A Case Study on the Application of UML in Legacy Development

Bente Anda
Simula Research Laboratory
P.O. Box 134, NO-1325 Lysaker
Norway
+47 67828306
bentea@simula.no

Kai Hansen
ABB Corporate Research Center
P.O. Box 90, NO-1361 Billingstad
Norway
+47 22874638
kai.hansen@no.abb.com

ABSTRACT
Model-driven development with UML is becoming a de facto standard in industry, but although much of today's software development is about enhancing existing systems, there is no well-defined process for model-driven development in the context of legacy systems. To ensure the relevance of research on model-driven development with UML, there is a need for studies of actual use of UML in software development companies. As part of a software process initiative, we conducted a case study in a large development project where some of the development teams enhanced existing components, while other teams developed software from scratch. The results from this case study showed that those who applied UML in modelling and enhancing legacy software experienced more challenges and fewer benefits from the use of UML than did the developers who modelled and developed new software. Overall our results show a need for better methodological support on applying UML in legacy development.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *object-oriented design methods.*

General Terms: Documentation, Design.

Keywords
Model-driven development, UML, Legacy software, Embedded Software

1. INTRODUCTION
Model-driven development views software development as a series of modelling steps, each of them refining the models of the previous step. Modelling is seen as a way to handle the growing complexity of software development by helping engineers work at higher levels of abstraction. Moreover, model-driven development is supported by the Unified Modeling Language (UML) [5], an evolving standard that is now in widespread use across industry. Nevertheless, there is little empirical evidence supporting the claim that UML is an effective approach to modeling software system, and the empirical studies that have been conducted on the use of UML have mostly been conducted with students in academic settings. The widespread use of UML means that there is a need to understand challenges and benefits of applying UML-based development in different types of development projects. Such understanding should, among other things, be based on evidence from actual use of, and the consequences of using, UML in industry.

A large proportion of software development in industry is enhancement of existing systems. Thus model-driven development with UML must frequently be introduced in legacy development, to leverage previous, often significant investments. To derive the maximum benefits from using UML, model-driven development projects should therefore be able to take advantage of legacy code [15].

The company ABB, a large global company with more than 100000 employees in over 100 countries, decided to attempt to improve the company's software development projects through the use of model-driven development with UML. The aim was to improve, among others, communication between stakeholders in the projects, the design and documentation of the systems as well as the testing procedures. A UML-based development method was developed and applied in a large development project that comprised approximately 230 people. Much of the development in the project was about enhancing existing systems with new features, while some components were developed from scratch. Consequently, ABB experienced a need to understand the particularities of applying UML in legacy development. Hence, it was decided to evaluate whether the experience of applying UML-based development was positive from the perspective of the individual practitioners and project managers. It was expected that unless the experience was positive, the new technology would risk rejection despite its potential for yielding benefits.

The particularities of this development project in ABB allowed us to conduct a case study to compare the use of UML in legacy development with new development. For the purpose of this study we define legacy development as the development of a new software system based on one or more existing systems.

Data on challenges and benefits of the UML-based method was collected through interviews with, and questionnaires from, experienced project members. Overall results from the interviews on applying UML-based development are reported in [2]. This paper focuses on challenges in legacy development compared with development from scratch. Hereafter, we use the term legacy group for those who applied UML in legacy development and the term new-code group for those who developed from scratch.

The results show that the legacy group experienced more difficulties than did the new-code group in the construction of the diagrams. In particular, identifying and documenting use cases was much more difficult for the legacy group than for the new-code group. The use of UML diagrams also yielded fewer benefits in legacy development, although there were some positive effects in legacy development, in particular from applying sequence and class diagrams. The most positive benefits were obtained with respect to testing. We believe that these results confirm that there is potential for model-driven development with UML in legacy development, although the costs of introducing it are higher than in new development. Furthermore, the results from this study could be used as input to methodological support on constructing and applying UML diagrams in legacy development.

The remainder of this paper is organized as follows. Section 2 describes the research method. Section 3 describes the project that was the case for this study as well as the UML-based development method applied in the project. Section 4 describes the results. Section 5 describes related work. Section 6 concludes and suggests directions for future work.

2. RESEARCH METHOD

Case studies are important if we are to understand the actual practice of software development, and such understanding is an essential prerequisite to the primary aim of empirical software engineering research, which is to inform practice [14]. A major strength of the case study approach is that it allows the study of a phenomenon within its real-life context [16]. A software process improvement initiative with focus on UML-based development in the company ABB provided us with the opportunity to conduct a case study on the use of UML-based development in a large project.

The members of this project experienced particular problems with applying UML in legacy development [2], and we consequently decided to investigate the particular challenges related to such development compared with development from scratch. The project provided a very good opportunity for investigating the use of UML in actual practice because safety constraints imposed by the standard IEC61508 [9] required the developers to use a semi-formal development method based on the use of UML. Furthermore, some parts of the system were developed from scratch, while other parts were enhancements of legacy code with new functionality to satisfy market and safety requirements. We could thus exploit natural differences in the project, as recommended in the literature on conducting case studies [10], to investigate differences between applying UML-based development in legacy development and in new development under otherwise similar conditions.

