Dimensions of Software Engineering Course Design

Mario Bernhart
Vienna University of Technology
mario.bernhart@inso.tu.wien.ac.at

Thomas Grechenig
Vienna University of Technology
thomas.grechenig@inso.tu.wien.ac.at

Jennifer Hetzl
Vienna University of Technology
jennifer.hetzl@inso.tuwien.ac.at

Wolfgang Zuser
Vienna University of Technology
wolfgang.zuser@inso.tu.wien.ac.at

ABSTRACT
A vast variety of topics relate to the field of Software Engineering. Some universities implement curricula covering all aspects of Software Engineering. A number of other courses cover detailed aspects, e.g. programming, usability and security issues, analysis, architecture, design, and quality. Other universities offer general curricula considering Software Engineering in few or single course only. In each case, a course set has to be defined which directly relates to a specific student outcome. This work provides a method for categorizing and analyzing a course set within abstract dimensions for course design. We subsequently show the results of applying the dimensions to the course degree scheme in use. The course design dimensions can also be related to the student outcomes defined in SE2004 CC Section 3.2 [10].

Categories and Subject Descriptors
K.3.2 Computer and Information Science Education.

General Terms
Management.

Keywords

1. INTRODUCTION
The field of Software Engineering (SE) has become enormously complex, integrating a high number of elements, e.g. technical aspects, management skills, and especially, people issues. The way in which SE is taught to Computer Science students is mostly naturally directly related to this degree of complexity, and therefore, a straightforward or teacher-centered approach for SE course development does not exist. As software products are among the most complex of man-made systems, the most essential of their inherent properties are complexity, abstraction, and adaptability. Although these issues are not easily addressed in SE itself [3], they must be represented as closely as possible in educational SE approaches. Thus, SE teaching requires a highly sophisticated approach. SWEBOK defines ten knowledge areas to be part of the field of SE: software requirements, design, construction, testing, maintenance, configuration management, engineering management, engineering processes, engineering tools and methods, and quality [1]. According to SWEBOK, SE is related to seven disciplines: computer engineering, project management, computer science, quality management, management, software ergonomics, mathematics, and systems engineering [1]. Consequently, the design of SE course must consider the following questions to cover a wide variety of disciplines:

- Which of the above knowledge areas and related disciplines can be considered in a single course scheme?
- If more than one knowledge area or discipline is considered, which area or discipline is focused?
- Which of the above knowledge areas and disciplines may be handled in separate specialized courses?
- Are collaboration and cooperation among different courses reasonable?
- Which course order is reasonable?

The Software Engineering Curriculum Guidelines 2004 for undergraduate degree programs in SE provides advice for defining the content of SE curricula [10]. The authors provide a considerable number of valuable guidelines on curriculum design and pedagogies, although advice for the design of a single course or a set of courses with a common focus or goal is missing. Since SE is a highly applied science, the didactic concept of SE teaching must emphasize on application as well to achieve the maximum learning success for students and to cover the fundamental aspects of the field for teachers. The concept of teaching and the course outline ideally mirror the current developments in SE. Therefore, the didactic concept equally undergoes permanent evolution.

2. RELATED WORK
Some work on the foundations and goals of SE education exists that provides a framework for successful course design [1][2][12][15]. Although details of courses may vary, the major issues for all courses are to teach state-of-the-art and state-of-the-practice techniques, and to provide a project framework matching real-life projects as closely as possible - within academic constraints and limitations - for practical application of these concepts. A major area of SE education research deals with defining and developing SE programs on the graduate [7][16] and undergraduate level [9][17][19]. The evaluation principle focuses optimal score of student skills and coverage of acquired techniques. In [7], courses on different levels are compared, with a more complex and realistic course on graduate level, emphasizing project work to enhance students’ SE skills. Collaborative courses may improve the training curve, since prerequisites for subsequent courses are covered only once [13]. As soft skills play an important role in SE education, they are

1 Authors appear in alphabetical order

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICSE’06, May 20–28, 2006, Shanghai, China.
Copyright 2006 ACM 1-59593-085-X/06/0005...$5.00.
considered in rather methodically or technically focused SE courses [18]. Other research efforts emphasize on novel techniques, e.g. the Personal Software Process (PSP), to assess and describe SE courses. Many examples reporting experiences with PSP exist, e.g. in [4], but only one references UP [14]. Robillard et al. describe a course considering a customized UP version, and present empirical data based on a sample of 30 students. Yet another research direction relates to the degree of similitude with real-world projects to render SE education more naturalistic. The purpose of these efforts is the optimal preparation of students for industrial projects. The evaluation scheme allows to assess courses having re-introduced change of requirements or staff fluctuation to mirror real-world problems and to overcome academic deficiencies in SE course design [4][6]. Other have reported evaluation results for cooperation between academic and industrial instructors to improve quality in SE education [8][11]. For example, interns have reported stakeholder communications as a crucial aspect of SE to be taught in courses [11].

