Thomas Arts
IT University of Gothenburg
Gothenburg, Sweden
thomas.arts@ituniv.se

Software Engineering and Management: A curriculum description

Abstract The curriculum of the Software Engineering and Management education at the IT University of Gothenburg is described. The education is build upon porblem based learning and uses a project orientation, in each term students spend as much time in projects as they spend in courses where the theory is taught. This educational model orginiates from the university of Aalborg in Denmark. It is used in the described curiculum to enable the students to aquire managerial and programming skills to complement their technical knowledge.

1. Introduction

The IT University of Gothenburg is a collaboration between the university of Gothenburg and Chalmers university of technology. The IT University of Gothenburg was created to enable the creation of a number of new educational programs, of which Software Engineering and Management is the largest.

This paper describes the personal view of the program manager of the education, responsible for implementing and shaping both a bachelor and a master program. Creating the programs and getting them where they are now is a collective effort of many people, staff and students, only few of them acknowledged by name at the end of this paper.

With the ongoing Bologna process in Europe, the design of bachelor and master programs has got a lot of interest. The Software Engineering and Management bachelor and master program are international programs, completely taught in English with students from all over the world. It is based on the Aalborg model [9], an educational model from Denmark that uses problem based learning in a project oriented setting. The main idea with project education is that it allows to teach skills and understanding that are hard to teach in a more traditional setting. Aalborg's motto is taken from an old Chinese proverb:

Tell me and I will forget
Show me and I will remember
Involve me and I will understand
Step back and I will act

This paper focuses on the curriculum and tries to place the curriculum in a context.

2. Vision

In the year 2000 the Swedish IT industry had not yet been hit by the telecommunication crisis caused by governments auctioning UMTS licences and an exploding IT buble. IT companies are booming and cannot satisfy their demand on graduated students.

Gothenburg has two universities which strong research groups in Computer Science and Information Systems. A tight collaboration between the universities ensures that computer scientists from both universities work at the same department. Similarly, the information systems scientists from both universities work in one department. Basically, people are located within a certain group based on their scientific interest, instead of based on their employer. Legally this is arranged by sharing faculties.

There are two five-year educations, one in computer science and one in information systems. These educations are also shared by the two universities. Thus, students can study computer science at either of them, but they will end up in a class together with students from the other university. This saves resources for both universities and at the same time it allows both of them to benefit from having IT educations. Among other advantages, it allows students in other disciplines of both universities to get their necessary IT courses in this way. Apart from these two larger educations, there are a number of other educations that contained a lot of IT related subjects.

From a student perspective, there is a difference which university one enrols, since the enrolment criteria are slightly different and when studying there are different examination criteria and even different grading systems.

Increase number of IT graduates

As mentioned before, at the end of the last century Swedish industry identified a clear demand for additional graduates in the IT area. A group of people from different companies, politicians and academics was formed to investigate how industry could help increase the number of students. Inspired by the IT University of Copenhagen it was decided to create a number of IT related master programs. Information technology is not only about programming IT systems. There are many other jobs in which one needs IT knowledge, mostly combining a certain domain with the IT domain. Therefore, cross-disciplinary master programs are necessary. These master programs should attract people with work experience as well as students in all kind of non-IT areas. A quick way to educate people for the kind of jobs available.

It was recognized that collaboration between the two universities would be vital and that quick action was required. A new organization would allow quick action and the IT University was born and quickly inhabited with five cross disciplinary master programs.

The new programs had attractive names and content and there was a clear interest from students. This motivated the creation of additional master programs as well as the start of a new bachelor program, partly to make sure there would be enough students to populate all those new master programs, partly because graduated bachelor students would also be a welcome contribution to the Swedish labour market.
The new bachelor program was designed in a dialogue between academia and

industry. There was a demand for students that had the programming skills of the already established educations and in addition a good understanding of the context in which they work, that is, understanding of the software process and the roles in that process. To put it down a bit rough: “nerds that understand that they work in a team”. Of course, one cannot expect a new three year bachelor program to offer students the same technical knowledge as a five year education as well as make them in addition aware of software processes and their role in those. Thus, the new bachelor *Software Engineering and Management* aimed to deliver technically skilled graduates that do have process awareness and development skills. However, choices had to be made and a careful selection of technical skills is included in the learning outcomes.

It was also directly recognized that the bachelor program would need a successor in the form of a master program in Software Engineering and Management. This master program would start as soon as the first batch of bachelor students finished, but not before. This was a pragmatic choice since resources did not allow to build two programs at the same time.