The extensive professional experience of developers at ABB was collected through interviews and questionnaires. The collected data thus represents the developers' opinions on the effects of using UML. One might, of course, have liked more objective measures, but the scarce previous research on the topic provides no measures that can be applied in the evaluation of UML-based development, and it was outside the scope of this project to attempt to develop new ones.

Interviews were conducted with 16 members of the project. The interviews included open-ended questions about experience with the different UML diagrams, the comprehensibility of the diagrams, costs and benefits of UML-based development, and experience with the use of UML diagrams in maintenance. The complete interview guide and a description of the analysis of the interviews can be found in [2].

A questionnaire was then developed with the aim of 1) validating the results from the interviews on overall challenges and benefits and 2) investigating in greater detail the differences between development from scratch and legacy development. This latter is the focus of this paper. The items in the questionnaire closely follow ABB's UML-based method (Figure 1). One project member, who had previously been interviewed, was a pilot to test the ease of comprehension of the questions. There were two types of question: *yes/no questions* where the respondents would indicate the use of the different diagrams, and *propositions* regarding which they would give their opinion on a five-point Likert scale (the options were: Totally agree (=1), Partially agree (=2), Neither agree or disagree (=3), Partially disagree (=4) and Disagree (=5).

All together, 55 project members who had not been interviewed responded to the questionnaire. Hard copy of the questionnaires was handed out to each respondent by an MSc student. We believe that personal contact with respondents helped to ensure serious responses, but a few of the project members declined to respond because they had not worked much in detail with UML, while others declined because their managers placed their priority on finishing the project and worried that completing the questionnaire would result in the loss of valuable time. There were also some consultants who had finished their job and left the project before the questionnaires were handed out.

All respondents were used to provide feedback to the company and validate the interviews. For the purpose of this study we focused on developers who had either developed from scratch or developed using legacy code as a basis, and on the experience of those developers with the individual UML diagrams. We consequently omitted managers and reviewers who had not been directly involved in development. Two respondents who answered very few questions were removed from the analysis, leaving us with 28 respondents, 14 in each group. There was also a group of 14 developers who had developed both from scratch and based on legacy code. This group was omitted to ensure distinctly different groups with respect to the focus of this study. The fact that there were 14 developers in each group was a coincidence. The respondents indicated themselves, on the questionnaire, what kind of development they had been involved in. The questionnaire is shown in Appendix A.

As part of this case study, we also had several meetings with representatives of the development project, and we had access to all project documents. The second author of this paper is an employee of ABB and is thus acquainted with the development project under study. In the interpretation and discussion of the responses to the questionnaire, we use knowledge of the project obtained through these sources.

External validity is about establishing the domain to which a study's findings can be generalized. A common criticism regarding case studies is the impossibility of generalising results obtained from a single case. However, according to Yin [16], "case studies are generalisable to theoretical[1] propositions and not to populations or universes". That is to say, the results from this case study represent issues related to the use of UML in legacy development that were encountered in the project under study, and which we therefore expect would be encountered in other projects that are similar to this one, in particular with respect to application domain and qualifications of the team members. The results do not represent issues that will always occur when UML is applied in legacy development. Furthermore, this is not a study of the full potential of UML-based development, but a study of the practical use of UML-based development in a large, global company that actually uses UML in their software development projects. The level of experience and skill in UML is probably representative for most developers in industry today.

3. THE CASE STUDY CONTEXT

This section describes the development project and the UML-based method that was applied in the project.

3.1 The Development Project

The project developed a new version of a safety-critical system based on several existing systems. The system was to be installed at several locations, and each installation could program its own logic on top of the system delivered by ABB. The workforce comprised approximately 230 people, 100 of whom were involved in development with UML. Some of the developers and all the product managers were domain experts. ABB relies on having domain expertise in-house because they sell a complete product to its customers. Safety certifiers, UML experts, quality managers and peer developers (also with domain knowledge) reviewed the UML models at predefined gates in this development process. The developers were organized in teams, which consisted of 8-10 people on average. Each team would typically be responsible for one or several components from analysis to the finished code.

The existing systems consisted of 3-4 million lines of code and there were approximately 1000 requirements for this version, one third of which concerned satisfying safety requirements and the remainder of which consisted of requirements for new functionality from the product management. C and C++ were used in the software implementation. UML version 1.3 and Rational Rose was used for modelling. The tool Doxygen[2] was used for the reverse engineering of code. Much of the software was embedded. This software project was ABB's most ambitious project regarding quality assurance in that it followed the requirements of IEC 61508 [9]. It was these requirements that motivated the use of UML-based development.

Most of the developers on the projects had a great deal of experience with software and hardware development. Most of them had previously used SDL, but the company had previously not used a standardized method for the analysis, design and documentation. The majority received training in UML, the UML-based method and Rational Rose before the start of the project. Some also had experience with UML-based development from previous projects.

The UML-models in this project were large and complex, and there were many challenges related to obtaining high quality models (these are described in detail in [2]). However, the UML-models were reviewed for syntactic and semantic errors at several stages in their construction to ensure good quality of the final models.

3.2 The UML-based Development Method

The overall development method in ABB is an implementation of the V-model, and the projects follow a Gate Model with predefined gates. The UML-based method was developed internally. It was not based on any particular method for UML-based development, but those responsible for it had experience with development based on UML, and were familiar with basic literature on such development, for example [5,7,8].