3. DIMENSIONS OF VARIATIONS

This section discusses the dimensions, in which lecturers have to decide how to design specific SE courses. For each dimension we will discuss the characteristics and the advantages and disadvantages for either end of the range. The dimensions and their characteristics are derived from the experiences with the design of our courses described in section 4 as well as from course descriptions from the referenced work.

3.1 Discipline focus: full vs. partial coverage of topics
The first and most critical decision is which parts of the software lifecycle are included into the course. Typical parts, which may be handled separately, are: business processes, requirements, design, implementation, test, deployment, maintenance, and evolution. Furthermore, there are many related activities and disciplines, which are necessary for successful software engineering: project management, quality management, teamwork, security, and usability engineering. Each part and disciplines are included within a single course, which raises the effort for the students and the supervisors. The more parts and disciplines are included, the more connections and dependencies between lifecycle parts and disciplines can be demonstrated to and experienced by the students. Focusing on single or few parts of the software lifecycle helps to get many experiences in this part. Concentrating on few methods and process steps within a single part will help to get the students awareness for dedicated details.

3.2 Development focus: Initial development vs. maintenance and evolution focus
The majority of real-world projects are not completely new developments, but rather an evolution or maintenance of an existent system. In contrast to a new development, an evolutionary project focuses on other demands and skills. Analyzing an existing system in various aspects is in the foreground, while new developments demand more creative skills. The boundary between software evolution and new developments is very flexible and can be set depending on the required course level and objectives, e.g. a maintenance and evolutionary basis is usually featured in advanced courses.

3.3 Process Model: Single in depth vs. overview of many

There are several process-models used in practice: (Rational) Unified Process, Extreme Programming, Microsoft Solution Framework, SCRUM, and variants of the waterfall model. Some of these models have similar characteristics and some totally differ in characteristics e.g. Extreme Programming compared to a classic waterfall approach. Teaching a single model allows to learn about the model and practice the model in depth. Preparing students for the industrial practice, however, requires teaching more than one of the models. Using several models in a single course provide a good overview about the most important characteristics of each model. Another possibility is to allow any model within the course and to leave the choice to the student. This requires a profound knowledge of many models from the supervisors. If the students have to attend more than one Software Engineering course it may be easier to alter the models used in the courses and providing in depth knowledge and experience as well.

3.4 Deliverables: Process vs. product focus
Course deliverables should answer two questions: what has a student done (product focus) and how has the student achieved this result (process focus). The product focus allows a more objective grading than the process focus. Products can be assessed following “hard” criteria like completeness, correctness, presentation style etc. The process focus allows determining the compliance of the student to the required process.

3.5 Organization: team vs. individual
Using individual assignments allows straightforward grading of each student’s performance but it is time-consuming to check each student’s deliverables. Furthermore it is not possible to assign complex problem statements to individuals. Complex assignments require the collaboration of a team of students. Team assignments enhance the learning experience about team processes like communication, coordination and conflict management. Grading individuals within teams is a complex activity, though [5][20].

3.6 Technology focus: universal vs. specific
Using specific software technology, which is vendor-dependent in most cases, allows education of students in this technology in depth. This is very valuable for students applying in companies after having finished their degree, if companies are looking for expertise in the selected technology. However, as there are many technologies available, each of them with certain strengths and weaknesses, there is no standard technology and each company is looking for a different expertise. It is impossible to teach all available technologies within an undergraduate and graduate program. Using universal or neutral technology (like experimental technology or technology with low industrial use) helps to focus on technology-neutral aspects, e.g. object-oriented principles like inheritance and polymorphism, rather than on the technology-specific issues, e.g. differences between Java and C#. However, using universal technology does not necessarily help students to improve their professional skill table.