All these educations together educate a large number of students that are able to work in the broad field of IT.

Workplace of the future

The vision of the creators of the IT University, in particular of Ann Str ¨omberg, director of educational development, is that the IT University is a meeting place between academia, students and industry. Students and staff work side by side, sharing the same office space and work under comparable conditions. Industry contributes with guest lectures, project ideas, equipment, and so on. A special building was reconstructed to enable this vision. In this paper, the working environment is described to present a complete picture, but one could think of implementing the curricula in a different environment.

In particular it is part of the vision to see students as a kind of employees that together with the staff achieve certain goals. This is rather controversial and many visitors from academia have been wondering how that could work, whereas all industrial visitors have immediately excepted it with a ‘cool idea’ kind of response.

The underlying thought is that if you give people freedom and responsibilities, then they will also take more responsibility and in particular responsibility for their own learning.

We sketch a few typical elements of the operating environment.

Mix workplaces for staff and students

It is part of the vision that students would sit at the IT university in the same way as people would go to work. The fact that the university is their workplace would make them focus better on their learning, enable easier contact with fellow students and allow them to have meaningful discussions with their teachers.

In Sweden the employer is normally offering staff free coffee and tea. This is a rather small costs and the IT University also offers it to their students. Kitchens were installed for warming lunches in microwaves, shared by staff and students. Deliberately there was no staff room created to make sure staff and students would mingle eating lunch, having coffee together and get to know each other in several ways. And, of course, like in a normal work place, staff and students respects each other's schedules and workload. In fact more understanding is created about the workload if students see staff in different roles than only in the teacher role and teachers see students work hard in their projects.

Long corridors with closed doors where students hardly dare to enter should be avoided to any price. Instead lecture rooms, offices, discussion rooms and project places are all mixed. Staff is also mixed by having researchers, teachers, administrative personnel spread over the building, carefully making sure that they don't form small sub-divisions. The walls of all rooms are glass, transparent on the bottom part, white on the top part, such that it can be used as whiteboard. This avoids the closed door feeling, as well as that one can keep the noise outside the office.

The intended effect of all this on the student learning is that when offering an environment in which you are taken very seriously, you are also behaving more seriously. When being offered the possibility to meet with teachers in different settings, the meetings quickly get very meaningful. Experiences from working in that environment for many years is only positive.

It was computed that the cost of building and maintaining terminal rooms was at least as high as just leasing each student a laptop and have them spread around in the building with all kind of multifunctional work places where they could sit with their laptop or discuss their work. They would share that space with staff and be able to book the same meeting rooms as staff would be.

The larger lecture rooms (for about 90 students) are floors that can dynamically be used for lecturing and group discussions. Special noise isolating curtains can be drawn from the sides to transform the large space into a collection of small group rooms. This enables the teacher to frequently change from group work to more classical lectures in one and the same teaching moment (typically a day or half day).

Almost each room has a projector installed and several modern gadgets are spread around the building to be used by anyone who likes it. *Expose to modern technology*, is the idea. Industry is very willing to provide that technology free of charge, since the business case is that students that have experience with this technology will introduce it in the companies they start working for. The university should, of course, make sure to be independent and not be limited in any way when accepting industrial sponsoring of hardware or software tools.

When the director software research of one of Sweden's large companies was visiting and had seen the environment, he remarked that this would indirectly put pressure on them, since students would expect a similar environment when they would start in industry.

Agile education

New trends come and go quickly in information technology. On the one hand the university wants to provide the students with a stable base that is valid for a long period. On the other hand, many students like to learn more about the new things they see around them. Most industries want students with a deep understanding and will send them to a course in the latest trend when needed, but they also see it as an advantage if newly recruited bring new technology knowledge into the company.

The philosophy at the IT University is to quickly be able to put together and start a new program, typical goal is at most one year from idea to first student intake. The program has a program manager that makes sure that a number of courses together form an interesting education in line with the ideas of the program. Courses can rather easily be adapted or replaced by other courses. In Swedish universities courses are created and approved on faculty level, which ensures short lead times.

A consequence of creating programs that reflect certain trends is that they will also disappear after a while. That is taken into account in the design. Of course, some administration remains for students that have not been able to finish a program before it is cancelled, but there are transition periods and study advisors that make it possible for these students to get a proper diploma in the end.

Within a program, agility is achieved by specifying learning outcomes more than course content. In particular the project courses are very open in nature. One year students can develop a distributed multi-user game, the other year they write an SMS gateway in the same project. Open ended courses and projects demand from the program manager and teachers that they discuss the content each year and that they align it with the other courses in the program.