Figure 1 gives an overview of the requirements analysis and analysis phases of the UML-based development. The use of activity and state diagrams was optional in the method, and such diagrams were therefore used by only a few of the developers, and hence were not a subject of study.

The method description recommended some iteration over the different phases and activities, in practice, however, the development mostly followed a waterfall model due to the gate model, but with some return to update previously developed models. Complete method descriptions are confidential.

In detailed design the high-level classes are realized with implementation class diagrams, and the classes are grouped into components.

The UML diagrams were recommended used as input to test cases. Use cases were input to functional test specifications, sequence diagrams were input to integration testing, and the class diagrams were input to unit testing. There was, however, no detailed method description regarding how the diagrams should be applied in testing.

There exists no well-defined process for the use of UML in legacy development. Consequently ABB's UML-based method was not specifically tailored to such development. Nevertheless, the UML-based method stated that existing code could be reverse engineered into classes, which would be included in top-down analysis and design of the system. New code that uses the existing code base should use the classes' interfaces when representing the existing code in sequence and class diagrams. In practice, however, it was considered too costly and consequently infeasible to completely reverse engineer the existing code, because of its size and because the existing code was not always designed according to object-oriented principles. At the outset of the project, it was not clear which parts of the legacy code should be completely reverse engineered into object-oriented code and which parts should be kept as they were, but wrapped to be used as objects in the new models. This decision was at the discretion of the different development teams, and was dependent on how much change was necessary to each component in order to satisfy the new requirements. Future versions of the system will further reverse engineer and improve the design of these components.

[1] There is no existing theory on the use of UML, but we expect that this one and other explorative studies could be used to formulate initial theories.

[2] http://www.stack.nl/~dimitri/doxygen/

1. Use case modelling
1.1 Identify and document actors
Actors are the system's external interfaces. Humans, timers, sensors or anything else that interacts with the system can be an actor. For a use case diagram in a subsystem, other (interacting) subsystems should also be defined as actors.
1.2.a Identify and document use cases
Use cases define the system seen from the actors' point of view. They capture the requirements and represent the different usage of the system. A use case is always initiated by an actor.
1.2.b Describe flow of events inside the use cases
Each use case has at least one normal flow of events. Then capture the exceptional flows of events for each use case. This is done in several iterations. (We investigated activities 1.2.a and 1.2.b together since the interviewees did not distinguish between the first identification of use cases and their textual description).
Templates, with basic and alternative flows of events, pre and post conditions, and extension points, were used for this purpose.
1.3. Group use cases and actors into subsystems
There should be strong cohesion within the subsystems and weak coupling between the subsystems.
(It is commonly recommended in UML development methods to identify high-level classes before the division into subsystems. ABB decided to do it this way because many of the actors and use cases described legacy code)
1.4 Refine the use cases and identify dependencies
If some use cases show common behaviour that can be extracted without disturbing the main functionality, it can be factored out as a separate use case and included into the diagrams using the <<include>> stereotype. If some use cases have behaviour that can be seen as additions to, or variations of, normal behaviour, such forms of behaviour can be factored out as separate use cases and included into the use cases using the <<extend>> stereotype.

2. Sequence diagrams
2.1 Create high-level sequence diagrams
High-level sequence diagrams model the textual description of each use case. The sequence diagrams should show the dynamics between the objects involved in the use case and the actors interfacing them, for both normal and exceptional flows of events.
2.2 Define interfaces between use cases in different subsystems
Define interfaces between interacting sequence diagrams of different subsystems.

3. Class diagrams
Identify high-level classes that describe the commonality between similar objects in the sequence diagrams and define the structure and behaviour for each object in the class. Assign objects to the correct classes. The interactions between the objects in the sequence diagrams help to identify the operations in the classes. The different messages will identify operations in the class of the receiving object. Find the information necessary to process each message in the sequence diagrams. This information will end up as attributes in the class of the receiving object. The class diagram should show associations between the classes. Finally, update sequence diagrams with correct class and operation names.

Figure 1. Brief description of the UML-based method

Since they were all developing one system, the overall architecture, as well as the non-functional requirements were, in principle, the same for all. Nevertheless, those who enhanced legacy code generally obtained a less object-oriented architecture within the components than did those who developed from scratch.

The method description was applied by both those developing from scratch and those enhancing the legacy code, but there were some differences between the actual development processes of the two groups:

1. Use cases: Both the new-code and the legacy group used requirements as input to constructing use cases for new functionality. The legacy group also documented the legacy code with use cases. They based the use cases on the existing code and their experiences with that code. Some of the legacy use cases were then enhanced with new functionality. Extending use cases were often used for adding new functionality to existing use cases.
2. Sequence diagrams: The new-code group made sequence diagrams based on the use cases. The legacy group used, in addition, the existing code, their experiences with that code and the reverse engineered classes.
3. Class diagrams: The new-code group identified class diagrams in parallel with developing sequence diagrams. The legacy group reverse engineered and manually improved the results of reverse engineering, also with the use of sequence diagrams.

4. RESULTS

This section reports the results with respect to ease of constructing the diagrams, and how they were used and utilised in the areas of improvements desired by ABB.

