C. Wohlin, "Meeting the Challenge of Large Scale Software Development in an Educational Environment", Proceedings Conference on Software Engineering Education & Training, pp. 40-52, Virginia Beach, Virginia, USA, 1997.

Meeting the Challenge of Large-Scale Software Development in an Educational Environment

Claes Wohlin
Dept. of Communication Systems
Lund Institute of Technology, Lund University
P.O. Box 118, S-221 00 Lund, Sweden
Phone: +46-46-222 3329, Fax: +46-4-145823
E-mail: claesw@tts.lth.se

Abstract

This paper describes a final-year Master´s course in large-scale software development. A number of issues such as closeness to an application domain and executing the software on a real target system are stressed in order to imitate an industrial environment. The experience gained from the course is discussed. In particular, effort data (in terms of man-hours) from the course are presented, both from the plan devised by the students, and the actual outcome. Furthermore, it is discussed how the data can be used to create an experience base for the future. The objective for next year´s course is to let the students plan their projects based on the experience base. The experience base will also form the basis to complement the course with a complete experience factory.

1. Introduction

Teaching large-scale software development is not a matter of teaching programming and software design. The only way in which students can really can learn to develop large software systems is in the form of projects, where they work together and are forced to cooperate and to experience all the problems related to large software projects. This paper presents a course where 14-19 students worked together to develop a number of telecommunication services based on an existing software system. The students are given different roles. The project is concluded with an acceptance test and actual execution of the students´ software on a telephone exchange, i.e. the services developed by the students are executed when the phones are in use. The length of the course is 7 weeks and each project is equivalent to 600-1100 hours. Approximately 20 hours of lectures are also included in the course.

Planning of the projects is stressed, and the students should prepare a software development plan at the beginning of the project. This plan includes, among other things, a schedule and an effort plan. The students acting as project leaders are requested to collect a number of metrics during the course, which means that at the end of the course, a great deal of data is available. The outcome can then be compared with the initial plan. The experiences gained from the project (course) are described in a project report. This report is part of the final presentation.

To further improve the course in coming years the data collected have been transformed into an experience base [1], which next year´s project leaders should be able to use as input in their planning. The data and the experience base are presented here.

It is planned to use the data from the projects as part of a four-step experiment [2], in which the experience base and the experience factory are further formalized each year. The objective of the experiment is to study the formalism required for the experience factory to make the estimates of the students better than when they are only given a rough estimate of the expected size of the project. This rough figure is assumed to model the personal experience the project leaders would have had if they had been project leaders or participants in an industrial project.

The course has several unique features.

- The students execute the software they have developed on a real telephone exchange at the end of the course. This means that they are able to use the telephone services developed during the course on real telephones.
- The data collected from the course are fed into an experience base, which will be available to students during forthcoming years to improve planning and control of the projects in the course. This is also connected with research at the Department concerning experimentation and the experience factory concept.
- The course in large-scale software development is used as the study object in the software engineering course provided by the Department. This includes evaluating the organization and the projects in the course using the Capability Maturity Model (CMM) [3].

This paper is organised as follows. In Section 2, some background is given about the Master´s Programmes and the abilities of the students when entering the course. Section 3 discusses the large-scale software development course in detail, while Section 4 presents an informal CMM assessment of the projects conducted within the course. Finally, some conclusions are presented in Section 5.

2. Background

2.1 Master programmes

In Sweden, a Master´s degree in engineering corresponds to 4.5 years´ full-time studies, which is equivalent to 180 Swedish credits (i.e. one year equals 40 credits). At Lund University, there are 7 Master´s Programmes in engineering, and software engineering is involved in two of them, i.e. Electrical Engineering and Computer Science and Technology. It is normal to enter Master´s Programmes after nine years of compulsory education and 3 years´ upper secondary education at an age of 19-20 years.

The Department of Communication Systems is one of 15 departments providing courses for students following the Electrical Engineering or Computer Science and Technology Programmes. The number of employees in the Department is 25, including academic staff, postgraduate students, and administrative and technical personnel.

