Software Engineering. ACM Press, 2000.

10. Genero, M., Piattini, M., Calero, C. Early Measures for UML Class Diagrams. L'Objet., Vol.6, No. 4, Hermes Science Publications, pp. 489-515, 2000.

11. Genero, M., Piattini, M. Empirical Validation of Measures for Class Diagram Structural Complexity through Controlled Experiments. 5th International ECOOP workshop on Quantitative Approaches in Object-Oriented Software Engineering, June 2001.

12. Glinz, M. A Lightweight Approach to Consistency of Scenario and Class Models. Proceedings of the Fourth IEEE International Conference on Requirements Engineering. Schaumburg, Illinois, pp. 49-58, June 2000.

13. Hohmann, L. *Journey of the Software Professional. A sociology of Software Development.* ISBN: 0-13-236613-4, Prentice Hall, 1997.

14. Humphrey, W. *Managing the Software Process.* ISBN: 0-201-18095-2, Addison-Wesley, 1989.

15. Jacobson, I., Christerson, M., Jonsson P., Overgaard, G. *Object-Oriented Software Engineering: A Use Case Driven Approach.* ISBN: 0-201-54435-0, Addison-Wesley, 1992.

16. Jacobson, I., Booch, G., Rumbaugh, J. *The Unified Development Process.* ISBN: 0-201-57169-2, Addison-Wesley, 1999.

17. Juristo. N., Moreno, A.M. An Adaptation of Experimental Design to the Empirical Validation of Software Engineering Theories. 23nd Annual NASA Software Engineering Workshop. Maryland, EE.UU, 1998.

18. Kitchenham, B, Pickard, L., Phleeger, S.H. Case Studies for Method and Tool Evaluation. IEEE Computer, Vol.12, No.4, pp.52-62, July 1995.

19. Kitchenham, B. *Software Metrics: Measurement for Software Process Improvement.* ISBN: 1-85554-820-8, Blackwell, 1996.

20. Kösters, G., Six, H-W., Winter, M. Enhancing Activity Graphs to Bridge the Gap between Use Cases and Class models. Technical Report, Dept. of CS, University of Hagen, August 1999.

21. Kösters, G., Six, H-W., Winter, M. Coupling Use Cases and Class Models as a Means for Validation and Verification of Requirements Specifications. Requirements Engineering Journal, Vol. 6, Nr. 1, Springer, London, pp. 3-17, 2001.

22. Pooley P., Stevens R. *Using UML. Software Engineering with Objects and Components.* ISBN: 0-201-64860-1, Addison-Wesley, 2000.

23. Reissing, R. Towards a Model for Object Oriented Design Measurement. 5th International ECOOP workshop on Quantitative Approaches in Object-Oriented Software Engineering, June 2001.

24. Richter, C. *Designing Flexible Object-Oriented Systems with UML.* ISBN: 1-57870-098-1, Macmillan Technical Publishing, 1999.

25. Rosenberg, D. Scott, K. *Applying Use Case Driven Object Modeling with UML. An Annoted E-commerce Example.* ISBN: 0-201-73039-1, Addison-Welsey, 2001.

26. 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.

27. Sommerville, I. *Software Engineering, 6th Edition.* ISBN: 0-201-39815-X, Addison-Wesley, 2000.

28. Sjøberg, D., Anda, B., Arisholm, E., Bybå, T., Jørgensen, M., Karahasanovic, A., Koren, E.F., Vokac, M. Conducting Realistic Experiments in Software Engineering. Accepted for publication at the 2002 International Symposium on Empirical Software Engineering, Japan, October 2002.

29. Tichy, W.F. Should Computer Scientists Experiment More? 16 Reason to Avoid Experimentation. IEEE Computer, Vol.31, No.5, pp.32-40, May 1998.