4.1 Ease of Constructing Diagrams

The respondents were asked how easy it had been to construct the different diagrams. The questionnaire closely followed the ABB method description with most focus on use case modelling. We also knew from the interviews that the project members had most opinions about use case modelling, since this technique was new for all of them and the starting point for the use of the UML based method. Consequently, the questionnaire included more questions on use case modelling than on the construction of sequence and class diagrams.

4.1.1 Use cases

Figure 2 shows boxplots of the responses to the questions 1.1– 1.4 for those developing from scratch, denoted new, and those enhancing legacy code, denoted legacy. The figure should be read as follows

- The median response is denoted by the cross circle.
- The rectangle shows the interquartile range of the responses for each question. The top of the rectangle is the third quartile (Q3), 75% of the data values are less than or equal to this value. The bottom of the rectangle is the first quartile (Q1), 25% of the data values are less than or equal to this value.
- The upper whisker extends to the highest data value within the upper limit (upper limit = Q3 + 1.5(Q3-Q1). The lower whisker extends to the lowest value within the lower limit (Q1 – 1.5(Q3-Q1).
- Outliers are values beyond the whiskers and denoted with an *.

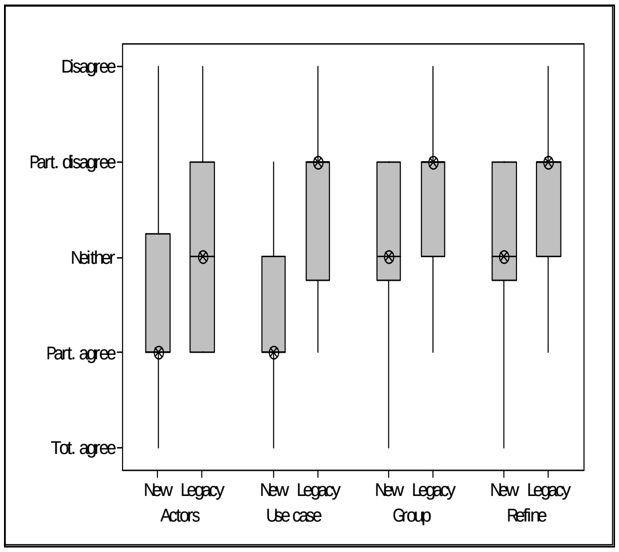

Figure 2. Ease of use case modelling

Identifying and documenting actors was considered a rather easy activity for the new-code group, but was more difficult for the legacy group. Those who enhanced legacy code had actors that were other (interacting) legacy subsystems. This required them to view those subsystems as actors with goals with respect to their own subsystem. From the interviews we learned that this was difficult conceptually, and it was also difficult because the different legacy subsystems often lacked appropriate descriptions of their interfaces at the outset of the project.

Identifying and documenting use cases was considered rather easy by the new-code group, but was much more difficult for the legacy group. This was the question in our questionnaire with the largest difference in median response between the two groups of respondents. From the interviews we learned that the legacy group experienced two difficulties:

1. Actors in the form of legacy subsystems do not have goals in the standard sense that can be used to derive use cases. Hence, it was particularly difficult to describe and delimit use cases initiated by such actors.
2. It was difficult to ensure a focus on functionality and top-down development when describing use cases for which there was already existing code. Little documentation of the existing systems meant that the code was an important source of information regarding functionality, and the code was also often more familiar to both developers and reviewers than was the external functionality.

Grouping actors and use cases into subsystems was considered rather difficult by most of the respondents, but was most difficult for the legacy group. From the interviews, we know that lack of documentation and traceability in the legacy code, in the cases where the existing code was unfamiliar, made it difficult to relate existing functionality, described as use cases, to code. Consequently, it was particularly difficult to group legacy actors and use cases into subsystems.

Refining use cases and deciding dependencies was also considered rather difficult by most of the respondents. Extending use cases were often used for adding functionality to existing use cases. From the interviews we know that this additional use of the extends construct had been difficult, which is a reason why the legacy group had more difficulties in this activity.

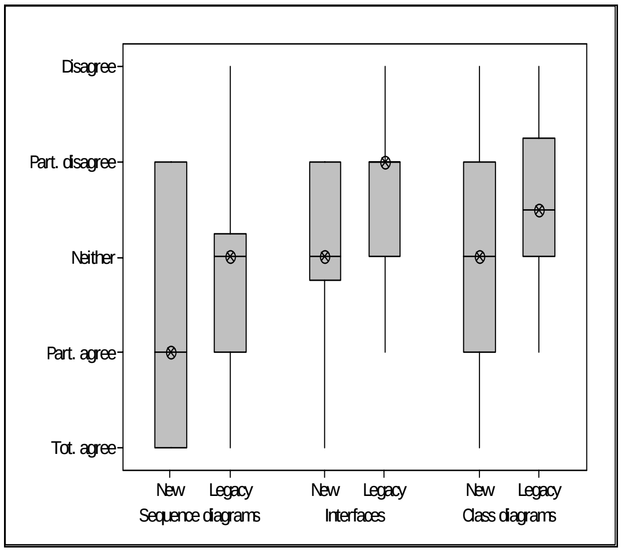

Figure 3. Ease of constructing sequence and class diagrams

4.1.2 Sequence diagrams

Figure 3 shows the respondents' opinions on ease of constructing sequence diagrams, and of defining interfaces between them, questions 2.1 and 2.2 in the questionnaire.