The Department gives nine courses within the Master´s Programmes, plus four other courses. The Master´s courses range from telecommunications courses to theoretical courses in queueing theory, and three of the courses are software engineering courses. The Department has traditionally focused on systems analysis, and in particular, performance analysis of telecommunication and computer networks. In the middle of the 1980´s, it was, however, realized that it was not possible to teach a system view of large systems without taking the software into account. Thus, it was decided to provide a number of software engineering courses, with a particular focus on large-scale software development, where the software process was judged to be a critical success factor.

It should also be noted that the staff teaching the courses at the Department are also involved in research. This is not always the case at other departments. It was, however, found important, both for research and education, for staff to be involved in both tasks. The feedback between teaching and research is important, and provides a natural way of enhancing the courses and including new material.

2.2 Students entering the courses

The software engineering courses are optional in both the Master´s Programme in Computer Science and Technology, and for the students in Electrical Engineering who have chosen to specialize in telecommunications. The courses are normally taken in the fourth or even fifth year, with a few students taking them in their third year. This means that the students are very familiar with computers, programming and mathematics. They have completed a number of courses in mathematics, statistics, physics and electronics, which, of course, are combined with a number of courses on computers and computer science. The computer science courses, which are of particular interest, include programming with different paradigms, compiler technology and object-oriented design. The students have worked on projects, but normally only with design and programming issues.

The students attending the courses often write their Master´s thesis within the subject, based on work done either at the Department or in industry. In Sweden, students are allowed to perform their Master´s project in industry, as long as it is not much like temporary employment. It is up to the supervisor at the university to ensure that the educational objectives of the project are maintained when the work is performed in industry. The academic supervisor is also the person responsible for actually approving the Master´s thesis, the industrial representatives have no voice in this decision.

2.3 Software engineering courses at the department

The Department provides three software engineering courses within the Master´s Programmes. The objective is that these should complement the traditional computer science courses taught at the Computer Science Department. The three courses are:

- Large-Scale Software Development (the course in focus here)
- Software Engineering [4]

  This course provides an overview of software engineering, and also includes a project in which the students are faced with a problem in software engineering. They are called upon to propose a solution based on information in relevant literature which they must find themselves. The project could, for example, be to propose how a fictitious company can implement a measurement programme.
- The Personal Software Process (PSP) [5]

  The PSP will be taught for the first time in the autumn of 1996. The objective is that the students should learn how to develop software systematically, and base their work on processes, process improvement and measurement.

The three courses within the Master´s Programmes have a number of things in common.

- They have emerged from an identified need in industry.

- They have been developed based on the ideas of the people involved in the development of the courses. The courses have not been influenced, at least not to any large extent, by courses at other universities. Even the PSP course is based on our own experience and tailored to our needs, although it is based on a book written more or less as a textbook [5].
- The focus is on software process improvement.
- They are project based, i.e. most of the work is carried out in project form, either individual projects or in project groups.
- They are based on role play, where the students and staff at the Department play different roles.
- Guest lecturers are used in the courses to emphasize industrial needs and importance. These lecturers are often highly appreciated by the students.
- Last, but not least, the courses have a good reputation among the students, and they are also acknowledged as important courses by some of the software companies in the south of Sweden. The large-scale software development course was ranked as one of the best courses by the students in a recent evaluation of the Master´s Programmes.

Although, the courses have much in common as can be seen from the above, they are quite different and their aims are different. This paper deals with the large-scale software development course, although some information is provided regarding the software engineering course. The reason is that in the latter course, the large-scale software development projects are used as study objects when the students are asked to perform assessments using CMM [3].

3. Large-scale software development

3.1 Introduction

This course emerged during the late 1980´s, with the objective of providing a course which, based on a standardized process model and design language, imitated a real software project as closely as possible, given the limitations of a university environment. This objective was made possible through the following three main industrial contacts, although others contributed through their advice.

- Telub AB, Sweden, where two software engineering consultants with long experience of the problems encountered in software development, in particular with quality assurance of software, put a great deal of effort into discussing the content of the course and providing an industrial view.
- Ellemtel AB, Sweden, which was a subsidiary of Ericsson and Telia (Swedish Telecom), provided the means of executing the software developed by the students on a real exchange. Ellemtel had, at this time, performed a number of experiments on programming a telephone exchange in different languages [6]. As part of their work, they had developed an easy method of programming the exchange in any language for which a compiler was available. This environment is further described in the next subsection.