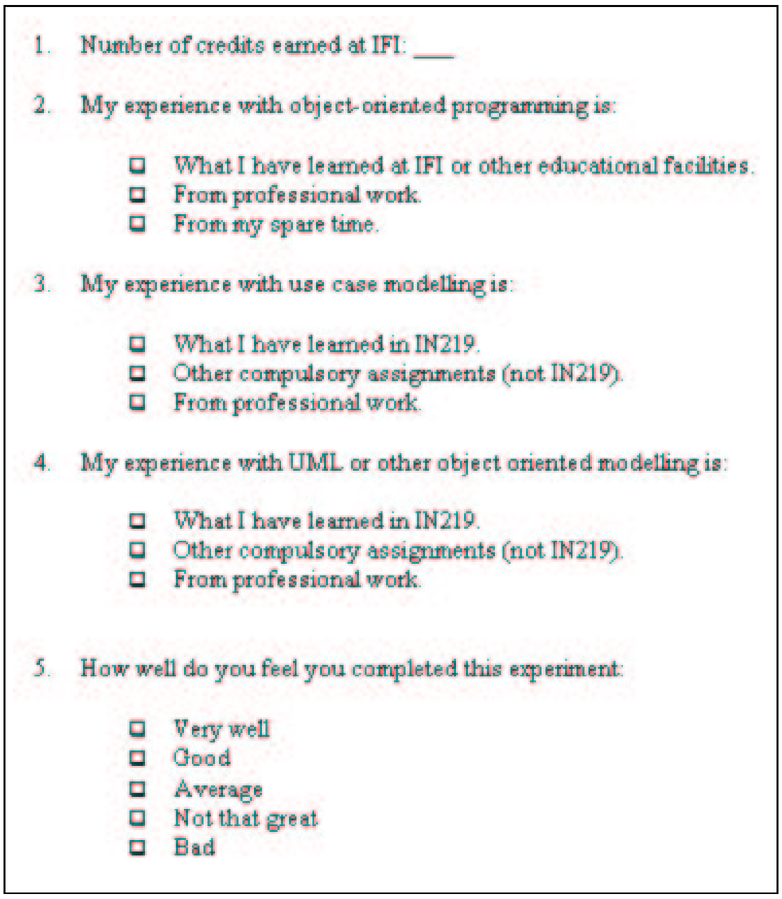
30. Zahran, S. *Software Process Improvement. Practical Guidelines for Business Success.* ISBN: 0-201-17782-X, Addison-Wesley, 1998.

31. Zelkowitz, M.V., Wallace, D.R. Experimental Models for Validating Technology. IEEE Computer, Vol 31, No.5, pp.23-31, May 1998

32. The UML meta-model, version 1.3. www.omg.org, 2001.

A Experimental Design

A.1 Background Information Qustionnaire



1. Number of credits earned at IFI: ____

2. My experience with object-oriented programming is:

   - ☐ What I have learned at IFI or other educational facilities.
   - ☐ From professional work.
   - ☐ From my spare time.

3. My experience with use case modelling is:

   - ☐ What I have learned in IN219.
   - ☐ Other compulsory assignments (not IN219).
   - ☐ From professional work.

4. My experience with UML or other object oriented modelling is:

   - ☐ What I have learned in IN219.
   - ☐ Other compulsory assignments (not IN219).
   - ☐ From professional work.