Constructing sequence diagrams was rather easy for the majority of the new-code group (based on the median value of the responses), but that was not the case for the legacy group. Sequence diagrams detailed out use case flows, so it is to be expected that more difficulties with modelling use cases led to more difficulties in constructing sequence diagrams for the legacy group.

From the interviews we know that those who used sequence diagrams to model legacy code with which they were familiar, tended to include actual function calls in the high-level sequence diagrams. Many of the interviewees also complained that reviewers who were familiar with the code often had a focus on discussing details in the code, rather than the higher level description of interactions between objects that is the focus of the sequence diagrams. A consequence of the latter was that more time was spent in reviews before the sequence diagrams were formally accepted, and consequently there was a feeling of more difficulties in the construction of sequence diagrams. The variation in the responses in the new-code group is very large on this question. Our data does not allow us to investigate in detail the causes for this. However, we know from the interviews that the developers had varying experience with SDL (System Design Language), which means that some had more previous experience with sequence diagrams than others. Some also felt that the syntax of UML 1.3 lacked expressive power, with respect to, for example, loops and guard conditions. We can thus speculate that some such factors have influenced the results on this question.

Defining interfaces to other subsystems was difficult and most difficult for the legacy group. From the interviews we know that it was difficult to identify appropriate interfaces to other subsystems also at this level. In practice, the interfaces tended to be actual function calls in the code, and it required good knowledge of the interacting subsystem to model this correctly.

4.1.3 Class diagrams

Figure 3 shows that the difference between the two groups was small with respect to the construction of class diagrams. The medians of the two groups show that neither of them found this activity easy, but more were positive in the new-code group. In practice, the legacy group did not manage to reverse engineer the legacy software completely, partly because the software was not completely object-oriented, and partly because of its size. From the interviews we know that it was particularly difficult to make interfaces for the old code, and to decide how much of the legacy code should be included in the new models to render them intelligible. Such decisions required good knowledge of the legacy code, as well as an ability to abstract away details. Some developers circumvented this by inventing classes to be used in the models and simulated interfaces to the old code. The class diagrams that modelled existing code were often very large. Nevertheless, the small difference between the groups indicates that it may not be much easier to model class diagrams from scratch than developing them based on the reverse engineering of existing code.

4.2 Use of the Diagrams

This section investigates the use of the different diagrams in the project activities that ABB wanted to improve by introducing UML based development; design, testing, communication in reviews and within teams, and documentation. All developers applied the UML diagrams in design and documentation, but although it was recommended to use the UML diagrams also in reviews, testing and in the communication within the teams, it was not compulsory. Hence, the respondents were asked to indicate if they had applied the diagrams in those activities. Table 1 shows how many of the 14 respondents in each of the groups had applied each of the diagrams in the different activities, for example 7 of the developers in the new-code group and 8 of the developers in the legacy group had applied use cases as input to testing.

Table 1 shows that the diagrams had been used little in code reviews, more in communication within the teams, but most in testing. The diagrams had been used equally much in the two groups. The diagram which was used the most was the sequence diagram.

TABLE 1. Use of diagrams

	Use case		**Sequence diagr.**		**Class diagr.**	
	New	**Legacy**	**New**	**Legacy**	**New**	**Legacy**
Review	0	2	1	4	5	4
Test	7	8	9	9	6	7
Comm.	5	4	7	7	6	6