- Telelogic AB, Sweden, provided the tool support. Telelogic markets the SDT™[1] (SDL Design Tool) environment, which supports high-level specification and design using SDL (Specification and Description Language). SDL is standardized by ITU-T (The telecommunication section of the International Telecommunication Union), [7, 8]. SDL is primarily aimed at telecommunication systems, but it can be used for other real-time applications. SDT is further described in the next subsection.

A realistic software development course is viewed by the Department as one of the cornerstones of software engineering education. The students need hands-on experience, where they have to work together in a large project using industrial tools and can actually execute their software on a real piece of hardware, in this case a telephone exchange. At the end of the course, the students are able to use their own software by phoning each other over the exchange.

It has not been possible to find any good generally available literature as the course is based on combining a target machine, a development tool, language and a process model from different sources. Therefore, the material provided for the students has been developed at the Department for this particular course [9]. This material is normally complemented with some general software engineering articles, for example [10], and slides from the guest lecturers.

The course is normally taken by 100-125 students each year.

3.2 Development environment

The development environment has briefly been mentioned above, but more should be said as it is one of the unique features of this course. The environment can be divided into seven building blocks.

- Exchange

  The exchange is a digital switch with a capacity of 100-10000 subscriber lines. The exchange is a modular SPC system (Stored Program Control) supporting integrated voice and data communication.

- Workstation

  Development is carried out in a unix environment, i.e. a number of workstations connected in a network. At the end of the project, the software is copied to the workstation connected to the telephone exchange. This workstation replaces the microprocessors in the exchange. This means that software executed on the workstation replaces the switching and controlling functions normally made by the microprocessors of the exchange, see Figure 1. This enables the execution of software in any language, although SDL (Specification and Description Language) [7, 8] and C are used in the course, see Figure 1.

1. Trademark of TeleLOGIC AB, Malmö, Sweden.

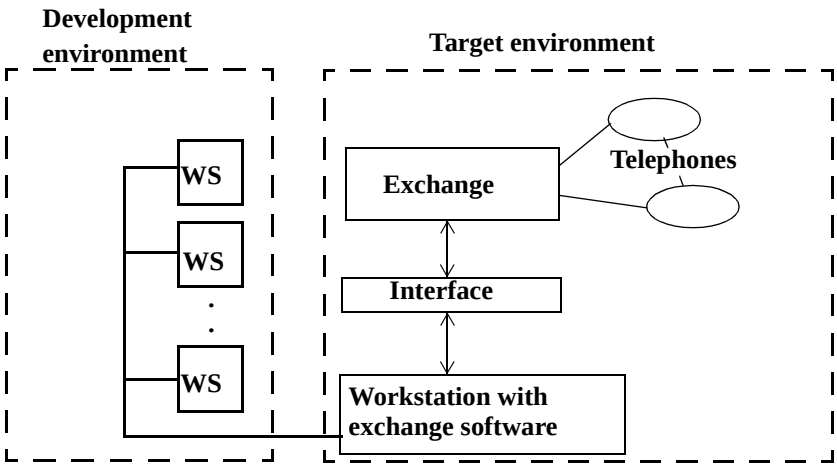

FIGURE 1. The basic system.

- Process model

  A somewhat simplified version of the US Department of Defence process model 2167A [11] is used. This is a waterfall model and the objective is to provide a standardized approach, rather than experimenting with a more sophisticated process model. The model, as used in the project, is outlined in Figure 2. In particular, the baselines in the model are emphasized to provide checkpoints in the process (see also the next subsection).

