# The allure and risks of a deployable software engineering project

**Author(s):**
Piccioni, Marco; Meyer, Bertrand

**Publication Date:**
2012

**Permanent Link:**
https://doi.org/10.3929/ethz-a-006820356 →

# The Allure and Risks of a Deployable Software Engineering Project

Bertrand Meyer
*Department of Computer Science*
*Clausiusstrasse 59*
*8092 Zurich, Switzerland*
*+41 44 632 0410*
*Bertrand.Meyer@inf.ethz.ch*

Marco Piccioni
*Department of Computer Science*
*Clausiusstrasse 59*
*8092 Zurich, Switzerland*
*+41 44 632 6532*
*Marco.Piccioni@inf.ethz.ch*

***Abstract***

*The student project is a key component of a software engineering course. What exact goals should the project have, and how should the instructors focus it? While in most cases projects are artificially designed for the course, we use a deployable, realistic project. This paper presents the rationale for such an approach and assesses our experience with it, drawing on this experience to present guidelines for choosing the theme and scope of the project, selecting project tasks, switching student groups, specifying deliverables and grading scheme.*

## 1. Introduction

An undergraduate computer science or IT curriculum should and often does include a "software engineering" course, typically 3rd- or 4th-year. One component seems to be accepted as essential: a course project where students have the opportunity to try out some of the principal concepts and techniques taught in the lectures. This software engineering course project — its purpose, its constraints, the issues it raises, how best to set it up — is the focus of the present paper. In particular we suggest that in a software engineering course, as distinguished from programming, design or analysis courses, it is desirable to use a realistic and deployable software project. We describe in detail our experience of applying this principle at ETH.

Section 2 presents the course's scope and emphasis, and the project's goals, scope and setup. Section 3 describes the specifics of our course and section 4 the details of the project. Section 5 assesses the project's outcome from both the students' and instructors' perspectives. Section 6 draws conclusions.

## 2. The software engineering course and its project

### 2.1. Course scope and emphasis

To discuss the project it is necessary first to define what the Software Engineering course as a whole is attempting to achieve. There is no absolute consensus on this point. Professional groups have come up with very useful recommendations through the SWEBOK project (Software Engineering Body of Knowledge) [14], and also the curriculum guidelines drawn up by ACM SIGSOFT [9]. In practice, however, curricula still vary widely, as we can see almost daily from examining the academic records of students from diverse universities who apply for a software engineering master's at ETH. In a surprisingly large number of institutions, the course mainly teaches UML, which clearly is not enough. In others, it is devoted to design patterns, a very important topic but not in our opinion suited as the focus of a software engineering course. A more appropriate approach is to follow standard textbooks in the area [2, 6, 7, 10], which cover a wide range of topics such as the software lifecycle, requirements, design, implementation, testing, metrics, project management. The danger with such breadth, however, is to turn the course into a catalog description that sacrifices depth for coverage.

Here is our own view. We define software engineering as covering all activities involved in producing and operating software, with five major dimensions captured by the acronym DIAMO: Describe (specification); Implement (design and programming); Assess (testing and other a posteriori quality assurance techniques, metrics); Manage (project management, software process); and Operate (software deployment and operation). The second dimension, covering design and programming, should in our view be de-emphasized in the lectures of the software engineering course; in a normal computer science curriculum students have had other courses focused on programming and design and will have more. The lectures of the Software Engineering course are for most students the only opportunity they will have, in the curriculum, to hear about non-programming aspects of software, with a special emphasis on such topics of great importance to industrial practice as requirements, quality assurance, process organization (conventional and agile).

## 2.2. Project goals

In this view of the software engineering course the project occupies a central place, reflected by its weight in the final grade (currently 50%) and its share of the exercise sessions. The project is not just an expanded exercise but also a central component of the course, along with the lectures. Like any educational technique, the project should have clearly defined pedagogical goals:

- As can be expected of a project in any course, it provides a test bed for applying the concepts introduced in the course.
- Another objective is to give the students a clear understanding of the challenges of actual software development in industry. To qualify this goal we note both that many students (78% in our case) have some industry programming experience, either because they are older students coming back to school or simply because they took internships, and that a university is not a company and should not try to be: we provide a controlled environment mimicking some but not all of the conditions of industry. As an obvious example of condition not mimicked, students are not paid for their work and cannot be fired (although failing the course may eventually lead to an equivalent result). An example of constraint that is actually stronger in academia than in most industrial environments is the inexorable time limit, in our case a 14-week semester: shifting the deadline and delivering late is simply not a possibility.
- Beyond an understanding of the challenges, we of course teach state-of-the-art answers to these challenges; the project prompts the students to learn and apply these techniques.
- Specific challenges and solutions that are important in industry and hard to teach in university curricula involve requirements engineering and testing, especially test plan preparation. The project emphasizes these goals, as discussed below.
- The project also emphasizes group work. This goal is useful in environments where previous software projects have been individual. It is not critical in our case since our students have already done several group projects.

