Teaching the Unified Process to Undergraduate Students

Michael Halling  Wolfgang Zuser  Monika Köhle  Stefan Biffl
Institute for Software Technology and Interactive Systems
Vienna University of Technology
Karlsplatz 13, A-1040 Vienna, Austria
 halling@swt.tuwien.ac.at

Abstract

A modern software development process framework, such as the Unified Process, is able to overcome problems of past frameworks, e.g., structural mismatches (notation, used method) between the products in different process steps. Moreover, it promotes an industry standard, which yields instantly tangible benefits from better communication in development teams and further secures investments into development methods based on such a standard.

We describe the concept of a one-year undergraduate course in SE based on the Unified Process in detail and elaborate why, where and how the Unified Process can be incorporated into the course. The students’ response to the process was very positive. We argue that the Unified Process (UP) qualifies very well for being taught to SE students, as it integrates and extends essential SE practices. Another key argument for applying the UP was to integrate the different aspects and topics covered in the SE course and embedding them into a single structured and systematic SE process. By doing so students develop a much better understanding of the relationships and interactions between different models, analysis steps and techniques.

Keywords: Software Engineering Education, Unified Process, Case Study, Course Design, Teamwork.

1. Introduction

Software Engineering (SE) is still struggling to establish itself as a reliable engineering discipline. It is without doubt an engineering discipline [30][32], only changes of SE methods and techniques occur at a higher rate than best-practice approaches can be empirically sorted out and dispersed. Software meanwhile penetrates most parts of business and private life.

For SE education it is therefore important to focus training thoughtfully in order to teach SE students appropriate techniques and skills. Identifying those techniques and skills which are most important for future software engineers, is a challenging and highly subjective task. Tvedt et al. [33] pointed out that SE education has to find compromises between the requirements of today’s technology-driven world and fundamental SE theory. Especially in the current Internet era, new technologies are evolving with great pace putting strong demands on developers.

In general, SE courses must deal with a large variety of partly conflicting goals (e.g., prepare students for real life but focus on fundamental methods), and partly impossible goals (e.g., introduce problems of real life projects to SE courses). However, some work on the foundations and goals of SE education exists that provide a framework for successful courses [2][4][27][30]. Details of the courses may vary but the main points of all courses we believe to be: to teach state-of-the-art and state-of-the-practice techniques and to provide a project framework, which is as
realistic as possible given the constraints at a university, for practical application of these

The Unified Process (UP) [24] [25] qualifies for being taught to SE students as it integrates
and extends essential SE practices. The key advantage of the UP lies in its ability to satisfy
important needs and solve current SE problems of industrial practice (e.g., late design breakage
[24]), which were revealed in the last few years due to software development process evaluations
and an increased commitment to quality.

In this paper we present a one-year software engineering course that successfully teaches the
UP to undergraduate students. The remainder of the paper is structured as follows: Section 2
summarizes current research in the area of SE education and shows how our paper contributes to
this research. Section 3 briefly introduces the Unified Process and describes those parts that are
suitable for undergraduate education. Section 4 gives an overview on the SE course at the Vienna
University of Technology and provides information on the course timeline, the technical content,
and central organizational issues. Section 5 reports lessons learned from the new course design
from feedback of course participants, supervisors, and data gathered during the course. Section 6
finally summarizes the main points of the paper for application in education practice.

2. Current Research in Software Engineering Education

Research in SE education is probably as heterogeneous as the SE field itself. We will briefly
discuss the following main threads of research, as they are all somehow related to the content of
this paper:

- Definition of SE curricula for graduate and undergraduate programs;
- Description of specialized SE courses that include new techniques (e.g., the Personal
  Software Process or the Unified Process);
- Simulation of real-world environments for students at the university.