5. How well do you feel you completed this experiment:

   - ☐ Very well
   - ☐ Good
   - ☐ Average
   - ☐ Not that great
   - ☐ Bad

Figure 13: the background information qustionnaire

A.2 Use Case Model

A.2.1 Use case diagram

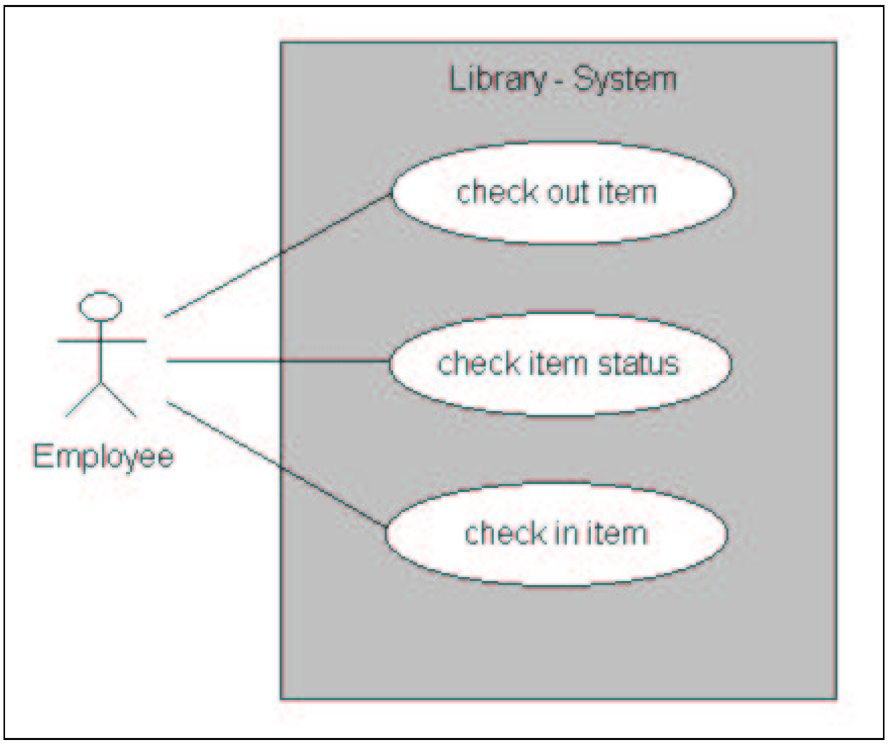
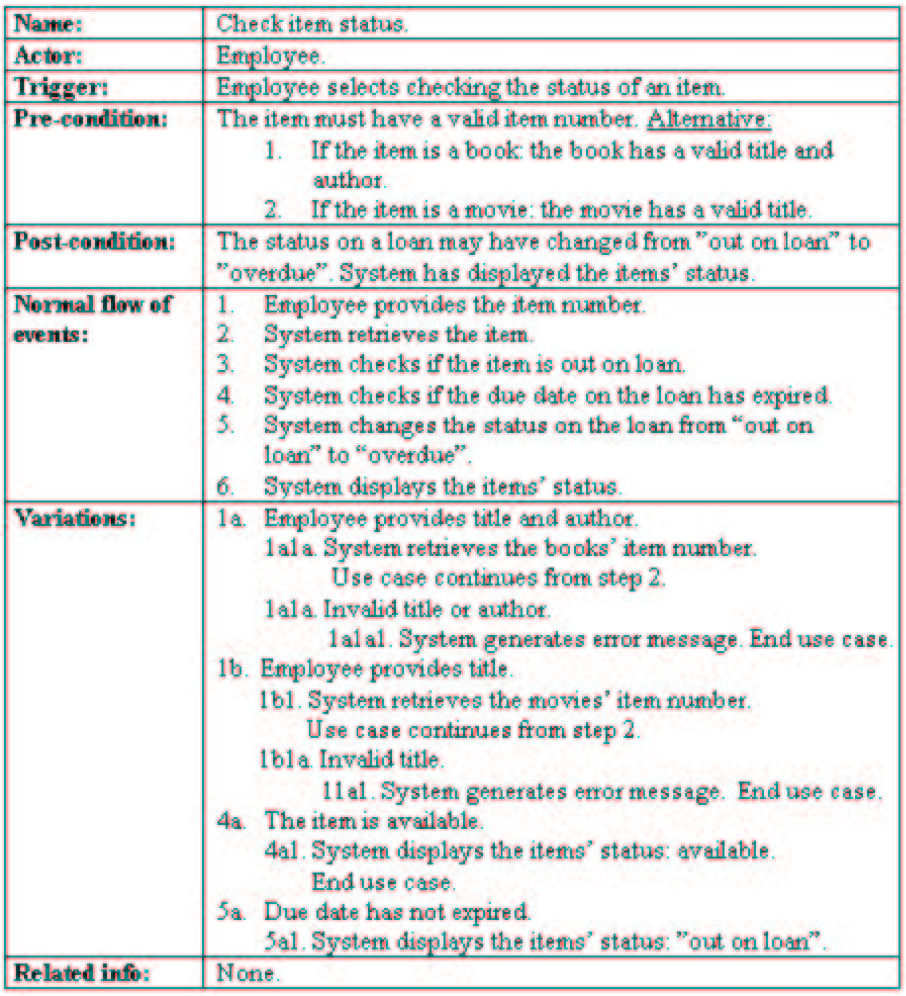