Agility in education provides a good platform to collaborate with industry. If the education likes to focus on real problems from industry, one cannot foresee in too much detail which company will contribute in the future and what their specific concerns are. Flexibility is necessary, on both sides, to transform a real industrial problem into a project that industry experiences as useful, students experience as motivating and challenging, and the university experiences as in-line with the learning outcomes.

3. Curriculum

In this section the ideas behind the bachelor and master curriculum of the Software Engineering and Management education are described. The curricula are in their second iteration at the moment of writing, thus some things we learned from the first iteration are brought forward as underlying ideas, although they actually are *lessons learned*.

Bachelor program

The curriculum for the bachelor program in Software Engineering and Management should fulfil two main learning outcomes. In the first place, the students should be able to enter a master program in Software Engineering or other IT related subject, in particular some cross discipline master programs like ”IT and law” or ”Intelligent Systems Design”. The curriculum, as we designed it, allows students to enter a computer science master program or an informatics master program, which is harder, since those master programs are often the top two years of an existing five years study with the earlier three years as prerequisites. By offering interested students the possibility to develop in a certain direction, these programs are within reach for them.

In the second place, the students should be able to get a job with their bachelor degree. Therefore, they have to be attractive to employers. In Sweden there are fewer students finishing a degree in an IT area than the demand on IT related jobs is. Therefore, the ability to write a program would almost certainly guarantee a job. At the same time, industry signals that they need people that are able to do more than programming; they have to put their work in a context. Industry would benefit from people that understand software architecture and software design issues; people that know how requirements relate all the way down to tests, people that understand what is possible and not possible to implement; people that can reflect over the software process used and can propose improvement or help make software process improvements happen.

These two main requirements are so general that any curriculum for a bachelor degree is an implementation of it. Our interpretation has been: we need to make sure the students are good programmers that do understand the managerial aspects of software development. We want our students to understand that they work in a software development process and that that process is something they influence and improve. We want the students to experience working in a software development process and therefore we let them extensively work in projects, from small to really large projects. In these projects they have roles and develop and experience skills to manage such process.

The design of the curriculum is highly influenced by the Aalborg model [9, 10], an education model originating from an experiment starting in 1974 at Aalborg university. The model is now recognized as an effective education system, producing highly qualified engineers.

3.1 Theme structure

An academic year consists of two terms; the Software Engineering and Management bachelor program consists of six terms. In line with the Aalborg model, each term is given a specific theme. Among other things, the themes constitute the professional profile of the curriculum. Half the time in a term is spend in one project course, the rest of the time is used for theory courses that support the project. Some theory is given before the project starts, some runs in parallel and feels from a student perspective as 'just in time'; they struggle with a problem and get the theory to address it, or theory is deliberately given after an experience such that absorbing the theory is easier (for example project management after two projects in which students find out themselves that something is missing).

For each project students form groups. The group formation is an important part of the educational model: students have to create their own groups in order to take responsibility in that group. In the beginning of each term, one day is assigned to present the theme, the structure of the term and to create the groups. There is a clear deadline when all groups need to be formed and students are left to do this with only one rule to follow: 'no group is fixed until every student has found a place in a group'. The first semester is an exception to this rule that the students determine groups themselves, they are 'randomly' put in a group where staff tries to create a mixture depending on their background.
A group consists by default of six students, but depending on the total amount, groups of five or seven may appear. Each group is assigned a supervisor, which is a teacher that supports the group during the term. Support consists of technical help with the subjects taught in the term; on-demand teaching of subjects students need for the project, but are not part of the term; reading and commenting on written artefacts; and training in social skills, help in getting the group be more efficient, teach how to resolve conflicts, etc. The Aalborg model describes the role of the supervisor in more detail [9].

During the education, the projects increase in size. The students evolve from working in groups to working in projects. If students do not collaborate in the first term, then passing the project is still possible, since in principle the workload allows an individual to carry out the project alone. Projects quickly get larger and the fourth project spans over the whole class and a lot of collaboration efforts, planning and responsibility is needed to make it work. The software development processes used are reflected in the size of the project. Students improve their software process from term to term.

The six themes of the bachelor program are listed below with a short summary of the content of the term.