One important area of research deals with defining and developing SE programs on the
graduate [14][31] and undergraduate level [22][31][33]. The main challenge for these programs is
to define a sequence of courses that covers the skills and techniques required for software
engineers in an appropriate way. As our one-year course is aimed at second-year students, we are
only interested in the area of undergraduate SE education. Papers in this area point out that
undergraduate SE curricula are at an “early stage” [22]. This is the case at the Vienna University
of Technology, where our course was embedded in a general computer science program, which
was not at all focused on SE. Therefore our course had to cover a large variety of topics, starting
from object-oriented analysis to design, coding, and quality assurance. Of course, covering such a
wide range of topics restricted the scope of the course and disabled us from going into other, less
fundamental but equally important areas of SE like, for example, risk management. However, one
argument for applying the UP was to integrate these different aspects and topics covered in the
course and embedding them into a single structured and systematic SE process. By doing so
students should develop a much better understanding of the relationships and interactions
between different models, analysis steps and techniques.

Another important research direction deals with assessing and describing SE courses that apply
new techniques as for example the personal software process (PSP) [23]. While there exist
numerous papers on experiences with teaching the PSP (see for example [8]), we found only one
reference on teaching the UP. In [29] Robillard et al. describe a one-semester course that teaches
a customized version of the UP and present empirical data based upon a course with 30 graduate
students. We extend this analysis by presenting a one-year undergraduate (second-year students)
course that teaches a set of components of the UP. A key difference to [29] is that we do not
customize the UP for the education purpose but actually tailored our course to follow the UP.
Furthermore our qualitative results presented in this paper are based upon the feedback of over 300 students participating in our courses in the past two years.

Yet another thread of education research tries to identify means of making SE education more realistic. The goal, of course, is to prepare students as much as possible for situations common in real industrial projects. Some researchers evaluate courses where certain real-world problems and deficiencies (e.g., fluctuation of staff, changing requirements) are reintroduced into the course [12][13], while others evaluate how the cooperation of academic and industrial instructors can improve the quality of education (see for example [26]). McMillan [26] states that practitioners determined that user and client communication is the most important and crucial aspect of SE to be taught in courses. This again encourages teaching the UP, as one of its key competences is an improved and structured way of defining, managing and handling user requirements. In general, we believe that our course (for details see section 4) actually represents a very good balance of academic state-of-the-art, and industrial state-of-the-practice SE needs.

3. Teaching the Unified Process

The UP has become state-of-the-practice in the last years. It represents a well-founded and structured approach to software development that was specifically designed to remove some of the most obvious deficiencies of previous development processes. In this section we provide a brief survey on the area of object-oriented development, introduce the UP and present the components included in our course.

3.1. Object Oriented Software Development – an Evolving Field

The 1990s can be called the age of the object-oriented (OO) paradigm. In the first half of this decade many OO methods were created, published and discussed. There was no need for standardization or unification of methods [28], yet. The second half of the 1990s was mainly influenced by changing methods, by defining the first standard in notation for software models (UML [7]) and by shifting towards component based SE. The unification of existing OO methods [18][24], and the creation of new methods tailored to specific domains ([3],[23]) caused a lot of discussion [1][17][19][20][21] about usability for the (whole) software development life cycle and adaptability for future technologies and changes in SE standards.

Coad/Yourdon [9][10][11] is an example for a first-generation OO development method, which we used to teach from 1993 to 1999 [5][6][16]. After those dramatic changes towards the end of the millennium though, we did no longer regard it as a state-of-the-art software development method. We started looking for something that would handle key issues in modern software projects, e.g., requirements engineering, component engineering, dynamic models, architectural models and an implicit process model.

Through our consulting activities in industrial software development we gained practical experience with the UP. The two main reasons for the success and acceptance of the UP in industry were its tight coupling to UML and the available tool support through Rational.

3.2. The Unified Process

In Ivar Jacobsons’ company Objectory AB several versions of an OO process have been designed and distributed starting in the mid 1980s. At about the same time Rational was working on a software development process. Bringing together Jacobson, Booch and Rumbaugh at Rational 1995 started the unification of these two process frameworks continuing and incorporating the unification of the Booch method and Rumbaugh’s OMT as well as adding the concept of use cases from Jacobson. These efforts resulted in the publication of “The Unified Software Development Process” in 1999 [24].
Figure 1. Comparison of Waterfall Model and Unified Process [24].