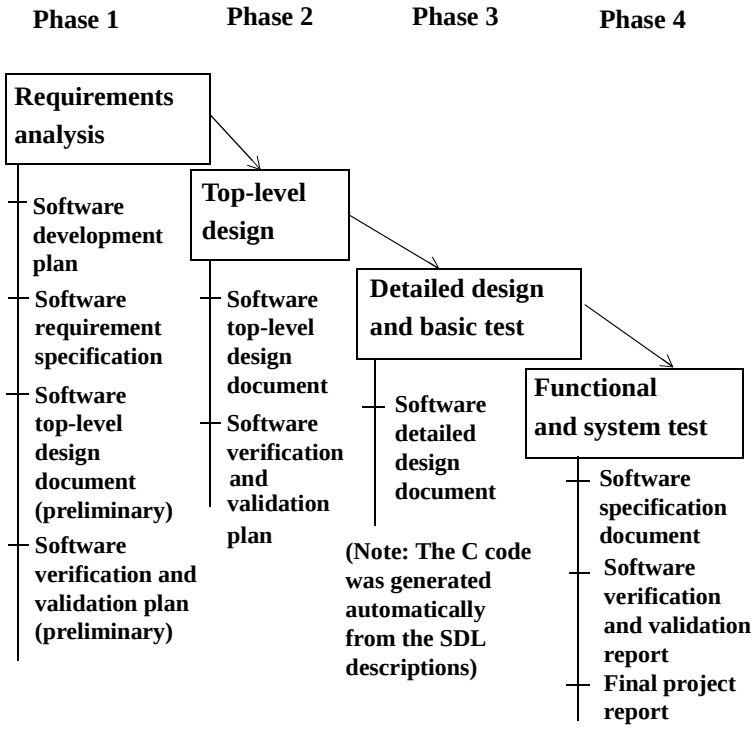

FIGURE 2. The waterfall development model.

- Project

  The software for normal telephone calls is already available on the workstation, and the students are therefore asked to provide some additional subscriber services. This, to some extent, turns the project into a maintenance project, as their software must work together with the existing software.

- New services

  The students are asked to implement the following four services: Call Forwarding Unconditional, Take Call, Debiting and Maintenance (groups with 14 students do not develop the maintenance service). “Call Forwarding Unconditional” is the service where upon ordering calls are moved to another subscriber number unconditionally. The “Take Call” service implements the opportunity to take calls to one number from another telephone by dialling a code and then picking up the incoming call.

  “Debiting” implements both normal debiting for calls and additional debiting resulting from other services, for example, double debiting for call forwarding. The “Maintenance” service includes adding and deleting subscribers, and changing telephone numbers. The services implemented will not be described further in this paper. The students are given quite straightforward customer wishes concerning these services, and no information is provided concerning service interaction, special cases and erroneous cases. These should be specified by the students and communicated to the customer, a role played by a Department representative.

- SDL

  To avoid the course becoming a programming course, it is essential to work with a well-defined design language. In this particular case SDL was chosen due to the availability of tool support and the fact that it has been standardized for telecommunication systems. Other high-level languages would have done as well; SDL is not a prerequisite. SDL is based on extended finite-state machines with states and signal sending. The students are taught SDL very briefly, which means that they learn to use the basic concepts without becoming SDL experts.

- SDT

  SDT allows the students to define their services in the graphical syntax of SDL. The service descriptions can then be analysed syntactically and semantically, before the C code is generated automatically. Thus, the code is generated from the detailed design in SDL. This means that the emphasis can be on project and process issues, rather than on programming problems. SDT runs on workstations and after being thoroughly tested the C code generated is copied to the workstation connected to the telephone exchange (see Figure 1). Thus, development can be carried out on any of the available workstations, while the final execution takes place in the workstation connected to the switch.

3.3 Project

The background to the project and some basic knowledge of the process model, SDL and SDT are provided through a series of lectures, although most of the work is performed by the students in their project groups. The groups consist of 14-19 students divided into 7-8 subgroups, one subgroup acts as project leaders, one subgroup is a system group responsible for keeping the system together, three or four subgroups are development groups, where each group

develops one service, and finally two subgroups are test groups testing two services each, and with joint responsibility for the system test.

The roles of the subgroups are combined with roles played by Department personnel. Staff from the Department play the roles of customer, external quality assurance personnel and, of course, technical experts, who can help the students when they run into difficulties. The customer reviews the material produced twice during the project and also performs an acceptance test at the end of the project. It should, however, be noted that executing the process correctly is viewed as equally important as the final product.