Programming, program and data

The professional profile of this theme is that of the programmer. In the project we focus on getting requirements from a customer (interviewed by a teacher as an example of requirement elicitation), who wants some simple information system for which a database with a graphical user interface seem a natural choice. Some algorithmic aspect is involved as well, but that is mostly rather limited. The project is supported by three courses. A course that combines theory of databases and user interface design from a software engineering perspective [11, 15]. Relational algebra and SQL are ingredients, but normalization was removed from the curriculum recognizing it as too advanced. The user interface approach emphasis on a method to get user interviews down to usable software. It gives a gentle introduction to software requirements and puts the end-user in focus, which is challenging and motivating for the students. In line with the Aalborg model, an open problem is presented and given 15 groups of 6 students that start with that problem, you will see at least 15 different solutions presented by the end of the project. That in itself is a learning experience for the students, to see what others made from the same customer interview.

The second and largest course is a Java course. Java is used to implement part of the project, but we also have traditional assignments in the Java course to make sure every students spends time programming and gets the help needed.

The third course is about software processes and is given after that the project is finished as an intensive two week course. After completion of the course, the students have basic knowledge of software processes. They know the different phases of a software process and the terminology used in software process literature; they understand the V-model and are able to instantiate a general software process for the needs of a specific project. The students are able to relate different activities in Software Engineering to different phases in the process. The notion of iterative and agile processes are also presented, since in the term after this course, the students are expected to use an iterative process in their project.

Although the students work in groups of six students in the project, the size and complexity of the project is such that some students could equally well implement the project on their own in the given time. Writing the required report in addition is challenging, but would be possible. That is, they can benefit from working in a group, but when things fail, there is time for the group and their supervisor to concentrate on the group dynamics and nevertheless meet the project deadlines.

System development

The professional profile of this theme is that of the programmer or, a more modern term, designer. Skills in programming are further developed as well as that object orientated analysis and design are introduced. In this term students are supposed to hand in several documents, among which the Software Requirement Specification and Software Design Specification. In order to make sure that many aspects of these documents are covered, we let student groups find their own industrial partner and get project ideas from them. This works highly motivating for the students and there is always one student in a group that has a sister, cousin or neighbour that needs a system to be developed. The enthusiasm and naivety of young developers guarantees that the projects they propose virtually always are challenging and require long lists of requirements. The project supervisor selects a few requirements that they have to design and implemented. Normally between one and two iterations are managed by the students. The full system is never implemented within the course but becomes a summer job or hobby project afterward.

The project is supported by three courses. A course in technical analysis and design [12] in which students learn both a bit about the development process as well as practical UML diagrams.

A course that supports them with some mathematical background in the form of set theory, recursion, matrices, graphs and elementary algorithms like sorting. The choice was made not to introduce much mathematics in the curriculum and together with a bit a relational algebra, this is one of the few courses where these skills are developed. It builds the fundaments for later courses in which recursion and algorithms are important ingredients.

The third course is in quality management. The course makes students aware of the role of quality assurance within a software process and presents the students with the novel idea of testing software before you hand it in.

The size of the project is of such a nature that students have to collaborate to pass the project. There is too much material to hand in and too much work to be performed to make it possible by one or two students to do all by themselves. The frustration the students are confronted with is that they normally want to be in control of everything and find it hard to define roles and divide work. We let students experience this frustration to make it easier for them to appreciate the course in project management in the next term.

Software Architecture for Distributed Systems

The professional profile of this term is that of a software architect. At the same time we continue to improve the skills of programming. The project is given and often carefully selected by talking to industrial partners in domains where they need distributed, fault-tolerant systems. The students concentrate on designing an architecture for such a system and implement it in the language Erlang in combination with parts in Java. Typical examples in this term have been distributed multi-user games and file sharing applications based on bit-torrent technology.

Supporting courses are a course in software architecture and a course in the programming language Erlang. The language Erlang is selected because of the author's personal bias, and fits well from an educational point of view. By using Erlang we present a functional language to the students that supports a simple way of writing concurrent and distributed applications. It gives the students a tool to simply write code for multi-core computers or large cluster machines. It provides students with a new view on what is possible with modern languages. In addition, the local industry loves to see students graduating that master this language. The programming course is provided as an industrial course, i.e., in one intensive week they work eight hours per day with the language and are supposed to be able to use it after that. The students find it tough, but they appreciate that they are exposed to this real life experience, where people with a busy job are send to a course and are supposed to use their knowledge when they return.

The third course, spreading over the whole term, is a course in project management. The students learn to divide the work, assign roles, run effective meetings, estimate risks and so on.