Looking at the long history and evolution of the UP, which is mainly influenced by the research of well-known companies, we consider the UP as both state-of-the-art and state-of-the-practice third generation software development framework. The UP is a generic process framework, component-based and uses UML. It therefore reflects all the achievements of the ending 1990s regarding software development covering all our requirements for being taught in a SE course:

- **UP is use-case driven**: This use-case-driven development shows the importance of early user involvement as well as increases the traceability of requirements to implementation and tests.
- **UP is architecture-centric**: Development of modern distributed computer systems depends heavily on a good design and implementation of architecture. Providing many models for describing architectures, the UP supports an architecture-centric approach. The focus on architecture in early stages of software development stresses the major impact of design on project success.
- **UP is iterative and incremental**: Iterative and incremental development is an easy-to-understand process model, which is proven in industry and usually outperforms sequential process models. Figure 1 compares the progress of a project following the waterfall model (i.e., thin line in figure 1) to the progress of a project applying the Unified Process (i.e., bold line in figure 1). The project following the waterfall model suffers from a major drawback in project progress due to late detection of defects (e.g., inconsistencies, missing information) in the requirements document (i.e., late design breakage). The project following the UP avoids this problem by performing feedback and review cycles on a regular basis (i.e., an iterative and incremental development).

In fact, the UP only defines core workflows, which allows incorporating additional methods for project management, quality management, and change management very easily. Therefore the UP offered the possibility to reuse several concepts taught in our former SE course and simply integrate them into the UP. This was an important point, as the development of a SE course from scratch would have exceeded our resources by far.

In addition to its flexibility the main principles of the UP (use-case-driven, architecture-centric, iterative and incremental) offered the following advantages:

- They are state-of-the-art for the kind of projects we mainly focus on in SE course (distributed administrative medium sized applications with database connectivity).
They demonstrate important SE principles (which already have been taught in the earlier SE courses) in an easy to capture manner. Based on these advantages and features of the UP we decided to shape our course to comply with the UP, which we could easily justify and motivate without having to reformulate our main SE concepts. Details on the resulting course are presented in section 4.

3.3. Experiences from Past SE Courses

In the 15 years of teaching SE to undergraduate students the course has been continuously improved and adjusted to reflect new developments and achievements in the SE area. In this subsection we want to briefly present some of our experiences with the former courses already based upon OO software development, but not following the UP:

- Due to the evolution of the SE field in the past decade we ended up having a method mix (Coad/Yourdon, State Charts, introduction of use cases), which was a good choice for that time, but finally no longer up-to-date. Besides that, it was difficult to maintain.
- Trying to exchange single methods in order to introduce improved techniques was difficult because of the lack of interfaces between various methods.
- The lecture notes focused only on some selected methods (Extended Entity Relationship (EER), Structured Analysis and Design Technique (SADT), Coad/Yourdon OOA/OOD) mainly explaining notation. There was no integrated and complete SE process that explained the relationship and interaction between different models.
- Students reported that the provided documents were difficult to understand and to relate to each other. Students had an especially hard time understanding software requirements and mapping them to the design because the role of user requirements and their impact on following development products was not appropriately emphasized by the standard OO techniques at that time.
- There were an increasing number of teams who managed to find real customers for the project due in the second semester of our course (for details see section 4). These groups were usually forced to complete their projects within specific time schedules and constraints (completeness, formal correctness) of industry projects. Furthermore the communication with users and customers became especially important in these situations. Due the user centric use-case driven approach, we expected the UP to particularly support these teams.

Therefore we finally decided to re-engineer our SE course and adjusted it in such a way that it followed the UP. The following section presents a brief outline of the resulting course.

