H. Cosmo, C. Wohlin and A. Johansson, "An Empirical Evaluation: Object-Based Cleanroom vs. Object-Oriented Software Engineering", Proceedings 3rd Annual International Conference on Cleanroom Software Engineering Practices, College Park, Maryland, USA, October 1996.

An Empirical Study: Object-Based Cleanroom vs. Object-Oriented Software Engineering

Henrik Cosmo, Claes Wohlin and Arne Johansson[1]
Department of Communication Systems
Lund University
Box 118, S-221 00 LUND, Sweden
Phone: +46-46-109009, Fax: +46-46-145823
E-mail: cosmo@tts.lth.se

Abstract

Experimental Software Engineering is needed to enable transfer of research results from the universities to industrial use. It is mostly not possible to take research results and apply them in a large experiment directly. Therefore, the technology transfer must be carried out in a number of steps: minor case study within a university environment, an enlarged experiment in industry and finally into real projects as the normal method or technique to use. This paper focuses on reporting some results from the first step in the dissemination process. A comparative study between two development methods is reported. The two methods are: object-based Cleanroom and Object-Oriented Software Engineering. The study shows that the object-based Cleanroom produces higher quality software with approximately the same effort as the more widely spread method. The results are, due to the small study, not statistical significant, but they are promising for the future and they ought to encourage industry to perform a larger experiment in an industrial setting. Both quantitative and qualitative results are presented.

1. Arne Johansson is currently with Q-Labs AB, IDEON Research Park, 223 70 Lund, Sweden.

An Empirical Study: Object-Based Cleanroom vs. Object-Oriented Software Engineering

Abstract

Experimental Software Engineering is needed to enable transfer of research results from the universities to industrial use. It is mostly not possible to take research results and apply them in a large experiment directly. Therefore, the technology transfer must be carried out in a number of steps: minor case study within a university environment, an enlarged experiment in industry and finally into real projects as the normal method or technique to use. This paper focuses on reporting some results from the first step in the dissemination process. A comparative study between two development methods is reported. The two methods are: object-based Cleanroom and Object-Oriented Software Engineering. The study shows that the object-based Cleanroom produces higher quality software with approximately the same effort as the more widely spread method. The results are, due to the small study, not statistical significant, but they are promising for the future and they ought to encourage industry to perform a larger experiment in an industrial setting. Both quantitative and qualitative results are presented.

1 Introduction

The transfer of new technology from research to practical application is difficult. New ideas and methods can not be incorporated into existing software processes due to lack of evidence in their ability to improve cost and quality. Therefore, experimental software engineering is needed. This can be done in several ways, either by performing experiments in real projects or in laboratories. Both of these approaches have their drawbacks, in the first case it might be too expensive to experiment in a real environment and in the second case it might be hard to draw general conclusions from a limited experiment. This paper presents a limited experiment performed within a university environment. The objective is to get indications, which could form the basis for deciding whether or not to continue with a larger experiment or pilot project in an industrial setting.

The presented study is limited and the reported work has been performed by master students during their thesis work. Therefore, it will not be argued that it has been proven that one method is superior to another, but it will be argued that the results are promising and hence a more extensive experiment with a real software project ought to be conducted. Thus, the study provides a basis for informed decisions instead of introducing a new method which has never been tried and evaluated quantitatively.

The results reported are based on four master thesis[1] performed, where the students developed several services for a telecommunication switching system. This paper is a summary of the results presented by the students in their theses, [Johansson94, Mahmoudi93, Engström94, Fridolfsson94][2].

1. A Master thesis in Sweden means approximately four months full time work, which includes writing the thesis.

The empirical study has been conducted to evaluate two different software development approaches. The aim was to compare a newly proposed object-based version of Cleanroom Software Engineering and Object-Oriented Software Engineering (OOSE) as presented by [Jacobson92]. OOSE is used in software projects today, while the proposed change to Cleanroom is quite new and it has never been applied fully in an industrial project. Cleanroom Software Engineering is presented in [Mills87, Linger94] and the proposed object-based change of Cleanroom is described in detail in [Cosmo94a, Cosmo94b]. OOSE and object-based Cleanroom are summarized in section 2, while the study is presented in section 3.