The size of the project is roughly as large as the second term project, but the technology is more complex. Students need to collaborate, but because we force them to take one of six different roles, they manage well in this collaboration. Students are informed before the project that they will be graded according to the work they performed in their role. Thus, a tester is graded by the quality of the tests and the test code and a software architect is graded by the quality of the architecture. In addition, for all students it is taken into account how well they performed in the group, whether they contributed in other ways to their project as purely by their role, and last but not least in their coding effort. Independent of their role, they have to write a substantial part of the code.

Industrial IT and Embedded Systems

The professional profile of this term depends on the role the student takes in the project. After running three projects, the students are eager to extend their boundaries and take a role in a larger project. We put the whole class, somewhere around 50 students, together in one big project. This project is a mock-up of a real industrial case for which we use one of the staff members as a customer. It is important to have a project that feels like real, with a customer one can go to in case requirements are unclear (which they always are) in case deadlines are not met, or better features have to be prioritized. We can, however, not really run just any industrial project, since such a big undertaking also requires quite a lot of planning to make it work reasonably well.

Typically communication is getting a major issue and miscommunication the rule. Some students will feel happy just to implement something or be responsible for the design, whereas some other ambitious students want to be project manager, software architect, quality manager or technical expert for a certain area. The roles are divided by the students themselves, staff helps in explaining the roles. Preferably, students shape task descriptions for each role and they have together to consider whether they cover all project aspects.

The project is provided with a hardware budget and students are supposed to select hardware for their project and build the software for it. One year they buy dumpers and cranes in the local toy shop, the other year they may decide to have industry sponsor their hardware and do even more fancy stuff. We have had projects to build active safety in cars, lane departure warnings, automatically stopping for red lights, etc. We also have had a complete logistics chain for a coffee rostery, including a ship to transport the beans, a crane, an oven and trucks to transport beans between places.

Each year one may concentrate on different sensors and therewith adapt the course material to those sensors. Image recognition is one way, radar signal analysis is another one. The supporting course for this is *Development of Embedded Systems* in which they also learn the language C, just to make sure that they have also seen that language in their education.

Embedded systems have string non-functional requirements on safety and correctness. Therefore, as second supporting course we introduce verification and validation to the students. The V-model is revisited and students learn how to construct test cases for each level. They are also made familiar with formal methods as an approach to find design errors early. Using a model checker is challenging for second year students and experience learns that only a few students actually master it, but they will use it in their project. In addition we teach QuickCheck [1, 5, 8] as a model based testing technique, combining formal methods with traditional testing.

The third supporting course is run after that the project is finished. It is a course in Software Process Improvement and comes just right after all frustration that the students are confronted with in the first time they run a large project. Of course things go wrong and putting that into context and explaining how one can measure and improve is very much appreciated.

Changing development process

The professional profile of this theme is that of a manager that is confronted with new technology and has to change the organization adapting to that technology. We got this profile from Ericsson, who approached us with the question whether we could help with educating their managers. Ericsson was working in code centric projects, i.e., the code was the central valuable thing produced. Of course there are all kind of documents, but in the end, questions like 'how much code is ready?' and 'how many errors did we found in testing the code?' were central. If problems arise, managers assign more developers to solve the problem, working on the code, of course. With model driven development, the code is either automatically generated or at least produced later than in the code centric projects. Progress is no longer measured in lines of code and errors may actually be detected in the model instead of by testing the code. Altogether, this requires a dramatic change in how one organizes a project.

Ericsson like to see the managers sit together with our bachelor students and run a project together. The big advantage we saw was that the students would learn a lot from these professional managers. The advantage Ericsson foresaw was that the managers would feel slightly impressed by the way those young students would install different tools, search the internet for information and quickly write a small program to perform an easy task. A detailed report about those two worlds meeting is described by the teachers of the course [4].

The project typically pairs up bachelor students and experienced people from industry. Together they have to implement a given standard by using a model driven approach. The industrial engineers contribute with the domain knowledge and are therefore a necessity for each group. As a consequence, we have to limit the amount of course participants for the project. The supporting course on model driven development can, though, be taken by a large number of students.

The other support course in this term is a course in change management, focussing on the theory of handling change.

An additional new thing for this term is that students are supposed to write about their project by means of a scientific article. Not necessarily content wise, but the form to restrict oneself to only a few of the results of the project and present them in a paper of only 15 pages with references to literature etc. They get a very short introduction to the (electronic) library and are helped in the writing process by their supervisors. After having written four project reports, students embrace this new form, where after they realize that writing 15 pages instead of 100 is at least as hard.