3.7 Tools: Hands on vs. Tool-support
Today software engineering heavily depends on proper tools. Tools increase productivity and help to focus on the important issues rather than on how to document or administrate artifacts.
Tools are very helpful, if experienced users use them. Getting used to tools causes considerable administrative overhead which exceeds the beneficial effect for inexperienced users. Software Engineering students are inexperienced with most of the available tool categories, e.g. modeling, versioning, requirements- and issue tracking. Introducing many tools causes a high amount of learning effort. Students spend much time on getting used to a tool rather than spending time on understanding and solving the assignment. Typical deliverables are well solved in terms of the tool but lack valuable content. Waiving tools helps to direct students’ attention onto the assignment, e.g. by withholding a modeling tool, but let students draw diagrams on paper. However, Software Engineering education should prepare students for industrial reality. They have to acquire some understanding which tools are suitable for different activities and should know the capabilities of state-of-the-art tools.

3.8 Assignment focus: Independent exercises vs. Projects

Single independent exercises allow concentrating on selected issues. Providing the students prepared artifacts or code for each exercise keeps the student efforts within reasonable boundaries. Failing to complete one exercise does not influence the following exercises. Using a single system, which is provided to the students, for the whole course makes exercises building on each other possible. Each exercise may focus on a different part of the system or system documentation. A whole project or project parts are the most realistic approaches teaching software engineering. Students learn what effects are caused by incomplete or buggy specification and design documents. They see the connections between the artifacts in a software lifecycle. However, doing a whole project within a single course is the most time-consuming variant for the students as well as for the supervisors. If students fail to deliver a minimum quality at the beginning, succeeding phases may receive a reduced quality as a work basis. This may reduce the learning experience in later phases.

3.9 Exercise domain: administrative software vs. technical software

Software plays an important role in nearly every technical as well as business field. Embedded systems, real time systems, and ubiquitous computing on the one side, management information systems, enterprise resource planning, web services, and service oriented architecture on the other side are buzz words concerning fields, where software engineering plays an important role besides technical or professional issues. For each field different approaches to software engineering are necessary: different architectures are designed, different development environments are used, different methods (e.g. for testing) are necessary to achieve different quality attributes. The exercises or projects in a software engineering course may focus on one of these fields to teach special know how. Or the exercises or projects are designed in a way, that students get universal know how in methods applicable to many of the fields.

4. CURRENT COURSES IN RELATION TO THE DIMENSIONS

The dimensions were defined by our experiences with SE courses and related research. Evaluation of selected courses currently held at our department shows an application orthogonal to the dimensions. This section provides an overview of two courses, SE1 and the subsequent SE2, and their evaluation against the course design dimensions. Our courses are embedded in a general Computer Science program, which is not focused on SE solely. Therefore our courses have to cover a large variety of topics, starting from object-oriented analysis to design, coding, and quality assurance issues. Of course, covering such a wide range of topics restricts the scope of the course and has kept us from going into other, less fundamental but equally important, areas of SE, e.g. risk management.

4.1 Software Engineering 1

Software Engineering 1 (SE1) is an intensive twelve-week base-course for undergraduate students in their third semester. The teaching focus is on developing team skills by elaborating a partly given software project in a group of six. The course is divided into two phases: phase A is a two-week introduction and selection phase, and phase B is the ten-week main part of the course. Phase A is a tutorial non-team-phase followed by a selection test for phase B covering three topics: business modeling, database design and programming. These topics are prerequisite knowledge for the course. It is essential for the course that every student outcomes these competences before entering the team phase. Students that have successfully passed phase A were assigned to teams of six to develop a common software project with given requirement specification documents and a technical foundation. The main objective is working following a customized “iterative and incremental” process model. In addition to the teamwork, each student has to solve two individual assignments during phase B. The didactic value is in researching and reflecting on SE-related topics, which were not explicitly part of the course. Examples of these topics are: stakeholder analysis, White-Box testing or risk analysis.

4.2 Software Engineering 2

As this course is the direct successor of SE1, obligatory for Software Engineering majors, the teaching objective is again on further developing team skills and following a process model. The main difference to the course of SE1 is the underlying project and the type of team allocation. The students are composing the teams on their own authority. These teams of five to seven have to find at least two industrial projects with a real customer to base the course on. These project proposals must be submitted before the course. The course supervisor reviews the proposals and approves or rejects the respective proposals. All project proposals must meet defined minimum standards, e.g. the amount of work must be in an appropriate range for this course. Software Engineering 2 (SE2) has neither an introduction phase, nor have the students to qualify for taking the course. Even if not a prerequisite, it is though highly recommended to complete the course of SE1 before attending to SE2. SE1 prepares the student in a protected environment to allow focusing on teamwork, SE methods and products without struggling to sustain the high pressure in a commercial environment. Most organizational aspects of SE2, despite of team allocation and the project, are the same as in SE1. Roles and supervision are equal and the assignments are almost equal. The main focus is on realizing a project in a commercial environment rather than in the protected environment provided in the preceding course. Regarding to the common process model one outstanding extension to SE1 is that the phase of initiation has to be completely passed in SE2, e.g. in SE1 a complete
requirements document is provided. In opposite to that, in SE2 capturing and managing the customer’s requirements is one of the most difficult parts of the course.