The objective of the study was to produce quantitative results and not only qualitative results in terms of judgements from the students. Prior to commencing the work a number of objective measures were formulated. The aim was to determine whether it was possible to state objectively that one method is better in terms of:

- method A is faster than method B in introducing new services,
- method A introduces fewer faults than method B.

It must be observed that one method may, of course, be better in one sense but worse in another. The measures of these aspects are presented in section 4.1. The objective measures are complemented with subjective judgements by the students. These are presented briefly in section 4.2, while a more extensive description can be found in the four theses. Finally, some conclusions are presented in section 5.

2 Brief introduction to the two methods

2.1 Brief introduction to Object-Oriented Software Engineering

As stated in [Jacobson92] Object-Oriented Software Engineering (OOSE) is an object-oriented process for developing large scale software systems. The process takes a global view of the system development and focuses on minimizing the life cycle cost of a system.

An essential part of the method is the concept of use cases. A use case specifies a flow of actions that a specific actor invokes in the system. For example a description of a normal telephone call between two users is a typical use case.

OOSE consists of three processes: analysis, construction and testing. In the work five different models are produced. During the analysis process the requirements model and the analysis model are produced. The result of the construction process is the design model and the implementation model. Finally, the testing results in the test model.

2. The theses and technical reports referenced are available upon request from the authors.

The Analysis process – Here the constructor builds a picture of the system to be created. The requirements and the analysis models are produced in this process (see figure 1).

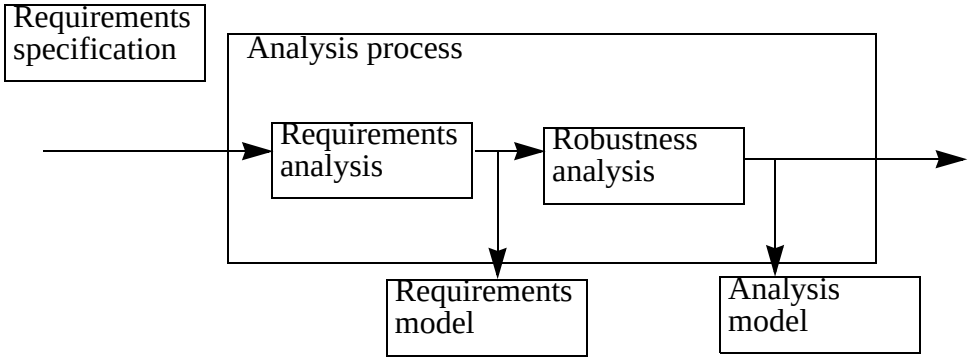

Figure 1. The Analysis process produces two models, the requirements model and the analysis model.

The requirements model – This model endeavours capturing the functional demands of the system, and hence all functionality expected in the system is specified here. The model contains three different parts: a use case model, an interface description and a problem domain model. In the use case model the functionality is described as use cases using natural language. The use case model is also the spine of the Construction and the Testing processes.

The analysis model – This model focuses on giving the system a robust and easy to change object structure. The requirements model is the input to this part of the Analysis process, i.e. where the object structure of the system is laid.

The Construction process – Both the requirements and the analysis models are inputs to this process. The design and the implementation models are produced in this process. Therefore, the result of this process is a complete system (see figure 2).

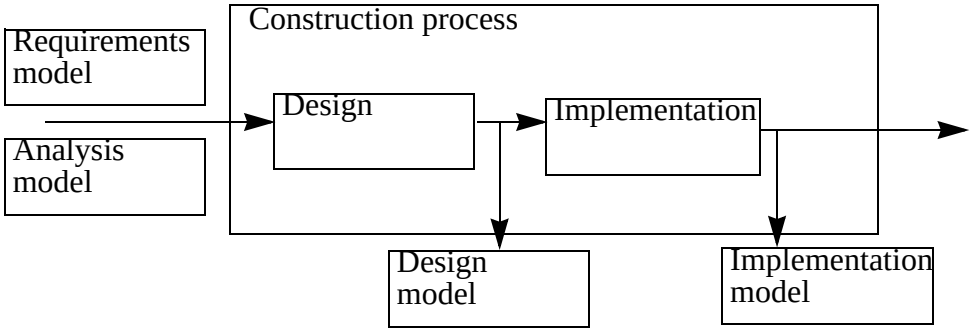

Figure 2. The Construction process produces two models, the design model and the Implementation model.