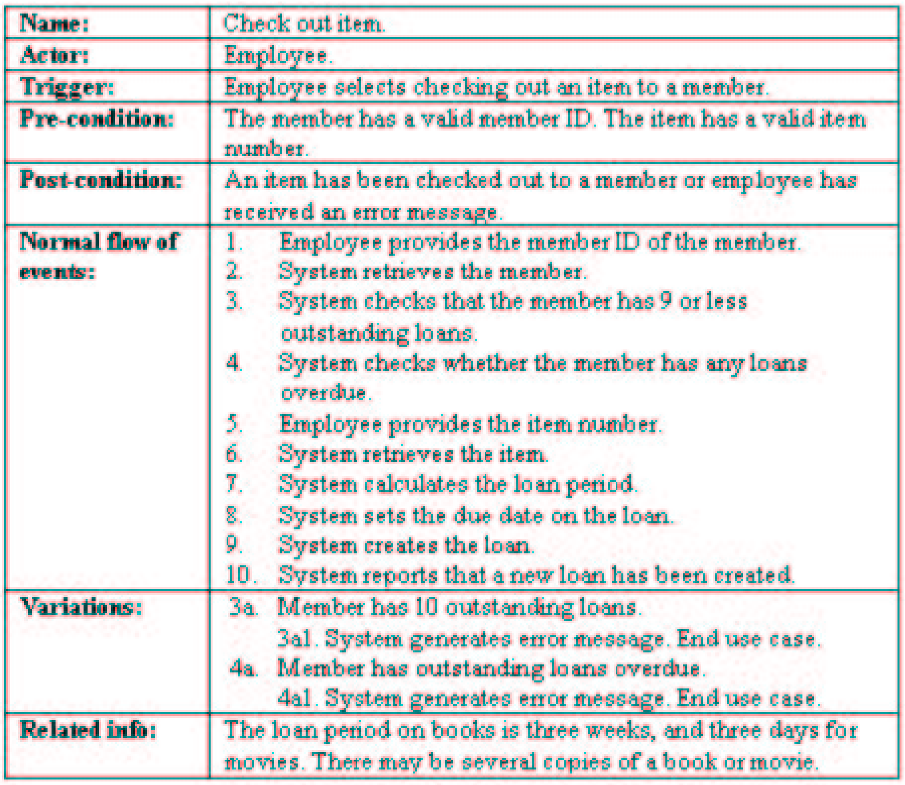
Figure 14: The use case diagram

A.2.2 Use case descriptions



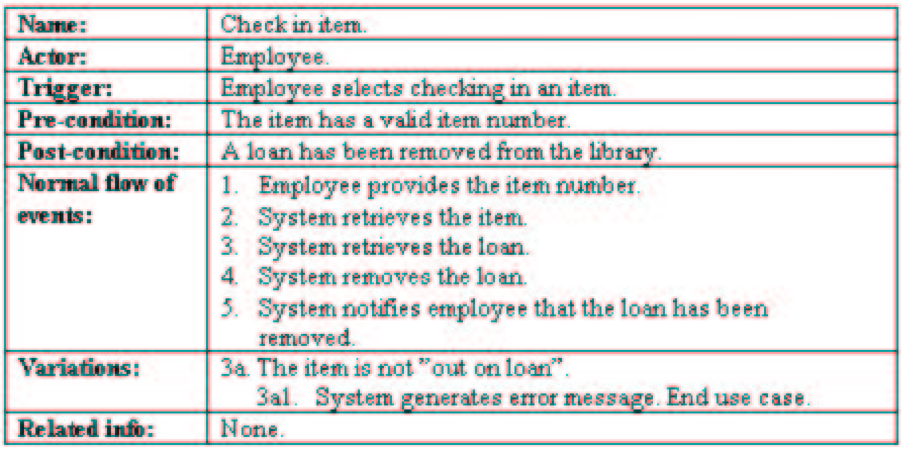
Name:	Check item status.
Actor:	Employee.
Trigger:	Employee selects checking the status of an item.
Pre-condition:	The item must have a valid item number. Alternative: 1. If the item is a book: the book has a valid title and author. 2. If the item is a movie: the movie has a valid title.
Post-condition:	The status on a loan may have changed from "out on loan" to "overdue". System has displayed the items' status.
Normal flow of events:	1. Employee provides the item number. 2. System retrieves the item. 3. System checks if the item is out on loan. 4. System checks if the due date on the loan has expired. 5. System changes the status on the loan from "out on loan" to "overdue". 6. System displays the items' status.
Variations:	1a. Employee provides title and author. 1a1a. System retrieves the books' item number. Use case continues from step 2. 1a1a. Invalid title or author. 1a1a1. System generates error message. End use case. 1b. Employee provides title. 1b1. System retrieves the movies' item number. Use case continues from step 2. 1b1a. Invalid title. 11a1. System generates error message. End use case. 4a. The item is available. 4a1. System displays the items' status: available. End use case. 5a. Due date has not expired. 5a1. System displays the items' status: "out on loan".
Related info:	None.

Figure 15: Use case 1: check out item



Name:	Check out item.
Actor:	Employee.
Trigger:	Employee selects checking out an item to a member.
Pre-condition:	The member has a valid member ID. The item has a valid item number.
Post-condition:	An item has been checked out to a member or employee has received an error message.
Normal flow of events:	1. Employee provides the member ID of the member. 2. System retrieves the member. 3. System checks that the member has 9 or less outstanding loans. 4. System checks whether the member has any loans overdue. 5. Employee provides the item number. 6. System retrieves the item. 7. System calculates the loan period. 8. System sets the due date on the loan. 9. System creates the loan. 10. System reports that a new loan has been created.
Variations:	3a. Member has 10 outstanding loans. 3a1. System generates error message. End use case. 4a. Member has outstanding loans overdue. 4a1. System generates error message. End use case.
Related info:	The loan period on books is three weeks, and three days for movies. There may be several copies of a book or movie.

Figure 16: Use case 2: check item status



Name:	Check in item.
Actor:	Employee.
Trigger:	Employee selects checking in an item.
Pre-condition:	The item has a valid item number.
Post-condition:	A loan has been removed from the library.
Normal flow of events:	1. Employee provides the item number. 2. System retrieves the item. 3. System retrieves the loan. 4. System removes the loan. 5. System notifies employee that the loan has been removed.
Variations:	3a. The item is not "out on loan". 3a1. System generates error message. End use case.
Related info:	None.