3.4 Conducting a project

The first two weeks are quite hectic, with a number of lectures providing the students with an introduction to the development environment and the project. The students are allowed to form groups on their own, and the project leader subgroup is appointed. This subgroup is then responsible for the division into the other subgroups. This should be settled after the first week, and they should familiarize themselves with their work in the forthcoming weeks.

The requirement analysis documents should be written during the second week, and should be ready for customer review during the third week of the project. The requirement specification should form a baseline after the review. In the third and fourth weeks of the project, the students mainly work on the top-level design document. This document is reviewed in the fourth week, which means that the customer can check that the project is heading in the right direction.

Work during the latter part of week four and into week six is focused on the detailed design document, and the product should be delivered at the end of the sixth week. A project report should be delivered together with the product. This report should contain information about the work, and in particular metrics should be reported concerning time expenditure and faults found during inspection and testing. The delivered product is acceptance tested in the seventh week, and the students are given feedback on their performance.

3.5 Data collection and analysis

Data from the projects: Each project reported effort data (in number of man-hours); the planned effort expenditure was reported in the software development plan at the beginning of the project and the actual outcome was reported in the final project report. The data were not reported consistently by the different project groups, but have been reorganized to make the data as comparable as possible across the projects. An example data set (Project 1) is shown in Table 1. The data in the table are quite typical in the sense that there is a footnote explaining some problems in attaining the data.

Some remarks may be pertinent here. Firstly, we can observe both very small and very large deviations between the plan and the outcome. Secondly, meeting times are sometimes included in the phase data, and sometimes reported separately. Thirdly, for one project individual times are not available for the subgroups. Fourthly, the services being developed have been judged as being of about the same difficulty, hence the individual times for the services are not used when creating an experience base (see below). The latter is also due to the fact that in a realistic situation, we would probably save data on a service level, perhaps divided into a number of classes based on difficulty, but we would not expect to develop the same service more than once. Finally, it should be noted that although the project as a whole is only approximately 800-1000 man-hours, it imitates a much larger project. The students carry out activities which should be natural parts in projects 1000 times larger than what is feasible within the course.

Table 1. Effort distribution for Project 1 in man-hours.

Group[a]	Phase 1 Outcome (Plan)	Phase 2 Outcome (Plan)	Phase 3 Outcome (Plan)	Phase 4 Outcome (Plan)	Total Outcome (Plan)
PG	31 (31)	7 (11)	7 (11)	13 (25)	58 (78)
SG	35 (34)	46 (20)	52.5 (24)	10 (8)	143.5 (86)
DG 1 (D)	21 (25)	4 (10)	19 (36)	1 (12)	45 (83)
DG 2 (TC)	23 (33)	22.5 (9)	7 (41)	0 (8)	52.5 (91)
DG 3 (CFU)	36 (56)	11 (12)	25.5 (40)	0 (8)	72.5 (116)
DG 4 (M)	19 (36)	14 (16)	24 (26)	2 (4)	59 (82)
TG 1 (D+M) + TG 2 (TC+CFU)[b]	108 (108)	61.5 (24)	0 (0)	130 (158)	299.5 (290)
Meetings	128 (140)	37.5 (104)	45 (32)	22.5 (80)	233 (356)
Total	401 (463)	203.5 (206)	180 (210)	178.5 (303)	963 (1182)

a. Legend: PG - Project Group, SG - System Group, DG - Development Group, TG - Test Group, D - Debiting, TC - Take Call, CFU - Call Forwarding Unconditional and M - Maintenance.

b. The individual times for the different groups are not available. The groups performed the system test jointly.

Creating an experience base: Based on seven projects run in the course last year, an experience base has been created. This is a typical activity carried out by people working in the experience factory or, rather, those assigned the roles defined in the experience factory. The experience base is created based on derivation of the following.

- Mean value and standard deviation of the effort spent for each group type (PG - Project Group, SG - System Group, DG - Development Group and TG - Test Group)
- Division of effort, in percentage, for each group type between the different phases
- Connection of the times to the phases and hence to the particular documents to be produced, see Figure 2.
- Calculation of meeting time as a percentage of the total time, (if the meeting time is included in the individual times for the groups it is approximated by using data from other groups in the experience base, where available.)