The design model – In the design model the object structure is adjusted to represent the implementation environment.

The analysis model has been developed under idealistic conditions and must now be adjusted to the circumstances prevailing. Any changes in the design model that are of logical nature, must be updated in the analysis model. The type of objects used in this model are called blocks. It is important to remember that the blocks are not the same type of objects as those in the analysis model. An interaction diagram is developed for each use case. It contains a detailed description of the different signals sent and received by the different blocks.

The implementation model – This model is the actual implementation of the system. It contains the source code. One does not necessarily have to use an object oriented programming language, but it is preferable. The design model is the base for the implementation.

The Testing process – In the Testing process the implementation model is tested, the test model is produced and a decision is taken whether or not the system is ready for delivery.

The test model – This model endeavours verifying the system.

2.2 Brief introduction to object-based Cleanroom

Cleanroom Object-Based Analysis and Design (COBAD) [Cosmo94a, Cosmo94b] is an object-based process for developing large scale software systems. The process focuses on minimizing the life cycle cost of a system by producing high reliability software. This is obtained by making the correct thing from the beginning and by making extensive verifications of the produced material early in the development. Each step in the development is verified against the previous step.

An essential part of the method is the concept of Stimulus-Response Diagrams (SRD) [Cosmo94a], which is a formal definition of a use case. An SRD specifies a flow of actions that a specific actor invokes in the system. For example a description of a normal telephone call between two users is a typical event that can be described in one SRD.

Development in COBAD is done through three processes: Specification and analysis, Design and implementation, and Certification. The work results in a number of models. During the Specification and analysis process, three models of the system are produced: the **use case model**, the **usage model** and the **object model**. During the Design and implementation process, five models are produced for each object: the **interaction model**, the **use case model**, the **usage model** and the **state box model** and the **implementation model**. Finally, the Certification process results in the **test model**. The names of the models are the same for some of the models in the Specification and analysis process and the Design and implementation process, which is due to that they describe the same aspects although on different system levels.

The Specification and analysis process – Here the constructor builds a picture of the external behaviour of the system. The use case model, the usage model and the object model of the system are produced in this process.

The use case model – This model focuses on capturing the functional demands of the system, i.e. all functionality expected in the system is specified here. The model contains three parts: a scenario model, a black box model and an interface description. In the black box model the functionality is based on use cases (or scenarios), using the Black Box Description Language [Cosmo94a]. In this study the black box model is not defined formally. In the interface description the actors of the system and the interface of the system are described.

The usage model – This model should capture the usage of the system. The model describes how the actors may use the system. Each possible sequence of stimuli into the system (the alphabet of the system) must be specified. This can be done as defined in [Runeson92] or in abstract grammar as it has been done in the study. The usage model is the input to the certification model.

The object model – The first step in analysing the requirements, to obtain the first internal description of the system, is to construct an object model. The object model shows the static data structure of the real-world system and organizes it into workable pieces. The object

model describes real-world object classes and their relationships to each other. The model focus is on the static structure as it is usually better defined, less dependent on application details, more stable as the solution evolves and it is easier for humans to understand.