Figure 17: Use case 3: check in item

A.3 Requirements Document

You are going to develop a partial design for a library system, handling the borrowing and returning of books and movies. Only library members may borrow books and movies from the library, and the system must keep track of which books and movies are currently lent out to which members.

Each book and movie has a unique item number that the system uses to identify that particular book or movie. An employee can find the status on a book or movie by providing the system with the item number. Alternatively, the employee can find the status on a book by providing the system with its title and author, and status on a movie by its title only. The system must give the employee appropriate error messages if invalid data is provided. The books and movies can either be in the library (available for loan), lent out, or lent out overdue. By overdue means the member has failed to return the movie or book within its due date.

There may exist several copies of a book or movie.

When a member borrows an item (book or movie), its status changes from available to out on loan. When an item is returned, its status must change from either out on loan or out on loan overdue to available. The only issue in consideration is that you must ensure that when the status on an item is checked, the correct status is provided by the system, and that the members fulfil the conditions for borrowing items.

Each member has a member card, with a unique member id. When a member borrows an item, the employee must provide the system with the member id and the item number. If the member is currently borrowing less than 10 items, and has no current loans that are overdue, the conditions for borrowing items are fulfilled, and the loan may take place. If the conditions aren't met, the system must provide the employee with an appropriate error message. A member may borrow a book for three weeks, and a movie for three days.

NOTE! You are not to take into consideration any other functionality than the specified requirements in this document when developing your design.

A.4 Assignment Description for the Use Case Driven Process

The assignment is to develop a partial design for a library system. In this experiment, you are to follow a use case driven approach design. You need not to be familiar with this approach, as explicit guidelines are provided, but hopefully you will have learned something about use case driven design when you are finished. The problem domain is limited to a smaller area in order for you to finish in time.

When creating the class diagrams, you are going to take on an implementation perspective. This means that the level of detail in the class diagram should be at such that it is suited for generating code.

Part 1 is the most important part of the assignment. Remember to write down the exact time you start and finish each exercise.

Exercise 1: Domain model
You will find a use case model attached to this document. Read the descriptions carefully and create a domain model for a system that realizes these and only these requirements. Follow the guidelines below.

NOTE! You are not to detect methods in this exercise. In addition to entities, the domain model should consist of attributes, and any obvious associations.

Attributes should be specified with:

- Type (int, string, etc.).
- Visibility (public, private, protected).

Associations should be specified with:

- Cardinality on ends (1,*, 1..*, etc.)

- Navigation

Guidelines:

1. Underline every noun in the use case descriptions (wherever they may be). Decide for each whether or not they are concepts that should be represented by an entity or class.
2. For those nouns not representing classes, decide if they are instead concepts that should represented as attributes of a class. (Not all attributes are identified this way).

Exercise 2: Sequence diagrams
Study the attached use case model and create a sequence diagram for each use case. The sequence diagrams should contain only methods derived from the use case descriptions, and the objects from the domain model from exercise 1.

Guidelines:

1. Study the use cases and their descriptions carefully.
2. Underline the verbs or sentences describing an action. Decide if these are actions that should be represented with methods in the sequence diagrams.

Note! It is important that your sequence diagrams contain all methods needed for realizing the use cases. The guidelines above do not guarantee that all such methods are identified.

Exercise 3: Class diagram
If you completed exercise 2 correctly, this exercise will simply be to fill in the methods and associations identified in exercise 2 into a class diagram., based on the domain model from exercise 1. Create a complete class diagram for a system that realizes the requirements of the use case model by following the guidelines below.

Attributes should be specified with:

- Type (int, string, etc.).
- Visibility (public, private, protected).

Methods should be specified with:

- Return type (void, int, etc.).
- A list of parameters and their type.

- Visibility (public, private, protected).

Associations should be specified with:

- Cardinality on ends (1,*, 1..*, etc.)
- Navigation

Guidelines:

1. For every method call in each sequence diagram:
    (a) If an object of class A receives a method call M, the class A should contain M as a method.
    (b) If an object of class A receives a method call M from an object of class B, there should be an association between classes A and B.

A.5 Assignment Description for the Responsibility-driven Process

The assignment is to develop a partial design for a library system. A requirements document is attached to this paper. In this assignment, you are to follow a responsibility-driven approach design. You need not to be familiar with this approach, as explicit guidelines are provided, but hopefully you will have learned something about responsibility-driven design when you are finished.

When creating the class diagrams, you are going to take on an implementation perspective. This means that the level of detail in the class diagram should be at such that it is suited for a generating code.

Part 1 is the most important part of the assignment. Remember to write down the exact time you start and finish each exercise.

Exercise 1: Class diagram
You will find a written requirements document attached to this document. Study this carefully and create a complete class diagram for a system that realizes these requirements. Follow the guidelines specified below.