Thesis project

The professional profile of this theme is purely individual and determined by what the student likes to work on after receiving the bachelor degree. Most students find a thesis project in industry.

We see the students as ambassadors of new ideas and technologies when they start working in a company and we indirectly help companies to get up-to-date knowledge. Therefore, we require students to find a scientific article in which they apply a certain method or technique in industry and ask them to copy the work described in that article during their bachelor project. For example, one student selected the published fault slip-through method used within Ericsson and he tried the same measurement in a completely different company. This was embedded in a more practical project in which the company wanted the student to design and execute tests [3].

Apart from helping to spread scientific results into industry, this idea also provides researchers the possibility to collect data for their research, it provides students with a little understanding of what research is about and it may in some cases even result in a publication. On average two to three bachelor theses per year are accepted for a workshop or national conference (e.g. [2, 3, 6, 7, 13, 14]. Of course, after that the original thesis is written and a new version is conducted together with the supervisor.

We hope that, as a side-effect, students are positively approaching research results and will later in their career now and then have a look at scientific results or return to the university to start research projects with their former teachers.

The form of the bachelor thesis is an article and the examination process is comparable with the review process of a conference. All theses are handed in at the same date and at least two teachers read and grade it. A joined review discussion enables the teachers to guarantee grading independence of a specific supervisor and to ensure overall quality of the theses. The article form of max 15 pages makes this process manageable, since the reading effort is limited.

There is only one supporting course in this term in which the students are introduced to research methods and develop writing skills.

Master program

When bachelor students get to a level in which they submit articles to workshops and national conferences, one may have a tough time to attract them to a master program. Therefore, the master program has to offer something the students feel they would like to learn more about. At the same time, we would like to get students from other bachelor programs all over the world to come to our Software Engineering and Management master program. Thus, we have an unavoidable repetition of basics in software engineering and project work. Experience is in particular that certain students have no project experience at all and putting those students in a group of students that have worked in groups for three years, makes it difficult for both. There will certainly be good pedagogical arguments that this mix is extremely useful for the learning of both parties, but useful for learning and attractive to students are not always in line.

We recognized the more general problem of having an education that is based on two major skills or subjects, like management skills and technical skills and the admission of students with a clearly stronger background in one of the subjects. Similar problem is faced by an education as ”Law and IT”. We have seriously contemplated to have two tracks in which the students from master programs with little software engineering in the curriculum would get one term to specialize in that, whereas students with a strong software engineering background would jump to the second term immediately. Effectively we planned to run each term twice per year instead of once. This would also simplify the intake of students. We could allow them to enter after Christmas, which solve a practical problem that students encounter in the new flexible education system: We require a bachelor degree for entrance, but the students have to apply in May for admission in September and they will earliest get their diploma in September. The term that students with software engineering background skipped, was to be compensated by additional freely selectable courses in the technical areas. Because of problems in recruiting staff, we never came to implement running each term twice a year.

The four cornerstones of the master education are

- close connection to our research,
- involvement of industry,
- focus on both people and technology,
- international.

In the bachelor program, we had a theme for each term that brought in a specific professional profile. In the master program, we fix a number of professional profiles that closely correspond to the research we perform. We defined the themes: **Software Quality**, **Software Architecture**, and **Software Management**. Within these themes there are several professional profiles. For example, 'Software Quality' is broad and can contain the profile of tester, but also of quality manager or formal verification engineer. These three themes are introduced in the first term, have an introductory course in the second term and are specializations for the last two terms.

Research

We involve the students in our research projects by introducing them to our research in the first term and then making them part of the research group during the second term (where they choose which group/research they like best). Students can typically help in different ways to contribute to ongoing research and their master thesis is often directly connected to ongoing research projects.

Industry

The program contains a large number of guest lectures provided by industry to make students aware of the professional profiles they can choose from. All projects are based upon challenges given by industry and quite often the projects are performed in close collaboration with industry. The master thesis project is almost always carried out in industry, normally arranged by the students who therewith widen our network.

People and Technology

We carefully select the bachelor students that enter the program, but 'people skills' are often not that visible in their CVs. Many students have had a kind of project management course in their education, but the maturity of working in a team differs a lot from student to student. Even if the students have some experience from working in teams, they seldom have worked in an international team with all cultural difference that that brings. Therefore, we start the curriculum with a project that very much focuses on people skills and which contains a modest technical challenge. We inform the students that they should focus on learning to work together in a project and support them by theory and practical advices. If we happen to have a good mix with our former bachelor students, the groups learn from them as well as that our former bachelor students can develop their skills by the fact that they are confronted with a team that has a different background.