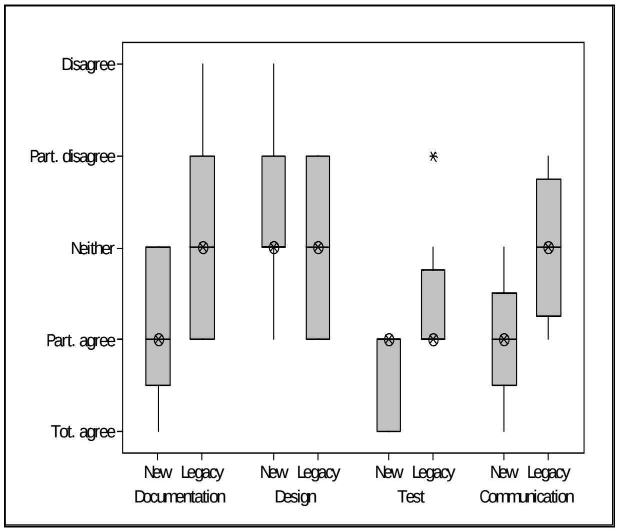

Figure 4. Utility of use cases

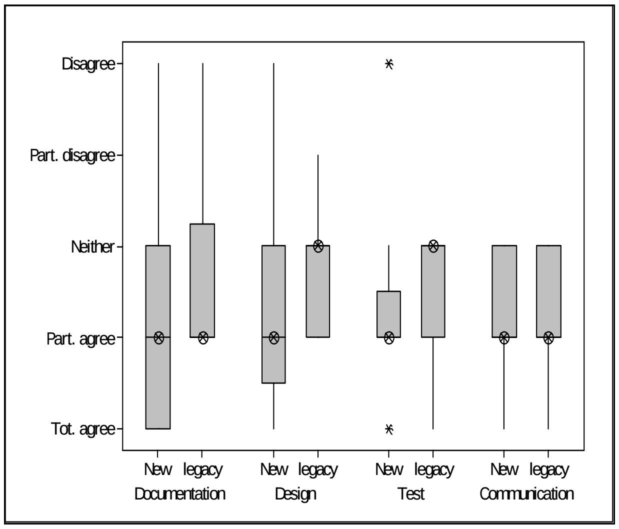

Figure 5. Utility of sequence diagrams

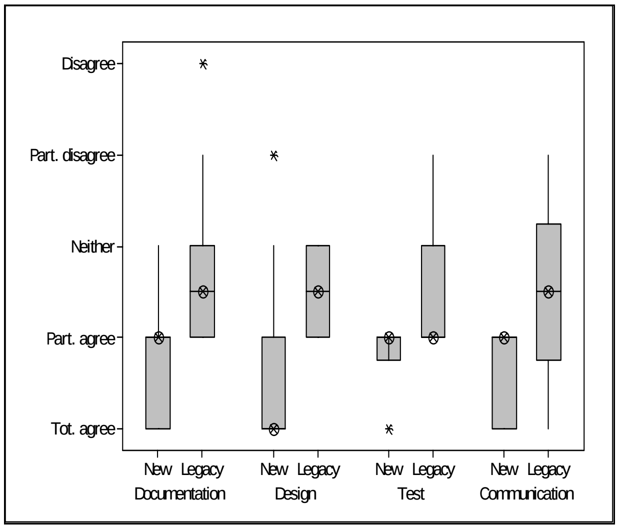

Figure 6. Utility of class diagrams

4.3 Utility of diagrams

ABB hoped for improvements from using UML-based development with respect to design, test, documentation and communication. Hence, the respondents were asked, for each of the diagrams, whether they had experienced any positive effect from using them in these activities. Figures 4 to 6 show the responses to questions 7 to 9 in the questionnaire. The responses are from those who had actually applied the diagrams in the specific activity. All the respondents had used the diagrams in design and as a means of documentation, but for test and communication the number of respondents is equal to the number of users shown in Table 1 (on question 7.4, about the effects on test cases, there were for example 7 respondents in the new group and 8 respondents in the legacy group). We have omitted the effects of diagrams on code reviews because of the small number who had actually used the diagrams in that activity.

4.3.1 Design

The respondents were asked whether the UML diagrams had a positive effect on the design of the system. For those developing from scratch, this question was about the extent to which the use of UML had been a help in obtaining a good design. The legacy group started out with code that was not always designed according to object-oriented principles. For that group this question was about whether the different UML diagrams had helped in improving the code structure towards a better and more object-oriented design.

Figure 4 shows that modelling use cases had not contributed positively to design for either of the two groups. There were, however, slightly more positive responses in the legacy group. From the interviews we know that both groups experienced difficulties with deriving classes and methods from use cases in design, but also that describing the existing code with use cases led to the identification of some possible improvements in the structure of the legacy code.

Sequence diagrams, however, had positive effects on design (Figure 5), and the new-code group had experienced more positive effects in design from applying sequence diagrams, than had the legacy group.

Class diagrams had contributed very positively to the design in the new-code group with the responses to this question being the most positive in the whole questionnaire (Figure 6). The legacy group was a bit more reluctant, but still this was the diagram that had had the best effect on design for this group.

From the interviews we know that the project members considered that the process of model-driven development, and consequently a focus on top-down development, leads to earlier and more focus on design. Using UML it was also easier to discuss a design among the developers and see how it would work.

4.3.2 Test

The responses about effects on testing in Figures 4 to 6 show that testing was the activity in which the effects of applying UML-based development were the most noticeable.

Having use cases had a positive effect on functional testing for both groups, but most for the new-code group. From the interviews we learned that functional test cases were quicker and easier to define when use cases were used as a basis, they were also defined earlier than before; and they were defined in a more structured way.

Sequence diagrams had been useful as input to integration test for the new-code group. Also in the legacy group some developers had experienced positive effects. From the interviews we learned that sequence diagrams were considered particularly useful for ensuring completeness of integration testing.

Class diagrams were also useful in unit test for all the respondents, and the responses from the new-code group were unanimous on this issue.

4.3.3 Documentation and communication

The opinions on documentation and communication were very similar, and these two uses of the diagrams were related in that good documentation was perceived as a prerequisite for successful use of it in communication.

For those developing from scratch, use case modelling had a positive effect on documentation and communication. Most members of the legacy group had, however, not experienced positive effects in these activities. We believe that that was because the legacy group experienced more problems in constructing use cases. From the interviews we know that use cases had improved the documentation of the new code because use cases enforced one structure on all the functional descriptions. The use cases made by the legacy group were also often too detailed to give an overall understanding of the functionality.

Sequence diagrams had had a positive effect on documentation and communication for most respondents. This indicates that the final sequence diagrams were of sufficient quality to be useful in providing an increased understanding of the system. From the interviews we learned that the sequence diagrams were found to be particularly useful for obtaining an overall understanding of the system.

Class diagrams had a positive effect on documentation and communication, in particular for the new-code group. Many interviewees meant that the “class diagrams were the code” and thus that the graphical representation of the class diagrams facilitated the understanding of the code.

From the interviews we know that communication within the teams was considered to have improved, due to having the UML models as a basis for discussions. It was for example easier to come up with suggestions for solutions when the UML diagrams were used in the discussions.