NOTE! Do not study the use case model before completing this exercise.

Attributes should be specified with:

- Type (int, string, etc.).
- Visibility (public, private, protected).

Methods should be specified with:

- Return type (void, int, etc.).
- A list of parameters and their type.
- Visibility (public, private, protected).

Associations should be specified with:

- Cardinality on ends (1,*, 1..*, etc.)
- Navigation

Guidelines:

1. Underline every noun in the requirements document. Decide for each whether or not they are concepts that should be represented by a class in the class diagram.
2. For those nouns not representing classes, decide if they are instead concepts that should represented as attributes of a class. (Not all attributes are identified this way).
3. Mark all verbs or sentences describing an action in the requirements document. Decide if these are actions that should be represented with methods in the class diagram. (Not all methods are identified this way).

NOTE! It is important that your class diagram contains all methods needed for realizing the requirements. The guidelines above do not guarantee that all such methods are identified.

Exercise 2: Sequence diagrams
Study the attached use case model and create a sequence diagram for each use case. The sequence diagrams should contain only methods derived from the use case descriptions, and the objects from the class diagram from exercise 1.

Guidelines:

1. Study the use cases and their descriptions carefully.
2. Underline the verbs or sentences describing an action. Decide if these are actions that should be represented with methods in the sequence diagrams.

NOTE! It is important that your sequence diagrams contain all methods needed for realizing the use cases. The guidelines above do not guarantee that all such methods are identified.

Exercise 3: Validation of the class diagram
In this exercise you are going to use the sequence diagrams from exercise 2 as means to validate that the class diagram from exercise 1 realizes the requirements. The names and composition of methods in the class and sequence diagrams need not to be identical, meaning the method names, the order of method calls, and number of method calls will probably be different. It is important that the methods in the class diagram support the functionality of the sequence diagrams.

If the validation fails, you must create a revised class diagram that realizes the use cases. To save time, you only need to draw those classes where changes are made in the revised class diagram. Write "the same" or similar on classes that are left unchanged. Do not make direct changes on the class diagram from exercise 1 by erasing etc. If you find that no changes are necessary, leave this exercise blank.

NOTE! It is important that you do not make changes only because you feel you have detected a more elegant way of structuring the class diagram. Only if the validation process detects flaws or errors should you introduce changes.

Guidelines:

1. Consider every method of the sequence diagram one by one. Draw a ring around the method. If one ore more methods together makes up a system service you may draw a ring around all of them an consider them as one system service.
2. For every method or system service:
   (a) Check if the class that receives the method call or service request has corresponding behaviour defined in form of one ore more methods.
   (b) If an object A receives a method call from an object B there should be an association between those two classes in the class diagram. (If your class diagram contains any hierarchies remember that it may be necessary to follow the hierarchy upwards when evaluating this).
3. If you found disagreements in the previous step, do the necessary modifications on your class diagram.

A.6 Part Two of the Experiment

Exercise 1: State and behaviour
In this exercise you are going to validate that the classes with a defined

behaviour, also has the necessary state to perform this behaviour. More specifically, if a class has a defined method, this class should also contain the necessary attributes to execute the method.

It is important that you carefully think through what information the classes need to execute their methods.

Guidelines:

1. For each method in each class: check whether or not the class has (access to) the attributes needed to execute the method.
2. Draw a new class diagram with the modifications if you found one or more classes lacked necessary information. To save time, you only need to draw those classes where changes are made. Write "the same" or similar on classes that are left unchanged.

Exercise 2: Extending the design
Imagine that you have generated code from your class diagram. This is far from a complete solution to such a system, so now you need to add further functionality.

The following is a prioritised list of new functionality to be added to the design:

1. To add and remove members. The system needs to know the name, date of birth, address, and phone number of every member.
2. To add new items. The system needs to know if it is a book or movie and the number of copies of the item. If the item is a movie, the system needs to know its title and genre, if it is a book the system needs to know its author in addition.
3. If a member fails to return an item within its due date, the system needs to calculate the fee upon return. The library management has decided on a fee of $3 pr day over due, but it must be possible to change this fee. The fee should be evaluated upon return, and stored in the system. It must be possible for an employee to delete this fee if the member pays cash upon return.
4. An employee must be able to change the personalia of members.

Guidelines:

1. Identify the new use case, and describe it using the mal outlined in IN219.
2. Draw a sequence diagram for the use case.

3. Draw a CRC card for the updated classes. You may draw a class diagram instead, but do not make direct modifications on the class diagrams from the previous part by erasing etc.

NOTE! Methods and attributes should be specified as in part 1.

B Analysis

B.1 Checklist

Realism in Class Diagrams:

Whether or not the class diagram contains a class representing:

1. The library.
2. The library members.
3. Lendable items (article).
4. Movies and books (subclasses of lendable item or article).
5. Member loans.
6. Lendables (copies) with an association to the abstract article.