In Table 2, the effort (in man-hours) expenditure for each group type, based on the seven projects conducted the last time the course was given, is presented. The objective was that the times shown should represent the effort put into the activities for which the group was responsible, and the data should not just represent the work of the individuals placed in the groups initially. From the table, we can see, for example, that the project group spent 57.2 hours on average on their activities. In the last column, the meeting time as a percentage of the overall project time is shown. The aim is that this information should help forthcoming students to better plan their projects. The information in Table 2 is complemented with the effort distribution between different phases, see Table 3.

Table 2. Total time expenditure in hours for each type of group.

Measure	PG	SG	DG	TG	Meeting time in percentage of total project time
Mean	57.2 h	123 h	67 h	94.7 h	39.6%
Standard deviation	16.5 h	28.1 h	20.2 h	29.2 h	14.4%

The division of effort between the phases for the different groups is shown in Table 3. For example, it can be noted that on average 65.1% of the project group´s work is related to phase 1. The explanation of this is that the phases are primarily related to documents, and not necessarily to when the work was conducted, see Figure 2.

Table 3. Effort distribution in percentage between the phases in the software development process.

	Phase 1	Phase 2	Phase 3	Phase 4
PG	65.1%	5.8%	3.7%	25.5%
SG	35.5%	35.2%	26.6%	2.7%
DG	31.0%	27.0%	41.1%	0.9%
TG	34.2%	31.3%	0.0%	34.5%
Meetings	29.2%	26.5%	23.7%	20.7%
Total	34.4%	27.2%	22.6%	15.9%

3.6 Experience

The students often find the course difficult, since a great deal of material must be grasped in a short period of time. Afterwards, however, they usually think it has been rewarding. In particular, the students realize the benefits of the course if they start working in software development after graduation. Feedback from students working in the software industry has been consistently positive.

Based on the data collected, it can be seen that overestimation and underestimation are both represented, see Table 4. The error is calculated as:

$$Error = \left(\frac{Plan - Outcome}{Outcome}\right)$$

It is difficult to estimate the effort, but based on the experience base from the seven projects, it should be interesting to see if the estimations improve the next year.

Table 4. Relative error in the planning of the projects.

Project	Project 1	Project 2	Project 3	Project 4	Project 5	Project 6	Project 7
Total effort	963 h	598.5 h	745 h	810 h	1030 h	999.5 h	893.5 h
Error	+22.7%	+48.0%	+1.3%	-14.8%	-6.3%	+0.1%	12.4%

It is also our objective to continue to update the experience base in forthcoming years, and to formalize the experience factory further. For example, next year it is planned to provide the students with better templates and guidelines for how to collect appropriate data, and to follow up the data collection procedure weekly through progress reports to be handed in by the project leader group.

Another conclusion is that the application approach, with actually having an exchange and the opportunity to run the students' software on the exchange, is very important. This makes the project more realistic for the students, and the lecturer can also exemplify how the software is actually developed and installed in a real switch. Some of the people teaching the course have an industrial background from the telecommunications field, and the guest lectures often emphasize the issues stressed in the course. Therefore, it is concluded that the closeness to an application is an important asset, when trying to teach software engineering realistically.

4. Process assessment of the project

In the software engineering course, one project consists of reading about CMM, and performing an assessment of the projects conducted in the large-scale software development course. The students do not receive any formal training in CMM, and gain an overview through reading the documentation and by talking to the lecturers. The projects are assessed using the software development process definition, the documentation from the seven projects and by interviewing the staff from the Department involved in the course.

The actual result of the assessment is, of course, difficult to judge since the assessors have no formal CMM education and neither do the teachers. From an educational perspective, it is interesting to see how the students cope with CMM, but also how they actually view the large-scale software development course. The outcome of the assessment was, in most cases, well in line with expectations. The projects are on level 1 of CMM, although a number of key process areas are fulfilled up to level 3. The main problem is in quality assurance, and some minor problems are related to requirements and configuration management.