Another important aspect of SE2 is that the requirements in SE1 were fixed throughout the course whereas in a real project a change of requirements is rather more than less probable.

| Discipline focus | The discipline focus for SE1 ranges widely among engineering, management and quality issues. As the main part of the course is based on a common, i.e. team engineering project, the focus is considered broad. In addition to the teamwork, the single assignments allow to integrate a view on specialized topics. |
| Development focus | The students have to decide whether to extend a given foundation or to build completely new software for implementing the requirements. When building upon the provided foundation, even partly, analysis and evolution of this is included to the student’s tasks. In general this is not maintenance and evolutionary as in real environments and therefore must not be considered for this aspect. |
| Process model | The course SE1 is a derivative of RUP. There are no other process models included into the course. The accompanying lecture includes an overview of process models such as XP or the MSF. The theoretical part side focuses the contrast of agile and heavyweight methods. |
| Deliverables | The grading scheme for this course consists of 50% products at the end of the course and an equal 50% for the process (during the complete course duration). Only when delivering appropriate artifacts and following the given process model, a team qualifies for a proper grade. |
| Organization | Even if a noticeable amount of work is done individually, the main part of the course is teamwork. More exactly, the work decisive for the grade is teamwork. Individual work is done for qualifying for the course (phase A) and in single assignments throughout phase B. The reason for individual assignments in phase B is to add the aspect of in-depth researching of SE relevant, mostly technical, topics. |
| Technology focus | Even if the teams do develop a real application, some technologies were simplified. E.g. a lightweight DBMS is used for simpler usage and faster learning. That can be, in some way, considered more universal technology. |
| Tools | The course assignments do not require the use of a specific toolset for implementing the required artifacts. In fact the most frequently used industrial tools are too complex and intensive to learn for an application in an undergraduate course. The course tutors try to find a toolset, which is appropriate for the respective skill level of a team. |
| Assignment focus | The major part of the course is based on a team project. The single assignments, in contrast, are |

Table 1 Evaluating the course within the dimensions

| Discipline focus | As SE2 is placed in a real (mostly commercial) environment, more aspects were included within this course. E.g. requirements engineering. |
| Development focus | Unlike in SE1, the real environment of SE2 forces the need of integrating with existing systems and platforms. Most industrial projects are evolutionary or maintenance projects or have a strong relation to these aspects. The project must not be maintenance only to be accepted by the course administration – a development part must be featured as a main aspect. |
| Process model | The specified process model is, as in SE1, a simplified derivative of the RUP. The course organization is based on iterative and incremental characteristics. As SE2 builds upon a real customer project, the process model of the customer’s development environment may influence (or sometimes replace) the provided model, e.g. large enterprises develop own process models and –methodologies and the course integrates this on an individual basis. |
| Deliverables | Mostly same as in SE1. As there is less control of the process, e.g. the process is hosted at the customer’s site, the evaluation and grading may be difficult for that area. |
| Organization | Mostly same as in SE1. SE2 further adds more stakeholders, e.g. the customer may add team members to the development team. |
| Technology focus | As commercially used technology is specific, the technology focus is specific. E.g. in contrast to SE1, most customers are using enterprise-scale DBMS or application servers. |
| Tools | The toolset and tool environment is mostly determined by the customer’s development environment. Therefore, as most industrial development environments are basing on sophisticated and more complex tools, the SE2 students were advised to use these tools as well. |
| Assignment focus | The major part of the course is based on a team project. The single assignments, in contrast, are exercises. |
| Exercise domain | Students may choose any project the like to realize. Many students groups work for real companies in any fields described above. Because of complexity issues we advise students to select projects from the business domain. Anyhow they are not limited to that. |

Table 2 Evaluating the course within the dimensions
Further difficulties, when basing on a real commercial project, is to coordinate the customer’s process with the process and schedule of the course. To obtain a maximum learning success both courses, SE1 and SE2 have to be taken in succession. The didactic concept is to prepare and teach the methods in SE1 and to apply and enhance the skills in SE2. Taking SE2 without SE1 in advance completely contradict the concept and has proven worse when measuring (e.g. grading analysis) the learning success.