4. A Software Engineering Course following the Unified Process

In this section we describe our two-semester (i.e., four months each) undergraduate SE course based on the UP, which consists of lectures and workshop tasks. Due to the limited amount of space we focus on describing rather the course content than the operational management of the course. As organizing such a large-scale undergraduate course involves much effort, effective course design is an especially important criterion in order to guarantee high quality education with limited resources.

The course is integrated into the general computer science curriculum at the Vienna University of Technology and is compulsory for all students during their second year. Therefore approximately 250 students with different levels of previous knowledge and experience attend our course each year. Basically our course cannot expect more knowledge from students than basic procedural programming skills.
4.1. Lecture

The focus of this paper is clearly on describing the workshop design used for teaching the UP. However, lectures are important components of SE courses and must appropriately provide the information needed for completing the workshop tasks. Therefore we briefly present the fundamental structure of the lectures.

In the first semester students are introduced to the area of SE and are taught fundamental models of the UP in 30 lecturing units. A new textbook that historically evolved out of our own lecture notes accompanies the lectures [34] in the first semester. The lectures and the workshop are closely related and coordinated in order to ensure that students get the information required for solving workshop exercises at the right time in the lectures. In very broad terms the lectures have the following structure and content: introduction into SE; introduction into OO SE and UP; requirements engineering; analysis; design; design of relational database systems with EER; OO system construction and system integration; testing; work organization and communication techniques; reviews and software inspection.

In the second semester we ask people from industry and research to present advanced topics in SE, which are rather independent from the workshop, as students are to apply the fundamental techniques acquired in the first semester in their own, industry-related project. These lectures intend to provide an outlook of potential challenges in industry and current research activities. Topics covered lately included risk management, legal aspects, usability issues, security considerations and UP implementations in small and medium-sized companies.

4.2. Workshop

The workshop of our SE course consists of three phases – A, B, and C. In phase A (duration of about 4 weeks) students are asked to solve a small software development example including a simple requirements analysis and OO code. In order to guarantee that every participating student has a minimum of OO software development skills, only positively rated students are allowed to further proceed with the course.

At the beginning of phase B students get randomly assigned to project teams so that there is a balance of skilled and less skilled team members (with respect to their performance in phase A). Random assignment introduces some aspects of real-life projects, as communication, project management and cooperative work management become very important concepts that must be actively applied during the workshop.

In each project team students are asked to assume certain roles, which are described in table 1. We had to merge roles from the UP due to the maximal team size of 6 students. On the other hand we added some roles, which are not covered by the UP but mandatory for simulating a complete software project according to industrial reality.

In phase B students have to finish a prepared, medium-size project, where a requirements document, a design document and a source code framework already exist. The focus of this phase is on understanding requirements, system design and system testing as well as applying review techniques in order to quality assure the received material.

Phase B represents an essential step towards a non-toy project. Especially when considering the fact that most SE techniques are totally new to the participants. Providing students with example documents of appropriate and realistic size, which exactly follow the guidelines described and taught in the lectures, is very instructive. Furthermore students are asked to complete certain tasks of the development process using the information provided by existing project documentation and source code, e.g., develop a part of the user interface, implement certain classes and integrate them with the existing infrastructure. The main goal of phase B is to prepare students for the second semester of the workshop, i.e., phase C, where their own project
of similar size must be solved entirely by the team (usually for a real client from industry) following the UP and applying techniques and methods acquired in the first semester.