Whether or not the classes in the class diagram contains the attributes:

1. Member ID number.
2. A lendable or article ID number.
3. Title for lendables.
4. Author for books.
5. Loan periods on lendables.
6. Status on either lendables or loans.
7. Date or due date on loans.

Whether or not the class diagram contains methods for:

1. Checking out a lendable to a member.
2. Checking the constraints on the number of outstanding loans a member has, or any loans overdue before checking out a lendable to the member.
3. Setting the date or due date on a new registered loan.
4. Checking in a lendable from a member.
5. Checking the current status on a lendable.
6. Changing the status of a lendable to out on loan or similar when a lendable is checked out. (If a class "loan" with a due date attribute and with an association to/from the lendable exists and lendables does not have a status attribute, this is not necessary, and will be evaluated positively).

7. Changing the status of a lendable to available for loan or similar when a lendable is checked in. (If a class "loan" with a due date attribute and with an associations to/from the lendable exists and lendables does not have a status attribute, this is not necessary, and will be evaluated positively).

8. Changing the status on a lendable or loan to overdue or similar if a lendable is not returned within its due date. (If a class "loan" with a due date attribute and with associations to/from the member and lendable exists, and lendables does not have a status attribute, this is not necessary, and will be evaluated positively).

Realism in Sequence Diagrams:

This evaluation is equal to the one for methods in realism in class diagrams. The three sequence diagrams are evaluated together.

Correspondence between Sequence and Class Diagrams:

For each sequence diagram:

1. Can the object classes (instances) be found in the class diagram?
2. Can the methods be found in the class diagram?
3. Are the right objects receiving the right method calls according to the class diagram?

Level of Detail in Class Diagram:

Whether or not the class diagram is depicted with:

1. Visibility on attributes.
2. Visibility on methods.
3. Type on attributes.
4. Type on methods.
5. Type on parameters.
6. Cardinality on association ends.