<table>
<thead>
<tr>
<th>Full topic coverage</th>
<th>SE1</th>
<th>SE2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independent exercises</td>
<td>SE1</td>
<td>SE2</td>
</tr>
<tr>
<td>Initial development</td>
<td>SE1</td>
<td>SE2</td>
</tr>
<tr>
<td>Single process model</td>
<td>SE1</td>
<td>SE2</td>
</tr>
<tr>
<td>Process delivery</td>
<td>SE1</td>
<td>SE2</td>
</tr>
<tr>
<td>Team organization</td>
<td>SE1</td>
<td>SE2</td>
</tr>
<tr>
<td>Universal Technology</td>
<td>SE1</td>
<td>SE2</td>
</tr>
<tr>
<td>Manual work</td>
<td>SE1</td>
<td>SE2</td>
</tr>
<tr>
<td>Business software</td>
<td>SE1</td>
<td>SE2</td>
</tr>
<tr>
<td>Partial topic coverage</td>
<td>Project</td>
<td>Maintenance and evo.</td>
</tr>
<tr>
<td>Many process models</td>
<td>Product delivery</td>
<td>Individual work</td>
</tr>
<tr>
<td>Specific Technology</td>
<td>Tool based work</td>
<td>Technical software</td>
</tr>
</tbody>
</table>

Table 3 Comparison Overview of SE1 vs. SE2

Table 3 shows a visual comparison of courses within the dimensions. This technique does not include or provide any quantitative or empirical information. A simple graph provides a comparative impression of the position of a course within a dimension. The same format is used for relating outcomes within dimensions the subsequent section.

5. FACING THE DIMENSIONS AGAINST THE SE2004 CC STUDENT OUTCOMES

When defining dimensions for SE course design, an evaluation of effects on student outcomes may be considered. Each dimension offers the option for setting up a course on either point of the dimension range. This section tries to evaluate the dimension (resp. certain points within a dimension) against the SE2004 CC Section 3.2 defined student outcomes. Some relations will be selected and discussed as examples.

{C, 3}: This outcome features existing systems. Professional SE as a non-classical engineering discipline needs to consider legacy systems and system integration constantly. When comparing a course set against the dimensions, at least advanced courses should be based on maintenance and evolutionary aspects.

{F, 2}: The SE2004 CC defines individual and teamwork as a required student outcome. When designing a course set and evaluating within the dimensions both, individual and teamwork should be parted within the superset of the evaluated courses. Individual work and teamwork must not necessarily be featured in one distinct course, but within the course set.

{G, 7}: One the one end of this dimension is abstract and academic technology; the other one is state-of-the-art professional technology. The defined outcome demands the professional and up-to-date. In some cases the usage of abstract and academic technology helps (inexperienced) students to build up skills in a low complex environment. Nevertheless a defined amount of a course set or curriculum must be based on contemporary industrial technology to meet the outcome.

Table 4 The dimensions defined in section 3

| 1 | Show mastery of the software engineering knowledge and skills, and professional issues necessary to begin practice as a software engineer. |
| 2 | Work as an individual and as part of a team to develop and deliver quality software artifacts. |
| 3 | Reconcile conflicting project objectives, finding acceptable compromises within limitations of cost, time, knowledge, existing systems, and organizations. |
| 4 | Design appropriate solutions in one or more application domains using software engineering approaches that integrate ethical, social, legal, and economic concerns. |
| 5 | Demonstrate an understanding of and apply current theories, models, and techniques that provide a basis for problem identification and analysis, software design, development, implementation, verification, and documentation. |
| 6 | Demonstrate an understanding and appreciation for the importance of negotiation, effective work habits, leadership, and good communication with stakeholders in a typical software development environment. |
| 7 | Learn new models, techniques, and technologies as they emerge and appreciate the necessity of such continuing professional development. |

Table 5 SE2004 CC [14] (Section 3.2) Student outcomes

{F, 6}: This outcome demands teamwork as communication and negotiation is teamwork-related only. At least a defined amount of work within a course set must be non-individual to meet the outcome.
6. CONCLUSION
The complexity inherent to the field renders SE course design an equal complex task. Although a variety of guidelines exists, single courses, either basic or specialized, require an abstract framework applicable to many types of courses. In this work, we have presented a set of current dimensions identified from teaching experiences in the last years. The set of dimensions described allows navigating through the specifications for SE education. Using the dimensions, SE course design may either be facilitated, as well as existing courses are evaluated regarding their compliance with the requirements. The one-dimensioned education. Using the dimensions, SE course design may either be

7. REFERENCES