Table 1. Dependency between course roles and original Unified Process roles

<table>
<thead>
<tr>
<th>SE course role</th>
<th>Original UP role</th>
<th>Role Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyst</td>
<td>System Analyst, Use-Case Specifier</td>
<td>The analyst communicates with users of the system in order to find and describe use cases.</td>
</tr>
<tr>
<td>Integrator</td>
<td>Architect, System Integrator</td>
<td>The integrator defines the architecture depending on technical constraints like system type (Standalone, Client/Server, Internet-Application), interfaces to legacy systems, non-functional requirements (e.g., throughput, response time) and deployment environment.</td>
</tr>
<tr>
<td>Programmer</td>
<td>User-Interface Designer, Use-Case Engineer, Component Engineer</td>
<td>The programmer builds prototypes for analysis and design, writes source code and provides binary components for integration.</td>
</tr>
<tr>
<td>Tester</td>
<td>Test Engineer, Integration Tester, System Tester</td>
<td>The tester sets up test plans, a testing environment and performs all kinds of tests.</td>
</tr>
<tr>
<td>Project leader</td>
<td>Not defined in UP</td>
<td>The project leader is responsible for project planning and representing the project team to instructors or external clients.</td>
</tr>
<tr>
<td>Group leader</td>
<td>Not defined in UP</td>
<td>A group leader is needed if more than one student is assigned to a role, e.g., developer. In this case the group leaders assigns tasks and checks if the tasks have been finished according to the project plan.</td>
</tr>
</tbody>
</table>

Table 2 summarizes main process steps followed during phase C, lists main documents to be produced and relates them to the process steps defined in the UP (for a more detailed list of products being developed during the workshop refer to Table A1 in the Appendix). The artifacts show a very strong connection to appropriate artifacts defined in the UP except for three situations:

- We put more focus on the distinction between roles of classes and objects during design and implementation steps. Students in our course with fairly low experience in designing object-oriented systems tend to create universal classes, which implement everything from user interface to database connection. We think it is necessary to promote more sophisticated approaches: design models like MVC [15] or design principles like low coupling and high cohesion. It is necessary to enforce different views on distinct roles of classes. Therefore all classes covering a certain role (e.g., database connectivity or business logic) have to be described as separate artifacts.

- The Unified Process covers deployment only superficially and maintenance not at all. As we want to cover the whole software lifecycle in our course, we added appropriate artifacts for system deployment and integration into production environments. For maintenance artifacts only bug reports have been added (for showing that there will be bugs and changes and therefore maintenance activities), as this step is not part of the workshop (see table A1 in the Appendix).

- Artifacts created through required project management and quality assurance activities (project plan, review reports) have been added because of the same reasons as in the previous point.
Table 2. Mapping of products and phases in phase C of the workshop to Unified Process steps.

<table>
<thead>
<tr>
<th>Course Phase</th>
<th>Documents Required</th>
<th>Duration (Team effort)</th>
<th>Unified Process Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Requirements Analysis</td>
<td>Vision Statement, Requirement Gathering</td>
<td>2 weeks (40 h)</td>
<td>Inception</td>
</tr>
<tr>
<td>System Design</td>
<td>Requirement Description, Architectural Design, Design, Test Preparation</td>
<td>5-6 weeks (220 h)</td>
<td>Elaboration</td>
</tr>
<tr>
<td>Implementation, Integration, Test Plan</td>
<td>Implementation, Integration, Testing</td>
<td>4-7 weeks (150 h)</td>
<td>Construction</td>
</tr>
<tr>
<td>Acceptance Test, Presentation</td>
<td>Final System Tests, User Documentation, Final Presentation</td>
<td>1-3 weeks (60 h)</td>
<td>Transition</td>
</tr>
</tbody>
</table>

One big limitation of our workshop design is that is not possible to utilize the incremental and iterative approach at its full extent. Because of the very limited time span during the semester (see table 2) students do two iterations during inception and elaboration phase and three to four iterations only in construction phase.

The most important aspect of phase C is to make students exercise the process steps and techniques learned in a real project with a real client, who is interested in receiving a high quality product on schedule. Although it is not too easy to find industry partners for such projects, students succeed in more than 75% of cases and actually appreciate the additional demands of this real life experience. If teams do not find any “real” project, we provide project partners for choice.

Another important point to mention is that the course also includes teaching and practicing standard quality assurance techniques, namely reviews and testing. After completion of a predefined development stage working products are reviewed by tutors and discussed in the team. Immediately after finishing the requirements analysis phase teams start to create test plans and test cases, which are then used to test the final product at the end of the semester.