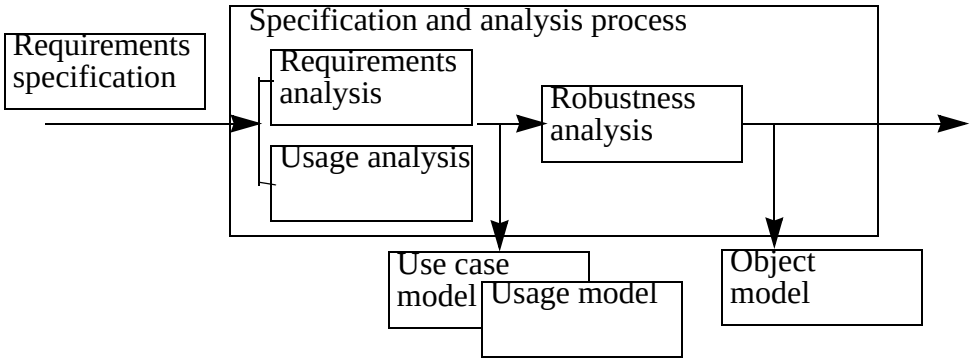

Figure 3. The Specification and analysis process produces three models.

The Design and implementation process – The three models from the Specification and analysis process are input to this process. The interaction model, the use case model of the objects, the usage model of the objects, the state box model and the implementation model are produced. The result of this process is hence a complete system.

The interaction model - An interaction diagram is developed for each use case. It contains a detailed description of the different signals sent and received by the different blocks. In the model the responsibilities of the system are distributed to objects in the object model.

The use case model of the objects – In this model the object structure is adjusted to represent the implementation environment and to capture the functional demands of the object. The model contains three parts: a scenario model, a black box model and an interface description. In the black box model the functionality is described as use cases (or scenarios), using Black Box Description Language. The model is not defined formally within this study. In the interface description, the actors of the object and the interface of the object are described.

The usage model of the objects – This model must capture the usage of each object of the system. The model describes how the actors may use an object. This is described using abstract grammar.

The state box model – In this model the focus is on distributing the responsibilities of the objects to data in the objects. The role of the state box is to open up the black box model of the object one step by making its data visible.

The implementation model – This model is the implementation of the system. It contains the source code. In this study, SDL [CCITT88] was used as implementation language. The state box model of each object is the base for the implementation model.

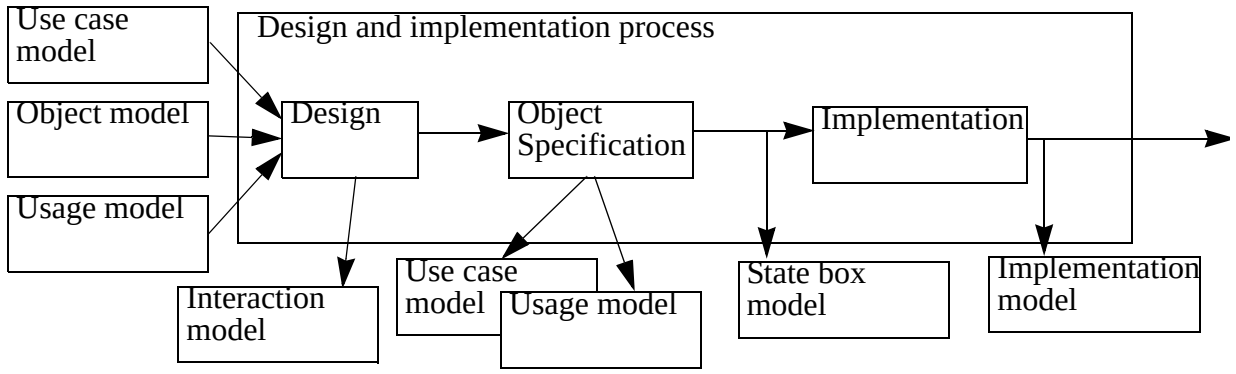

Figure 4. The Design and implementation process produces five models.

The Certification process – The usage models produced for the system and the objects are input to this process. These models are the basis for certifying the implementation model. The test model is produced and a decision taken whether or not the system is ready for delivery.

The test model – This model documents the verification of the system.

3 Overview of the study

3.1 Introduction

The comparison between OOSE and COBAD was made by developing a number of telephony services four times using the two methods, OOSE and COBAD, two times each. The study was made by letting four master students develop the services independently of each other using the two development methods. The development was made in three increments with one service in each increment. To make the study as independent as possible of the personal capabilities of the students, all four students have used both methods, hence avoiding a systematic error.