In the code reviews, however, there had been little use of the UML diagrams, and thus no improvements due to having UML diagrams. One reason for this may be that many of the participants in the code reviews were unfamiliar with UML.

Overall the respondents were most satisfied with the effects of class diagrams and least satisfied with the effects of use cases. We believe that is because the class diagrams are the closest to code, and thus it was easier for the respondents to have an opinion on the effects of this diagram.

5. RELATED WORK

There are few empirical studies on model-driven development and the use of UML to which we can relate the results of this study. To the authors' knowledge, there is only one study that evaluates model-driven development in a legacy environment [11]. In that study, an existing component from an industrial system was redeveloped and integrated back into the system by a student. The results of that study support our results in that they show difficulties with defining the boundary between model and legacy code, but also positive effects in communication due to the use of UML. In turn, together with a graphical notation that could easily be navigated, maintainability was improved.

In another study, the usability of use cases, sequence diagrams and class diagrams was investigated through the use of a questionnaire completed by students with experience of using UML on students' projects [1]. The results from that study showed that the students did not find UML diagrams to be very usable, and that, for beginners, use cases and state diagrams were easier to use than class and interaction diagrams. Our results were based on the opinions of professional developers with experience from applying UML on a large development project. Our results do not confirm that use cases are easier to use than class and interaction diagrams. This may be because of differences in the application domain, as the application domain has been shown to impact the ease of use of UML diagrams [12]. The different results may also indicate that the ease of use of the UML depends on the background of the developers.

A controlled experiment with students found that for complex task and when the subjects have passed a certain learning curve, the availability of UML documentation may result in significant improvements of the design quality of changes [3]. These results agree with our results in that we also found that the use of UML documentation may have a positive effect on design. Our results do not, however, support positive effects of UML diagrams in design quality in maintenance of large, complex systems. Our results are obtained in a very different context where the developers also had to construct the UML diagrams. Hence, the diagrams were not as "perfect" as the diagrams in the abovementioned study, thus the results are not directly comparable.

An experience report based on 15 years experience with model-driven engineering in a large company support our results in that they found improvement in testing to be the most important improvements due to the application of model-driven engineering [4].

Another experience report, based on the author's experience from several applications of UML in the design of embedded systems, claims that one of the most immediate benefits observed from adopting a use case driven UML design is the improved visibility to stakeholders [13]. Through the application of UML, software engineers were able to more readily communicate with systems engineers and end customers. This supports our results on communication to some extent, although our results on the lack of use of the UML diagrams in code reviews indicate that the UML diagrams are not always useful for stakeholders who are not familiar with UML. The experience report also pointed at challenges with respect to describing interfaces between UML models.

Furthermore, a web-based survey on the use of UML is reported in [6]. That study investigated to what extent key components of UML (including use case diagrams, use case narratives, sequence diagrams and class diagrams) are used and how useful those components are in clarifying technical issues, as programmer specifications and as maintenance documentation. The survey collected 171 responses from analysts using UML and 11 responses from people using UML components as part of another methodology. The results showed that use case, sequence and class diagrams are frequently used in software development; more than 50% of the respondents used the diagrams in more than 2/3 of their projects. Use case narratives were slightly less used with 44% using them in more than 2/3 of their projects. Class diagrams were considered very useful, 93% of those having used them in more than 1/3 of their projects found them useful in clarifying technical issues, 89% found them useful as programmer specifications, and 92% found class diagrams useful as maintenance documentation. Sequence diagrams were also considered very useful in these activities, although slightly less than class diagrams, while use case diagrams and narratives were found useful by between 60% and 80% depending on activity. These results on usefulness agree with our results on the utility of those UML diagrams with respect to documentation and communication.

6. CONCLUSIONS AND FUTURE WORK

A case study was conducted as part of a software process improvement initiative related to the introduction of model-driven development in a large company. The study investigated the ease of constructing, the use and the utility of use cases, sequence diagrams and class diagrams in modelling and enhancing legacy software compared with development from scratch. The case was a large development project applying UML in the development of a new version of existing systems, with most of the software being embedded. Some parts of the system were developed from scratch while others were based on existing components.

The results show that adopting the abovementioned diagrams in a legacy environment, where it was necessary to model existing code and functionality, as well as to introduce functional enhancements, yielded more challenges and was consequently more costly, than adopting UML in development from scratch. In particular, identifying and documenting use cases was much more difficult in legacy development than in development from scratch. The use of UML diagrams also yielded fewer benefits in legacy development, although there were some positive effects also in legacy development, in particular from applying sequence and class diagrams. The most positive benefits were obtained with respect to using UML diagrams as input to testing.

We believe that our results confirm that there is potential for model-driven development with UML, also in legacy development. Our results support a previous study that showed challenges with applying UML in legacy development, as well as previous studies showing that the use of UML may be beneficial in testing, documentation and communication. Nevertheless, the results also show a need for more methodological support on constructing and applying UML diagrams in legacy development, in particular, steps should be taken to improve methodological support for (i) reverse engineering from code to UML models, and (ii) the application of UML in the design of enhancements of legacy code.