5. Lessons Learned

So far the course outlined in the previous section was conducted twice. We collected qualitative feedback and derived the following lessons learned related to teaching the UP:

- Changing the previous SE course (eliminating obsolete concepts and strengthening the theory behind the UP) was an intensive task. The preparation of workshop documentation took over one year for three persons (10-15 hours a week) for the change to UML and the UP. Similarly restructuring the existing lecture notes and extending them caused another 500 hours of effort. Therefore drastic changes must be well planned and well thought-out. Similarly to project and quality planning it is important to take economic cost-benefit aspects into consideration. In this case we are convinced that the benefits justified the investment.

- From students already working in industrial projects, we expected to get positive feedback about using a modern state-of-the-art process, which currently is broadly used in industry. This was only true for modeling with UML but not for using UP as software development process. Only a few industrially experienced students have had experiences with formal software development processes in general and almost none with UP in particular. Most of the students therefore had no explicit expectations about the used software development methods and processes. ??An appropriate trade-off between being
up-to-date teaching the newest advances in software engineering and available resources for preparing and conducting the course seems to be justifiable.

- Evaluating the students’ feedback showed that the deficiencies of the former course (see section 3.3) were effectively removed due to applying the UP. The most important improvement was that students were able to relate different models (e.g., use case diagrams and domain models) to each other and understand the correlation. This was not the case in the previous courses where different models existed but their relationships to each other were not obvious and well defined.

- Further student feedback revealed that using a homogenous process framework allows students to concentrate on gaining important experiences in team dynamics, project management, and quality management during the course.

- Our instructing experience shows that the Unified Process is comparatively easy to teach due to its modularity. Each process step and its associated models form an integrated whole from a teaching as well as a learning perspective.

- The workshop design is effective in a way that allows us to administer the workshop with limited resources (i.e., 1-2 members of permanent academic staff; 15 tutors for student team supervision and support) but to the satisfaction of participating students. Providing templates for documents and using existing documents and project material as examples enables students to link theory to practice.

- We intended to use as many artifacts as possible in the workshop to cover a wide range of UP’s capabilities and advantages, like traceability of requirements. This overview of artifacts serves well as introduction into the UP but does not allow to practice the creation of single artifacts intensively. The limited possibility to perform sufficient iterations (see section 4.2) contributes to this problem.

6. Summary

In this paper we describe a two-semester undergraduate SE course that teaches software development following the Unified Process to more than 200 students. We present related work that shows that (a) the UP qualifies for SE education as it is a well defined, theoretically sound and practically important development process and (b) that not many courses teaching the UP exist so far.

We motivate why the UP represents a key concept of modern SE. Furthermore we point out problems when teaching OO development techniques without the UP. Finally we outline a SE course that follows the UP and discuss most important lessons learned in the past two years doing this course. Besides following the UP our workshop is interesting because it shows how a realistic project environment can be established even for undergraduate SE students (e.g., by random assignment of students to teams, pre-developed medium size projects, studying of existing documentation and source code, real client projects).

Our main result is that the UP fits very well to SE education in our framework (i.e., a one-year course including lectures and workshops). We could observe wide acceptance during our courses. The UP clarified many aspects for the students by providing an integrated and combining process.

Assuming that usually curricula for undergraduate students are not specialized in software engineering education, the amount of necessary information taught in an undergraduate software engineering course following the UP definitely requires a one-year time horizon. Please, note that this situation might be different for graduate students [29] and curricula with an emphasis on SE. However, the UP also qualifies for “shorter” (e.g., one-semester) software engineering courses because it can be easily tailored to the specific requirements of each course (e.g., limited amount
of time; different previous knowledge of students) due to its high level of modularity and flexibility.

References


[34] Zuser W., Biffl S., Grechenig T., Köhle M., Software Engineering mit UML und dem Unified Process; Munich, Pearson Studium, will be published in Oktober 2001

Appendix

Table A1. Dependency between Course Artefacts and Original Unified Process Artefacts.