Measurements are a prerequisite to allow for comparison between the two methods based on the work conducted by the four students. The primary objective was to have objective measures on the differences, but as a complement subjective judgements from the students were collected as well. Therefore, the students were asked to summarize their experience in the theses [Johansson94, Mahmoudi93, Engström94, Fridolfsson94].

3.2 Organization of the study

In the study three telephony services (s1, s2 and s3) were implemented in four systems (I, II, III and IV). Two of the systems were specified, analysed and designed using COBAD and the other two systems were specified, analysed and designed using OOSE.

First, all four students (A-D) developed service s1 for one system each. Two of the students used OOSE and the other two used COBAD. When the development of the first service was finished then A and B exchanged systems and so did C and D, thus changing development methods too. Table 1 shows the relations between service, method and student. In the first increment, student A

developed service s1 in system I using the OOSE method, while student B developed the same service for system II using the COBAD method. The work was split in a similar way between student C and D. Thereafter, student A developed service s2 for the second increment of system II using the COBAD method, while student B developed service s2 for system I using the OOSE method. The students continued exchanging the systems as summarized in table 1.

Table 1. Relations between service, method and student

Method	**OOSE**	**OOSE**	**COBAD**	**COBAD**
System	System I	System III	System II	System IV
Service s1	Student A	Student C	Student B	Student D
Service s2	Student B	Student D	Student A	Student C
Service s3	Student A	Student C	Student B	Student D

It was emphasized for the students that it was extremely important that they did not cooperate as that would affect the results of this study. This was clearly understood by the students and to the knowledge of the authors there has not been any cooperation.

3.3 Development

The development of each increment was divided into two phases: Specification and analysis, and Design and implementation. These are the names of the processes in COBAD, but the names of the processes in OOSE can easily be mapped to these as follows: the Specification and Analysis phase is the Analysis process and the Design and Implementation phase is the Construction process.

3.4 Testing

In order to test the systems, test cases were produced for each service prior to development. The testing in COBAD is referred to as Certification in accordance with Cleanroom. Expected result files that predicts anticipated behaviour of the system, were also produced. The system was tested by functional simulation of the SDL graphs on the host machine, hence being a pure software test without taking the communication with the hardware into account. As a result of the testing, result files were produced. These were compared to the expected result files in order to certify the system. If a test case was not successful it was considered a failure. All test cases were run before the result files were studied. If there were failures, the location of the fault in the SDL design that caused the failure was investigated.

3.5 Metrics

As mentioned both objective measures and subjective measures were collected. The former was defined exactly, while the latter kind was rather unspecified.

3.5.1 Objective measures

Four objective measures were used:

1. Number of faults detected when testing
2. Consumption of time

The time used to finish each activity of the process was measured during the following activities:

-Specification and analysis

-Design and implementation

-Testing

3.5.2 Subjective measures

The subjective measure are not quantified, but explained in words. The following subjective measures were chosen:

1. Understandability of the design
2. Understandability of the method
3. Using the method

   This metric was used in order to find out if the method was easy to use, inspiring to use, fun to use or whatever was adequate to describe what was not covered by other metrics.
4. Coherence of the system

   This was used to measure how easy or difficult it was to make changes and additions to the system when introducing a new service.

4 Result of the study

4.1 Introduction

The collection of objective measurements as well as the subjective experiences are presented in this section. It was not possible to obtain statistical significance of the results in a minor study as the one presented here. The objective was rather to see if it was possible to establish an indication of the usefulness of a specific method. It is not feasible to conduct large experiments within a university framework. The objective of this type of minor studies at the university must be that they may work as pre-studies before testing a method proposal from the research community in an industrial environment, hence providing decision makers with some information to support them in their decisions. In particular, it was important for COBAD since it has not been used in an industrial environment to the same extent as OOSE. Only the subjective information concerning method usage is presented here, since it is judged that the most important issue is the usefulness of the method. The other subjective comments can be found in the four theses [Johansson94, Mahmoudi93, Engström94, Fridolfsson94].