B.2 Kruskal-Wallis Test Results

```
----------------------------------------------------
Level of Detail in Class Diagrams

Process            N    Median    Ave Rank         Z
Use case driven   10     5,000        12,2      0,81
Validation        11     4,000        10,0     -0,81
Overall           21                  11,0

H = 0,66  DF = 1  P = 0,418
H = 0,72  DF = 1  P = 0,396 (adjusted for ties)
----------------------------------------------------
Number of Classes

Process            N    Median    Ave Rank         Z
Use case driven   10     6,000        10,5     -0,39
Validation        11     6,000        11,5      0,39
Overall           21                  11,0

H = 0,15  DF = 1  P = 0,699
H = 0,17  DF = 1  P = 0,680 (adjusted for ties)
----------------------------------------------------
Number of Attributes

Process            N    Median    Ave Rank         Z
Use case driven   10     17,00        11,5      0,35
Validation        11     14,00        10,5     -0,35
Overall           21                  11,0

H = 0,12  DF = 1  P = 0,725
H = 0,12  DF = 1  P = 0,724 (adjusted for ties)
----------------------------------------------------
Number of Methods

Process            N    Median    Ave Rank         Z
Use case driven   10     15,50         9,6     -0,99
Validation        11     16,00        12,3      0,99
Overall           21                  11,0

H = 0,97  DF = 1  P = 0,324
H = 0,98  DF = 1  P = 0,323 (adjusted for ties)
----------------------------------------------------
Number of Associations

Process            N    Median    Ave Rank         Z
Use case driven   10     5,000        11,4      0,32
Validation        11     5,000        10,6     -0,32
Overall           21                  11,0
```

```
H = 0,10  DF = 1  P = 0,751
H = 0,10  DF = 1  P = 0,747 (adjusted for ties)
----------------------------------------------------
Number of False Classes

Process            N    Median    Ave Rank         Z
Use case driven   10  1,00E+00        13,5      1,76
Validation        11  0,00E+00         8,7     -1,76
Overall           21                  11,0

H = 3,10  DF = 1  P = 0,078
H = 3,50  DF = 1  P = 0,061 (adjusted for ties)
----------------------------------------------------
Number of False Attributes

Process            N    Median    Ave Rank         Z
Use case driven   10  5,00E-01        11,3      0,21
Validati          11  0,00E+00        10,7     -0,21
Overall           21                  11,0

H = 0,04  DF = 1  P = 0,833
H = 0,06  DF = 1  P = 0,814 (adjusted for ties)
----------------------------------------------------
Number of False Methods

Process            N    Median    Ave Rank         Z
Use case driven   10  0,00E+00        10,4     -0,42
Validati          11  0,00E+00        11,5      0,42
Overall           21                  11,0

H = 0,18  DF = 1  P = 0,673
H = 0,22  DF = 1  P = 0,637 (adjusted for ties)
----------------------------------------------------
Number of False Associations

Process            N    Median    Ave Rank         Z
Use case driven   10  0,00E+00        11,0      0,00
Validati          11  0,00E+00        11,0      0,00
Overall           21                  11,0

H = 0,00  DF = 1  P = 1,000
H = 0,00  DF = 1  P = 1,000 (adjusted for ties)
----------------------------------------------------
Number of Superfluous Classes

Process            N    Median    Ave Rank         Z
Use case driven   10  0,00E+00        10,5     -0,35
Validati          11  0,00E+00        11,5      0,35
```

```
Overall           21                  11,0

H = 0,12  DF = 1  P = 0,725
H = 0,91  DF = 1  P = 0,340 (adjusted for ties)
---------------------------------------------------
Number of Superfluous Attributes

Process           N    Median    Ave Rank         Z
Use case driven  10     6,500        12,6      1,09
Validati         11     2,000         9,6     -1,09
Overall          21                  11,0

H = 1,19  DF = 1  P = 0,275
H = 1,23  DF = 1  P = 0,268 (adjusted for ties)
---------------------------------------------------
Number of Superfluous Methods

Process           N    Median    Ave Rank         Z
Use case driven  10  0,00E+00        10,1     -0,63
Validati         11  0,00E+00        11,8      0,63
Overall          21                  11,0

H = 0,40  DF = 1  P = 0,526
H = 0,63  DF = 1  P = 0,427 (adjusted for ties)
---------------------------------------------------
Number of Superfluous Associations

Process            N    Median    Ave Rank         Z
Use case driven   10  1,00E+00        12,9      1,30
Validati          11  0,00E+00         9,3     -1,30
Overall           21                  11,0

H = 1,70  DF = 1  P = 0,193
H = 1,94  DF = 1  P = 0,163 (adjusted for ties)
---------------------------------------------------
Realism in Classes

Process            N    Median    Ave Rank         Z
Use case driven   10     4,000         9,8     -0,81
Validation        11     5,000        12,0      0,81
Overall           21                  11,0

H = 0,66  DF = 1  P = 0,418
H = 0,76  DF = 1  P = 0,384 (adjusted for ties)
---------------------------------------------------
Realism in Attributes

Process            N    Median    Ave Rank         Z
Use case driven   10     4,500         8,9     -1,48
```

```
Validation        11     5,000        12,9      1,48
Overall           21                  11,0

H = 2,19  DF = 1  P = 0,139
H = 2,58  DF = 1  P = 0,108 (adjusted for ties)
----------------------------------------------------
Realism in Methods

Process            N    Median    Ave Rank         Z
Use case driven   10     4,000         8,0     -2,15
Validation        11     6,000        13,8      2,15
Overall           21                  11,0

H = 4,61  DF = 1  P = 0,032
H = 4,84  DF = 1  P = 0,028 (adjusted for ties)
----------------------------------------------------
Realism in Sequence Diagrams

Process            N    Median    Ave Rank         Z
Use case driven   10     4,500         8,9     -1,48
Validation        11     6,000        12,9      1,48
Overall           21                  11,0

H = 2,19  DF = 1  P = 0,139
H = 2,25  DF = 1  P = 0,133 (adjusted for ties)
----------------------------------------------------
Correspondence:  Class - sequence diagrams

Process            N    Median    Ave Rank         Z
Use case driven   10     7,000        13,2      1,51
Validation        11     6,000         9,0     -1,51
Overall           21                  11,0

H = 2,29  DF = 1  P = 0,130
H = 2,41  DF = 1  P = 0,120 (adjusted for ties)
----------------------------------------------------
Time Exercise 1

Process            N    Median    Ave Rank         Z
Use case driven   10     35,00         6,7     -3,06
Validation        11     56,00        15,0      3,06
Overall           21                  11,0

H = 9,38  DF = 1  P = 0,002
H = 9,43  DF = 1  P = 0,002 (adjusted for ties)
----------------------------------------------------
Time Exercise 2

Process            N    Median    Ave Rank         Z
```

```
Use case driven   10     39,00         7,8     -2,25
Validation        11     48,00        13,9      2,25
Overall    21                  11,0

H = 5,08  DF = 1  P = 0,024
H = 5,10  DF = 1  P = 0,024 (adjusted for ties)
----------------------------------------------------
Time Exercise 3

Process            N    Median    Ave Rank         Z
Use case driven   10     28,50        15,3      2,99
Validation        11     19,00         7,1     -2,99
Overall           21                  11,0

H = 8,96  DF = 1  P = 0,003
H = 9,07  DF = 1  P = 0,003 (adjusted for ties)
----------------------------------------------------
```