<table>
<thead>
<tr>
<th>SE Course Artifact</th>
<th>Original UP Artifact(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project plan (*)</td>
<td>Phase Plan (RUP), Work Order (RUP)</td>
</tr>
<tr>
<td>Domain model (*)</td>
<td>Domain model</td>
</tr>
<tr>
<td>System overview and glossary (*)</td>
<td>Glossary</td>
</tr>
<tr>
<td>Directory of actors</td>
<td>Actor</td>
</tr>
<tr>
<td>Use-Case model (*)</td>
<td>Use-Case model</td>
</tr>
<tr>
<td>Use Case description (*)</td>
<td>Use Case</td>
</tr>
<tr>
<td>User-Interface prototype (*)</td>
<td>User-Interface prototype</td>
</tr>
<tr>
<td>Analysis model</td>
<td>Analysis model</td>
</tr>
<tr>
<td>User interfaces</td>
<td>Analysis class: Boundary class</td>
</tr>
<tr>
<td>System interfaces</td>
<td>Analysis class: Boundary class</td>
</tr>
<tr>
<td>Controller description</td>
<td>Analysis class: Control class</td>
</tr>
<tr>
<td>Entity description (*)</td>
<td>Analysis class: Entity class</td>
</tr>
<tr>
<td>Architecture description (*)</td>
<td>Architecture description (View of the design model)</td>
</tr>
<tr>
<td>Description of subsystems and components (*)</td>
<td>Design model, Design subsystem, Interface</td>
</tr>
<tr>
<td>Class diagram for business logic (*)</td>
<td>Use Case realization: Class diagrams</td>
</tr>
<tr>
<td>Design of classes of business logic</td>
<td>Design class</td>
</tr>
<tr>
<td>Error handling</td>
<td>No equivalent artifact in UP</td>
</tr>
<tr>
<td>Design of user interface (+*)</td>
<td>Design class</td>
</tr>
<tr>
<td>Class diagram for data-access-layer</td>
<td>Use Case realization: Class diagrams</td>
</tr>
<tr>
<td>Design of database (+*)</td>
<td>Design class</td>
</tr>
<tr>
<td>Integration build plan</td>
<td>Integration build plan</td>
</tr>
<tr>
<td>Architecture</td>
<td>Architecture description (View of the Implementation model)</td>
</tr>
<tr>
<td>SE Course Artifact</td>
<td>Original UP Artifact(s)</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>Subsystems (+*)</td>
<td>Implementation subsystem</td>
</tr>
<tr>
<td>Classes</td>
<td>No equivalent artifact in UP</td>
</tr>
<tr>
<td>System</td>
<td>Implementation model</td>
</tr>
<tr>
<td>Test plan (+*)</td>
<td>Test plan, Test procedure</td>
</tr>
<tr>
<td>Test case (+*)</td>
<td>Test case</td>
</tr>
<tr>
<td>Testing reports (+*)</td>
<td>Test evaluation</td>
</tr>
<tr>
<td>Error reports</td>
<td>Defect</td>
</tr>
<tr>
<td>Description of production environment (*)</td>
<td>Deployment plan (RUP)</td>
</tr>
<tr>
<td>Description of data-migration</td>
<td>Deployment plan (RUP)</td>
</tr>
<tr>
<td>Report of installation and migration procedure</td>
<td>Not part of UP nor RUP</td>
</tr>
<tr>
<td>Running system (+*)</td>
<td>Installation artifacts (RUP)</td>
</tr>
<tr>
<td>Change request</td>
<td>Change request (RUP)</td>
</tr>
<tr>
<td>Integration-Testing-Reports (+*)</td>
<td>Not part of UP nor RUP</td>
</tr>
<tr>
<td>Bug report</td>
<td>Not part of UP nor RUP</td>
</tr>
<tr>
<td>Review report (+*)</td>
<td>Iteration Assessment (RUP), Review Record</td>
</tr>
<tr>
<td></td>
<td>(RUP)</td>
</tr>
</tbody>
</table>

+…Is mandatory for SE workshop Phase B; *…Is mandatory for SE workshop in Phase C