4.2 Objective measures

The amount of time the students spent in the Specification and analysis phase of the study is shown in table 2, where the identities of the students are indicated by their code (A-D) in parenthesis. The total amount of time the students spent specifying with OOSE was 138 hours and with COBAD 231 hours. Thus, 67% more time was spent on the Specification and analysis phase in COBAD than in OOSE. This difference can be explained with that Cleanroom focuses more on the specification phase than OOSE does. The major difference between COBAD and OOSE in this phase is

the production of a black box model and a usage model in COBAD. Models like these are not produced in OOSE. This difference accounts for the 67% longer time in this phase, which is in accordance with the expectation.

Table 2. Time in the Specification and analysis phase.

Method	**OOSE**	**OOSE**	**COBAD**	**COBAD**
System	System I	System III	System II	System IV
Service s1	39 (A)	35 (C)	64 (B)	53 (D)
Service s2	33 (B)	13 (D)	37 (A)	27 (C)
Service s3	15 (A)	2 (C)	28 (B)	22 (D)
Total	88	50	129	102

The objective was not to compare the different students, since this is of little interest for the methods, but some remarks can be worth making. From table 2, it can also be noted that the total time spent in this phase by the different students varies between 64 (C) and 125 (B) hours. The fastest student used OOSE for two of the services while the student who spent most time in the Specification and analysis phase used COBAD for two services. The tendency that COBAD takes more time is quite clear from the study.

Table 3 shows the amount of time the students spent in the Design and implementation phase of the study. The total amount of time the students spent in the Design and implementation phase with OOSE was 321 hours and with COBAD 330 hours. The amount of hours spent on design with OOSE and COBAD was approximately the same. The major difference between COBAD and OOSE in this phase is that in COBAD the use case model, the usage model and the state box model are produced. Models like these are not produced with OOSE. It is interesting to see that although COBAD requires that three more models must be produced, the amount of time spent in this phase was approximately the same. One possible explanation is that the extra time spent in the Specification and analysis phase starts to pay back. It can be noted that the design time varies considerably between the different increments (services). The tendency was that it went faster for each increment, which actually has two explanations: the services were judged as becoming a little simpler and the students had learnt the method by developing the first service. The latter explanation is only valid for the third service, which is the first time when they use the same development method.

Table 3. Time in the Design and implementation phase.

Method	**OOSE**	**OOSE**	**COBAD**	**COBAD**
System	System I	System III	System II	System IV
Service s1	76 (A)	106 (C)	81 (B)	69 (D)
Service s2	59 (B)	48 (D)	73 (A)	42 (C)
Service s3	20 (A)	12 (C)	46 (B)	19 (D)
Total	155	166	200	130

The time in the Design and implementation phase varied between the students from 136 (D) to 186 (B) hours. The difference in this phase was, however, smaller than in the Specification and analysis phase.

The amount of time the students spent in the testing phase of the study is shown in table 4. The total amount of time the students spent on testing with OOSE was 108 hours and with COBAD 59

hours. 83% more time was spent in testing using OOSE compared to COBAD. The original testing methods of OOSE and COBAD were not used in the testing phase. Instead the testing process briefly described in section 3.4 was used. The same test cases were run on all systems. The test cases were developed by the student producing system I. The other students do not know about the test cases prior to testing.

Table 4. Time in the Testing phase.

Method	**OOSE**	**OOSE**	**COBAD**	**COBAD**
System	System I	System III	System II	System IV
Service s1	16 (A)	51 (C)	8 (B)	14 (D)
Service s2	18 (B)	16 (D)	10 (A)	18 (C)
Service s3	2 (A)	5 (C)	4 (B)	5 (D)
Total	36	72	22	37

The total testing times for three of the students were very similar, but there was one student who spent more than twice as much time testing as the others, that was student C. This student was one of two students who spent least time in the development phases, and this is the payment for going to fast through the development phases, see below. It is, of course, impossible to state this with any certainty with just this small set of data, but it does support the suspicion that it is important to do the work thoroughly and right from the beginning. The latter being the leading star in Cleanroom.

The total time spent developing the three services varies between the students from 259 (D) to 341 (B) hours. This difference is not very large taking into consideration that it can be explained both with differences between the methods as well as differences between the students.