Our course adds another goal: to produce a system that fills some of our own needs. This builds on and goes beyond the idea of developing an application for students in other departments [1]. As explained below, this goal, while self-serving, also follows from pedagogical considerations.

### 2.3. Project scope

Designing the project is a delicate matter. The first question is whether to include implementation. One may be lured by either of two opposite answers:

- The project might not include an implementation, limiting the students' work to requirements and design, perhaps a test plan. This would be in line with the course's emphasis on non-programming aspects. It also makes it easier to fit the project in the stranglehold of a 14-week semester, but it does not serve the purpose of the course. It is just too easy for a project that does not ask for an implementation to result in hand waving. As an example, the main criterion for judging requirements is whether they can be implemented at reasonable cost. There is essentially no way to judge this except by requiring students to implement them. A similar observation applies to test plans.
- At the other extreme, it is easy to give just a programming project. But if it's only about programming, even if this is taken to include design, it misses the point. Students need such programming projects, but in other courses. In software engineering they should also work on the non-programming aspects listed above. So even if the lectures themselves spend relatively little time on implementation the course should, in our opinion, include implementation.

### 2.4. Project tasks

The following tasks each corresponds to a milestone in the project: requirements; test plan; design and implementation; deployment and test.

Requirements are a key step in industrial software development. It's one of the skills that are often not acquired in university education. It is difficult to teach, and its absence in new graduates is most bemoaned by industry. The software engineering course and its project are the right place to teach it. Successful requirements engineering requires a good computer science basis (otherwise, one just does not know what among user requests is trivial, easy, expensive, hard or infeasible), and so cannot be taught at the introductory level; once students have that basis, they need to be introduced to the challenges of requirements: identifying stakeholders, getting them to state their requests, understanding and qualifying these requests, separating them into categories — essential, desirable, moved to second release, discarded —, turn them into a cogent requirements document, getting buy-in from managers, users and developers. Along with technical knowledge this requires communication skills and an engineer's sense of the possible.

Writing a test plan is for most students a novel exercise; they are used to testing their own programs, but not to devising acceptance tests for a program that does not yet exist, on the sole basis of requirements. This experience is again of high value to industry. The exercise is also useful to highlight the importance and difficulties of the previous task, requirements: a requirements document is little more than wishful thinking unless it is precise enough to enable a QA team to write a test plan independently of the development team.

Design and implementation are, as noted, a required part of a credible project. As the main emphasis of the course lies elsewhere it is acceptable to treat these two tasks as a single step. The students taking our course at ETH have already had challenging projects involving design and implementation.

Running the test is an easier task than the previous three if the test plan is good enough, but an indispensable part of the task package, as it makes it possible to check both the test and the implementation.

In the scheme we propose, the project description provided at the beginning of the semester [13] clearly lists these four tasks, with due dates of 5, 7, 12 and 14 weeks into the semester, and grading weights as in Table 1 (see 4.6).