International

We recruit students from all over the world. Swedish education is free, even for foreign students. The government is planning to change this, which will impact the student applications. At the moment we get several thousands of applications. Selecting the right type of student from this pile is challenging and a lot of work. The cost of selection should be seen as an investment; mistakes are unfortunate for students in particular when group work is affected and expensive for the university in extra hours necessary for teaching and coaching.

We try to benefit from the fact that we have a group of international students. The students get a possibility to work in multi-cultural teams and we address that with theory and examples given in class. We teach the students how projects are run according to the Swedish model, but discuss differences with other models.

An international team offers also possibilities such as comparing software processes of companies in several countries. There is always a student that can translate a Chinese process description and another that can explain the content of a Russian process description. They can call and interview people in the local language and one can use the student network to get in contact with companies abroad.

4. Discussion

We have experienced the creation of a program in Software Engineering and Management according to the Aalborg model. We have had help and coaching from the University of Aalborg in the first few years, which was necessary in order to learn the model in practice.

People have to buy in to the Aalborg model, this is for most teachers a change. Starting a new program with the possibility to recruite people to the program simplified a change process, since we recruited people that liked this teaching model. Even so, one cannot sit back and relax, as part of a large university, many decisions are taken that effect the teaching philosophy. Just to mention an example, the idea of having group rooms for students is cornerstone in the model. Clearly a conflict arises when group rooms have to make place for terminal rooms, or when courses become the administrative unit instead of programs causing support courses to drift their own way. A constant involvement of management is required to enable the different pedagogical model.

We have established a teacher discussion group which focuses on the teaching philosophy and in which we learn from each other and from invited experts to constantly improve our education. This is essential for new teachers but also a strong support for more experienced teachers.

The economical models of more traditional courses compared to the project with supporting courses do not differ much. The project courses are more scalable and less vulnerable when only few students apply. Each group of six students has a supervisor, which makes it easier to increase or decrease staff involvement. In our economical model, the project is considered a large course and we get assigned hours for that course, which we distribute to the supervisors. If a supervisor has two or three groups in the same project, then the additional overhead for the teacher is little and the hours assigned suffice for the job. The necessity of group rooms for students is balanced in our case by a smart design of the building in which lecture hall and group rooms are integrated. Since lecture halls are seldom fully utilized, the costs are not significantly different.

Students that enter the education often are attracted by project work. We state that people skills are as important as technical skills. This has to be reflected in the examination as well. That is a challenge. Traditionally both teachers and students have much more experience in examining students by just focussing on technical competence. Now we want to give a high grade to a student that moved a group forward, that actively helped the group to work as a team, a good project manager. If this student has hardly any code written, has no documented design solutions, then teacher and students find it hard to find the right motivation for a high grade. Partly this is a change in mentality and a matter of well formulated learning outcomes. On the other hand, we help teachers and students by requiring artefacts like minutes of the meetings, risk estimations and follow-ups, but also more technical artefacts like test cases and test reports.

Students should be graded individually, even if they perform the work together in a project. Groups that work very dynamically often cannot account for who exactly did what. Thus, in groups that really work well it is harder to see individual contributions. What one certainly wants to avoid is to make a grading system that is contra-productive to group dynamics. We copied from the Aalborg model that all project members are responsible for the total outcome of the project. The examination model is that of an oral examination at the end of the project in which all students get questions over the whole project and some specific, more detailed questions about their own contribution. Note that each project contains an implementation and that all students have to contribute to the code. We always ask them which part of the code they are especially proud of and to explain that part of the code. The Aalborg model furthermore advices to examine whole project group at the same session, such that students hear the answers of their fellow students. If this is done in the right way and when a proper written summary of strong and weak points of each individual student are given after the oral exam, then they understand why they got the grade compared to the other students. It is namely not always obvious to students that grading is only partly related to the amount of work put into the project. Some students feel they have done much more than certain others and after an examination the group must be confident that the grades are not randomly selected. Project grading is difficult and it took us time to find our form. Specifying helpful learning outcomes and defining project roles against which students are evaluated simplifies matters a lot. We use a board of three people to examine the students: the supervisor of the group, who asks most questions and drives the examination forward; the course responsible, which often has the official role as examiner of the course and should sign the final grade; and an experienced person that has been in similar examinations before, but has not been involved in the project. In Aalborg they even use external industry relations to join the examination, which seems like a very good quality control and interesting way for industry to acquire knowledge about the education.