Table 5 shows the amount of faults found in the testing phase of the study. The total amount of faults found with OOSE was 52 faults and with COBAD 24 faults. 116% more faults were found in the code produced with OOSE compared to the code produced with COBAD. The results are not statistically significant, but they give quite an indication that COBAD actually produces software with fewer faults.

Table 5. Faults found in Testing phase.

Method	**OOSE**	**OOSE**	**COBAD**	**COBAD**
System	System I	System III	System II	System IV
Service s1	12 (A)	23 (C)	3 (B)	6 (D)
Service s2	4 (B)	8 (D)	6 (A)	7 (C)
Service s3	1 (A)	4 (C)	0 (B)	2 (D)
Total	17	35	9	15

The number of faults found by the different students varied between 7 (B) and 34 (C), which is interesting as student C was the one who spent quite little time in the development phases, but had to stay longest of the students in the testing phase. It is also worth to notice that student B spent most time of all students on the development, but on the other hand produced the best software (from a fault point of view). These implications must be carefully judged as it is a trade off between cost (in effort) and quality (in terms of faults).

The total time spent on OOSE was 567 hours and on COBAD 620 hours. The total time spent on implementation of the system using COBAD was 8% more than the amount of time spent with

OOSE. The code produced with OOSE contains 116% more errors than the code produced with COBAD.

On the one hand, it can be seen that the difference in time spent is quite small while the difference in fault content is quite large. On the other hand, the results depend very much on the individuals involved, but the figures seem to indicate that the industry would not take a chance by introducing COBAD. A tendency is also that it seems to pay off to stay in the Specification and analysis phase a little longer to be sure to be in control before continuing on to the next phase. Based on the above, it is concluded that this minor study indicates that the research conducted to develop COBAD is ready for a larger industrial evaluation before being disseminated to a whole organization. The latter depends, of course, on the outcome of the industrial trial.

4.3 Subjective measures

4.3.1 Cleanroom Object Based Analysis and Design

Student A expressed his feelings with Cleanroom as: “Working with the design required an enormous patience, since each step of the development seamed almost infinite. The reason it seamed endless was probably the necessity of verifying every step against the previous one.”

Student B said: “The frequent used verification of the system contributes to find the faults in the same step they are introduced in. It may take more time to develop a system using stepwise verification. When you verify each step against the previous one, the faults are hopefully found and will not cause any complicated and hard to fix faults in the design.”

Student C has written: “When I first read the requirements specification, I quickly thought that I could write the solution down in SDL code directly. As I later studied the development method, I realised that this solution was not accepted in the method. It was a little bit hard to restart and do it all over again, as I felt I already had a good solution. I feel that introducing a service in a small system is a little bit too much work. I find that the only problem I had when developing the system was to understand what to do in the different steps and not the service itself. What I liked about Cleanroom was the Stimuli-Response pairs. Each Stimuli-Response pairs enforced that you had to think through every possible error case, hence I felt that I had very good control over the development.”

In student D’s thesis we can read: “A big advantage with Cleanroom is that you have to think the problem over several times. For example, the Stimuli-Response pairs with error cases are analysed already in the beginning of the development. Already after the two first weeks, when studying OOSE and Cleanroom, I thought that Cleanroom was rather difficult to learn. But as soon I had learnt it, the method was quite easy to work with. This feeling has grown stronger as I have worked with the method.”

In summary, we can see that there are three positive students and one negative student. In particular, the latter student thinks he could have done the work much faster by going directly to the coding phase. When considering the statements from student C, we must remember that student C was the student who had most problems with the fault content and found more than twice as many faults as the other three students. This seems to be a quite clear indication that it is worth spending time in the early phases of the life cycle. Quality comes from hard work from the beginning and not just focusing on the coding phase.

4.3.2 Object Oriented Software Engineering

Student A writes: “In the end it was easy to find the detected faults. It was quite easy to correct the faults as well, the latter was probably due to the object orientation of the models. From the start of the use case design the work turned out to be straightforward.”