### 2.5. Switching: when and how

It is often interesting in a multi-stage software engineering project to switch tasks between groups at specified stages, for example to have one group test another group's implementation. Such an approach exposes students to the needs of making their work usable and assessable by others. It has its roots in Horning's early "Software Hut" technique [4], where student groups had to bid on components developed by other groups. Instead of putting money into the picture, we foster cross-group interaction by switching tasks at two specified stages. A common strategy is to ask each group to hand over its requirements to another group for design and implementation. It is not pedagogically appropriate, however, because it removes one of the principal issues of requirements: striking a proper balance between the desirable and the possible. If you do not have to live with the results of your requirements imagination, why restrain yourself? The result is that some groups will produce pie-in-the-sky requirements, which others are then unfairly required to implement, taking the blame if they miss some of the functionality, however outlandish. Instead, one should make sure that each group implements its own requirements. In grading the Software Requirements Specification (25% of the project grade), we assign 30% to "Extent and usefulness of functionality"; in grading the implementation (40% of the project grade) we assign 30% to "SRS coverage". So the students must make a tradeoff between writing requirements that are ambitious enough, to maximize the first criterion, and not too ambitious, to be able to implement them and avoid losing points on the second criterion. The place to switch groups is, in our view, between requirements and test plan. Each group is asked to review another group's requirements and devise from it a test plan, which it will be asked to run in the last step of the project. Among other benefits this makes it possible to highlight important software engineering principles about testing, quality assurance in general and, as noted, the necessity and difficulty of devising tests uninfluenced by an existing implementation. The assignment of one group's requirements to another for test plan preparation should be anonymous, to preserve the soundness of the process. We make it clear that revealing identifying information would be considered cheating and lead to failing the course immediately. Note, however, that inadvertently identifying a particular group, while undesirable, would not destroy the overall setup.

## 3. Specific context

The software engineering course at ETH (a surprisingly recent addition to the curriculum) is taught in the third year. It is one of 7 "core courses" of which all CS students must take at least 4; in practice most select it. A typical class size is 100-120 students. The course is worth 6 credits; per week over a semester (14 weeks) it has three hours of full-class lectures and two of exercise sessions in small groups. In the course iteration reported here the exercise sessions were partly devoted to the project, which was introduced at the very beginning of the semester. The grading was 50% for the project and 50% for the exam, held in the following semester break. There were no graded homework or midterms, allowing students to devote full attention to the project. For the exercise sessions and in general for course organization, we had at our disposal a group of 7 assistants. One of the assistants did not have an exercise group but served as "back-office assistant" in charge of the course Web site and overall organization.

## 4. Project details

### 4.1. Project start-up

The students were asked to aggregate in teams of 3 people in each of the 6 exercise groups. As in an industry project we imposed the technologies: MySQL as the database, Eiffel as the design and programming language, the EiffelWeb library as the web framework. The CSEL project [12] source code was provided for guidance. The different projects were to be deployed on an Apache server under Linux. As Eiffel technology and the EiffelStudio IDE are available on Windows as well as on Linux, the students were free to choose their own environment and set up a local server for testing.

### 4.2. Assignment 1: requirements elicitation

The initial description of the expected CSÁRDÁS system was intentionally loose, to stimulate questions from students. A Wiki page was set up to provide interaction in question-and-answer format between the groups and the stakeholders, a role played by the course assistants. Some questions realistically elicited different and sometimes contradictory answers from the stakeholders. This had been foreseen and led to setting up a second page for the project leader, in this case the professor, to provide authoritative answers. The deliverables included a requirements document following the IEEE standard structure (IEEE-STD-830/1998) for a Software Requirements Specification (SRS), an Object Oriented Requirements Specification (OORS), meant as a graphical O-O requirements description in Eiffel or UML, and anything else that they find useful, e.g. screenshots or use cases. Since in the second assignment these products were handed over to another group, the students had to ensure that they were usable in the absence of any contact with their original developers.

### 4.3. Assignment 2: test plan & specification

For the second assignment, groups were swapped randomly. Every group was asked to develop the tests from the requirement produced by another group, without knowing the identity of its members. The deliverables of the second assignment included a Test Plan following the IEEE standard structure (IEEE-STD-829/1998) and a Test Specification with a structure defined by the same standard. To allow everybody to accomplish the task in the allotted time, we asked each group to focus on a set of 10 core functional requirements from the target SRS.

### 4.4. Assignment 3: design, implementation and documentation

The purpose of the third assignment was for each group to *design*, *implement*, *document* and *deploy* the CSÁRDÁS system for which it had been devising the requirements in the first assignment. The deliverables included the source code, the Software Design Description (SDD) and release notes. We also provided students with instructions to deploy the compiled binaries to the target server via FTP and to manage the databases assigned for testing.

### 4.5. Assignment 4: test execution & reporting

The purpose of the fourth assignment was to test the CSÁRDÁS candidate system. Each group was given the URL and release notes for the system under test, developed by the same group for which they devised a test plan in the assignment 2. It was allowed to suggest changes to the previous test plans. In line with the IEEE standard structure (IEEE-STD-829/1998), the deliverables included a test log, a test incident report and a test summary report. To limit the test effort we asked each group to focus on a subset of the features developed, devising and executing at least one test case to demonstrate the usual expected behavior and three test cases to test exceptional behavior if applicable.