The positive results with respect to UML-based testing has led ABB to an increased focus on how the UML-models can be used in a more systematic way to further improve the company's testing procedures. A follow-up study to this one is planned on the project developing the next version of the same system in order to continue gathering experiences with the use of model-driven development with UML in the company.

7. ACKNOWLEDGMENTS

We acknowledge all the employees of ABB in Sweden and Norway who participated in the interviews and completed the questionnaires; in particular, Cato Bratt and Ulf Hagberg. We also thank Hanne Kristin Thorsen for her help with transcribing the interviews and administering the questionnaires, and Dag Sjøberg and Ray Welland for comments on this paper. The reported work was funded by The Research Council of Norway through the industry project SPIKE (Software Process Improvement based on Knowledge and Experience).

8. REFERENCES

[1] Agarwal, R. and Sinha, A.P. Object-Oriented Modeling with UML: A Study of Developers' Perceptions. *Communications of the ACM*, Vol. 46, No. 9, pp. 248–256, September 2003.

[2] Anda, B., Hansen, K., Gullesen, I. and Thorsen H.K. Experiences from Using a UML-based Development Method in a Large Safety-Critical Project. Forthcoming in *Empirical Software Engineering*, 2006 (available as Simula Research Laboratory, Technical Report 2005-5).

[3] Arisholm, E., Briand, L.C., Hove, S.E. and Labiche, Y. The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation, *IEEE Transactions on Software Engineering*,Vol. 32, No. 6, pp. 365–381, 2006.

[4] Baker, P., Loh, S. and Weil, F. Model-Driven Engineering in a Large Industrial Context — Motorola Case Study. In L. Briand and C. Williams (Eds.) in MoDELS 2005, LNCS 3713, pp. 476-491, Springer-Verlag, 2005.

[5] Booch, G., Rumbaugh, J. and Jacobson, I. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[6] Dobing, B. and Parsons, J. How UML is used. *Communications of the ACM*, Vol. 49, No. 5, pp. 109–113, May 2006.

[7] Douglass, B.P. Real Time UML: *Advances in the UML for Real-Time Systems*. 3rd edition, Addison-Wesley, Boston, MA, 2004.

[8] Fowler, M. *UML Distilled. A Brief Guide to the Standard Object Modelling Language*, 3rd edition, Addison-Wesley, Boston, MA, 2003.

[9] IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems, (http://www.iec.ch/),1998.

[10] Lee, A.S. A scientific methodology for MIS case studies. *Management and Information Systems Quarterly,* Vol. 13, No. 1, pp. 33-50, 1989.

[11] MacDonald, A., Russell, D. and Atchison, B. Model-driven Development within a Legacy System: An industry experience report. Proceedings of the 2005 Australian Software Engineering Conference (ASWEC'2005). IEEE Computer Society, 2005.

[12] Otero, M.C. and Dolado, J.J. Evaluation of the comprehension of the dynamic modeling in UML. *Information and Software Technology*, Vol. 46, pp. 35-53, 2004.

[13] Pettit, R.G. Lessons Learned Applying UML in Embedded Software Systems Designs. Proceedings of the Second IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (WSTFEUS'04), pp. 75-79, Vienna, Austria, May 11-12, 2004.

[14] Segal, J. When Software Engineers meet Research Scientists: A Case Study. *Empirical Software Engineering*. Vol. 10, pp. 517-536, 2005.

[15] Selic, B. The Pragmatics of Model-Driven Development. *IEEE Software*, Vol. 20, No. 5, pp. 19-25, September/October 2003.

[16] Yin, R. *Case Study Research: Design and Methods*. 3rd edition. SAGE Publications, Inc., Thousand Oaks, CA, 2003.

APPENDIX A – EXCERPT FROM QUESTIONNAIRE

(All propositions in 1, 2, 3, 7, 8 and 9 are rated on a scale from totally agree to disagree. Several alternatives can be ticked in questions 4 – 6)

I was involved in:

development from scratch	□
enhancements of an existing system	□
both	□

Construction:

1. Use case modelling

1.1 Identifying and documenting actors is easy

1.2 Identifying and documenting use cases is easy

1.3 Grouping use cases and actors into different subsystems is easy

1.4 Refining the use cases and identifying dependencies that can be modelled using included and extending use cases is easy

2. Sequence diagrams

2.1 Constructing sequence diagrams is easy

2.2 Defining interfaces between subsystems is easy

3. Constructing class diagrams is easy

Use:

4. How did you apply the use cases in the project?

In code reviews	☐
As a basis for functional tests	☐
As a means of communication	☐

5. How did you apply the sequence diagrams in the project?

In code reviews	☐
As a basis for integration tests	☐
As a means of communication	☐

6. How did you apply the class diagrams in the project?

In code reviews	☐
As a basis for unit tests	☐
As a means of communication	☐

Utility:

7. Use cases contributes to

7.1 Good documentation of the system

7.2 Good design of the system

7.3 Efficient code reviews

7.4 Thorough testing

7.5 Good communication within the development team

8. Sequence diagrams contributes to

8.1 Good documentation of the system

8.2 Good design of the system

8.3 Efficient code reviews

8.4 Thorough testing

8.5 Good communication within the development team

9. Class diagrams contributes to

9.1 Good documentation of the system

9.2 Good design of code

9.3 Efficient code reviews

9.4 Thorough testing

9.5 Good communication within the development team