Student B states: “It was first in the middle of the design model, the method forced me to think of other cases than the normal scenarios. I think this was too late, because in this step the amount of information I had to deal with was so extensive that even in this small project, it was quite hard to consider all possible signals between different objects.”

Student C express his feelings as this: “In the requirements model I had a little bit of problem of defining how detailed the use cases should be as I thought I already had the solution. I could have made the use cases unrealistically detailed. I think the model was easy to find and the model stayed the same for the rest of the increment. As our system was quite small I can imagine that the design model makes more sense if you are developing a much larger system.”

Student D says: “The OOSE method is very easy to learn but it is hard to use. OOSE tends to give, so called, unstructured code (unreadable code). I realize that this also has to do with how experienced and capable the programmer is. In the OOSE method, it is easy to forget some of the error cases. First in the testing, it is shown that some are forgotten. Quite big changes have to be made in the code if forgotten error cases are found. OOSE is fast and good on smaller projects, but it is a rather difficult method if you are developing a large system.”

The viewpoints of OOSE varies between the students, but two of them believe that it is difficult to apply to large systems. They have quite different opinions concerning how easy it is to change the model and whether the method supports you enough in finding the different error cases. It is therefore difficult to come to any general conclusion based on the subjective statements given by the students when the finished their work.

5 Conclusions

Experimental Software Engineering is important. Unfortunately, it is infeasible for most universities to conduct large experiments to validate method proposals. Therefore, it is important to realise that limited studies can first be conducted within a university framework to make it plausible that a particular research result, for example a development method, is a valuable contribution to the software industry. Based on the results from this type of limited studies, it may be possible to determine whether or not to try a larger experiment within an industrial environment. This way is believed to be one of the few ways to transfer new research results to broad industrial use.

This paper presented one limited study, where one established method was compared with some new research results in the form of an object-based Cleanroom method. The results are inspiring for the new method as it produces the same software with fewer faults in approximately the same time. Therefore, it is concluded that the new method ought to be worth introducing on a trial basis in an industrial environment.

The method has since this study been introduced into a pilot project in the industry. The people working with the method has learnt the object-based Cleanroom without any major difficulties. The project is currently run so it is not possible to provide any quantitative results from the project,

but the people working with the method are positive and hopefully some quantitative experience can be released in the future.

6 References

[CCITT88] CCITT recommendation Z.100, 1988.

[Cosmo94a] Henrik Cosmo, “Black Box Specification Language for Software Systems”, Licentiate thesis, Department of Communication Systems, Lund University, 1994.

[Cosmo94b] Henrik Cosmo, “Cleanroom Object-Based Analysis and Design”, Technical report, Department of Communication Systems, Lund University, 1994.

[Engström94] Jan Engström, “A Comparison Between Objectory Software Engineering and Cleanroom Software Engineering”, Master thesis, Department of Communication Systems, Lund University, 1994.

[Fridolfsson94] Jonas Fridolfsson, “Cleanroom vs. Objectory: A Comparison of Two Software Engineering Methods”, Master thesis, Department of Communication Systems, Lund University, 1994.

[Jacobson92] Ivar Jacobson, M. Christerson, M. Jonsson and G. Övergaard, “Object-Oriented Software Engineering”, Addison-Wesley, 1992.

[Johansson94] Arne Johansson, “Objectory and Cleanroom: A Comparison of Two Software Development Methods”, Master thesis, Department of Communication Systems, Lund University, 1994.

[Linger94] Richard C. Linger, “Cleanroom Process Model”, IEEE Software, pp. 50-58, March 1994.

[Mahmoudi93] Ahmad Mahmoudi, “A Comparison Between CRSE and OOSE”, Master thesis, Department of Communication Systems, Lund University, 1993.

[Mills87] Harlan D. Mills, Michael Dyer and Richard C. Linger, “Cleanroom Software Engineering” IEEE Software, pp. 19-24, September 1987.

[Runeson92] Per Runeson and Claes Wohlin, “Usage Modelling: The Basis for Statistical Quality Control”, Proceedings 10th Annual Software Reliability Symposium, Denver, Colorado, pp. 77-84, 1992.