The conclusion is hence that the projects in the large-scale software development course are really quite similar to those in industry, i.e. many companies are on level 1 even if they fulfil a number of key process areas from higher levels in the CMM.

5. Summary

To meet the challenge of large-scale software development in an educational environment, a number of issues have been found to be important. These are:

- Closeness to an application

  This makes the courses realistic, in particular if the software can be executed on real systems.

- Industrial experience

  It is very valuable if, at least, some of the lecturers have industrial experience.

- Project work

  Some of the work in the courses should preferably be carried out as projects as this is the normal way of working in software development.

- Industrial guest lecturers

  Bringing in people from industry highlights the problems in the software industry and stresses the need for the material being taught in the course.

- Applied research

  Applied research gives good contact in industry, and provides a good basis for software engineering problems which can be adapted to the course objectives.

- Research and teaching

  The same staff are involved in teaching and research. This ensures the evolution of the courses and that the courses are up to date.

An important issue in relation to research is to let the course evolve with the research being conducted at the Department. Currently, this means two things. Firstly, the introduction of an experience factory concept in the course, which supports the students and provides a basis for research relating to learning organizations. Secondly, the changes in the course related to the experience factory form a basis for experimental research. Experimentation in software engineering is, by many for example [12], viewed as an important research method to better understand software development. Thus, experiments are used as a means in research, and the course provides a good opportunity for trying out new ideas and carrying out experiments. It is, however, important to stress that the experimentation should always be carried out so that the students gain from it.

In the future the course will be complemented with software quality assurance activities.We are currently considering allowing the students to play the roles of quality assurance personnel and let them evaluate process conformance and follow up on project data as part of the course.

Acknowledgements

The author would like to thank the companies, and the people at those companies, who over the years have contributed to the software engineering courses at the Department, in particular: Ellemtel AB, Ericsson Telecom AB, Q-Labs AB, Telelogic AB, Telia AB, Telia Engineering AB and Telub AB. The software engineering research group at the Department is acknowledged for providing a good research and educational environment. In particular, I would like to thank Björn Regnell and Anders Bruce who taught the course together with me during the autumn of 1995.

Finally, I would like to thank the reviewers for providing many valuable comments, which improved the quality of the paper.

References

[1] Basili, V., Caldiera, G. and Rombach H.D. Experience Factory", in Encyclopedia of Software Engineering, Vol. 1, edited by J.J. Marciniak, pp. 469-476, John Wiley & Sons, New York, 1994.

[2] Wohlin, C."An Effort Experience Base Experiment", Submitted to International Conference on Software Engineering, Boston, Massachusetts, USA, May 1997.

[3] Paulk, M.C. "The Evolution of the SEI´s Capability Maturity Model for Software", Software Process: Improvement and Practice, Vol. 1, No. 1, pp. 3-15, 1995.

[4] Sommerville, I., "Software Engineering", Addison-Wesley, 1996.

[5] Humphrey W.S., "A Discipline for Software Engineering", Addison-Wesley, 1995.

[6] Däcker, B., Elshiewy, N., Hedeland, P., Welin, C-W. and Williams, M., "Experiments with Programming Languages and Techniques for Telecommunication Applications", IEE Proceedings Sixth International Conference on Software Engineering for Telecommunication Switching Systems, pp. 128-135, 1986.

[7] ITU Recommendation Z.100: Specification and Description Language, SDL, Blue book, Volume X.1, 1988.

[8] Belina, F., Hogrefe, D. and Sarma, A., "SDL with Applications from Protocol Specifications", Prentice-Hall, London, UK, 1991.

[9] Student binder: MD110-project, Dept. of Communication Systems, Lund University, Lund, Sweden, 1994 (in Swedish).

[10] Rook, P., "Controlling Software Projects", Software Engineering Journal, Vol. 1, No. 1, January 1986.

[11] DoD-STD-2167A, Defence System Software Development, 1988.

[12] Basili, V., Selby, R. and Hutchens, D., "Experimentation in Software Engineering", IEEE Transactions on Software Engineering, Vol. SE-12, No. 7, pp. 733-743, 1986.