Our students perform well in industry and we have build a positive reputation among local industry. We are constantly approached with request for bachelor and master student projects and can easily get hold of guest lectures from industry.

We have only examined two batches of master students so far and we are aware of two of those students that started a PhD education (abroad). We do still have active collaborations with many of our students that found a job in industry and now return for guest lectures, presenting project ideas, or collaborate with us in our research projects.

Acknowledgements

Many people have contributed to the creation of the Software Engineering and Management programs as they exist today. I am grateful for the devotion of both staff and students to the program and I would like to mention a few people in particular; without them, the programs would not exist. Ann Str ¨omberg was director of educational development of the IT University and the strong driving force behind new pedagogies, brilliant design of the building and carrying out the visions as mentioned in this paper. Lena Holmberg, Lars Mathiassen and Per Stenstr¨om have been immensely valuable in laying the foundations of the education and a constant advice and support during the first years.

Lars Pareto was among the first teachers of the course and as a real adept of problem oriented learning, he has contributed a lot with forming the curriculum. Many of today's courses are based upon his foundations. In particular his design to get a whole class involved in one project is outstanding.

People move on and program managers change. I am grateful that Helena Holmstr¨om Olsson has taken responsibility for the bachelor program and improved it even further. Thomas Lundqvist managed the first difficult years of the master program without many resources to help him. Miroslaw Staron got the master program to work well with happy students and good results.

A special thank also for the support from all our partners from industry. Without their support we would not be able to realize our vision. In particular we are grateful to Anna B¨orjesson Sandberg and Mats Lind´en for their long term commitment to our curriculum and research.

References

1. Thomas Arts, John Hughes, Joakim Johansson, and Ulf Wiger. Testing telecoms software with quviq quickcheck. In *ERLANG '06: Proceedings of the 2006 ACM SIGPLAN workshop on Erlang*, pages 2–10, New York, NY, USA, 2006. ACM.

2. Simon Aurell. Remote controlling devices using instant messaging: building an intelligent gateway in erlang/otp. In *Erlang Workshop*, pages 46–51, 2005.

3. Jonas Boberg. Early fault detection with model-based testing. In *ERLANG '08: Proceedings of the 7th ACM SIGPLAN workshop on ERLANG*, pages 9–20, New York, NY, USA, 2008. ACM.

4. Anna Borjesson, Lars Pareto, Ulrika Lundh Snis, and Miroslaw Staron. Continuing professional development by practitioner integrated learning. In *OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, pages 897–907, New York, NY, USA, 2007. ACM.

5. Koen Claessen and John Hughes. Quickcheck: a lightweight tool for random testing of haskell programs. In *ICFP '00: Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*, pages 268–279, New York, NY, USA, 2000. ACM.

6. Carl Magnus Olsson Helene Dahlberg, Francisco Solano Ruiz. The role of extreme programming in a plandriven organization. Transfer and Diffusion of IT for Organizational Resilience, IFIP WG8.6, Ireland, June 2006.

7. Emil Hellman. Evaluation of database management systems for erlang. In *ERLANG '06: Proceedings of the 2006 ACM SIGPLAN workshop on Erlang*, pages 58–67, New York, NY, USA, 2006. ACM.

8. John Hughes. Quickcheck testing for fun and profit. In *9th International Symposium on Practical Aspects of Declarative Languages*, pages 1–32, 2007. Springer.

9. Finn Kjersdam and Stig Enemark. *The Aalborg Experiment*. Aalborg University Press, 1994, reprint 1997. http://adm.aau.dk/fak-tekn/aalborg/engelsk/index.html.

10. Anette Kolmos, Flemming K. Fink, and Lone Krogh. *Aalborg PBL model : Progress, Diversity and Challenges*. Aalborg University Press, 2004.

11. Sören Lauesen. *User Interface Design*. Pearson Education, 2005.

12. Craig Larman. *Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, October 2004.

13. Francesco Cesarini, Lukas Larsson, and Michal Ślaski. From HTTP to HTML: Erlang/otp experiences in web based service applications. In *ERLANG '06: Proceedings of the 2006 ACM SIGPLAN workshop on Erlang*, pages 52–57, New York, NY, USA, 2006. ACM.

14. Lars Pareto Erik Sjoesten-Andersson. Costs and benefits of structure-aware capture/replay tools for graphical user interface testing. In *In Proceedings of the Sixth Conference on Software Engineering Research and Practice in Sweden (SERPS06)*, 2006.

15. Jeffrey D. Ullman and Jennifer Widom. *A first course in database systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.