### 4.6. Grading scheme

The project grade was half of the final course grade. Table 1 shows the detailed grading scheme used for the project.

**Table 1. Detailed project grading scheme**

| Assignments | assignment grade % | project grade % |
|---|---|---|
| **Requirement document and OORS** | ***100*** | ***25*** |
| *Readability* | *20* | *5* |
| *Testability* | *30* | *7.5* |
| *Precision and detail of description* | *20* | *5* |
| *Extent and usefulness of functionality* | *30* | *7.5* |
| **Test plan and test specification** | ***100*** | ***25*** |
| *Readability* | *30* | *7.5* |
| *Precision and detail of description* | *30* | *7.5* |
| *Test coverage* | *40* | *10* |
| **Design, implementation, documents** | ***100*** | ***40*** |
| *Implementation: coverage of the SRS* | *30* | *12* |
| *Implementation: quality* | *30* | *12* |
| *Design: quality documented in SDD* | *30* | *12* |
| *Precision and detail of documentation* | *10* | *4* |
| **Test execution and reporting** | ***100*** | ***10*** |
| *Chosen test cases* | *30* | *3* |
| *Precision and detail of documentation* | *70* | *7* |

## 5. Outcome and evaluation

### 5.1. Project outcome

The project outcomes are remarkably good, especially if considering the installation and setup issues faced along the way. The overall project grades were uniformly distributed in a range between 60% and 96%. Every assistant selected the best solution from his own group. The best indicator of success is that CSÁRDÁS is now deployed on the Informatics Europe server.

### 5.2. Project evaluation: the student view

To get specific feedback on the project the students were asked to fill in a questionnaire based on the Questionnaire on Current Motivation (QCM), which in turn relies on a well-known model of motivational psychology [8]. We have experience with QCM-based evaluation from a previous, unrelated course [5]. We took as a sample all the 45 students that answered the questionnaire; this constitutes half of the course population. To provide a reasonable qualitative insight into the result, we have selected the mode as a statistical measure of central tendency. It represents the highest point in each distribution of answer scores for each question, and in our case provides a better description of the results than the mean, which is influenced by extreme scores. Table 2 only shows the questions for which answers reached a significantly high mode.

**Table 2. Questions with answers that reached a significantly high mode**

| | |
|---|---|
| *1. I consider myself as a proficient or excellent programmer* | *78 %* |
| *2. I have already worked in a job which involved programming* | *78 %* |
| *3. I considered the project uninteresting after reading the description* | *59 %* |
| *4. I experienced pressure having to perform well solving the project* | *69 %* |
| *5. I believed to have been able to tackle the difficulties of the tasks involved in the project* | *60 %* |
| *6. I have worked very hard for the project* | *91 %* |
| *7. I am very curious about how well I have done in the project* | *63 %* |
| *8. I think that devising an implementation for the project application was tough* | *64 %* |
| *9. I would have preferred a tighter integration between the project and the exercise sessions* | *62 %* |
| *10. I have managed to test my application locally before deploying it on the target server* | *78 %* |

About what caused trouble during development, the three top answers were server downtime (80%), EiffelWeb support (76%) and deployment to target server (58%).

## 5.3. Project outcome: the instructors' view

The answers shown to the selected questions are representative of the general feelings of the students about the project. Items 1, 2 suggest that the majority of students in the sample are confident in their skills and have part-time jobs related to their studies. Items 4, 6 and 8 indicate that the project was difficult, that it generated pressure and that students worked very hard at it, a message emphatically confirmed by direct feedback. Yet items 5 and 7 suggest that the students have mostly been able to cope with the issues faced and were curious about the result. A possible interpretation is that in spite of the initial low expectations, (item 3) the project generated interest along the way, probably when it raised real challenges to students. Item 9 suggests that we should prepare more project-focused exercise sessions, to provide more guidance and mentoring. Item 10 confirms that although most of the students deployed and tested their applications locally in advance, the few that didn't caused problems to all the others by repeatedly killing the server. Also note that while the project work was organized in groups, the questionnaire collected single answers, so the good overall quality of group projects does not express anything about possible frustrations of individuals in these groups. This matches industry situations, where the interest of the company (project delivered in time) typically prevails over the interest of any specific individual. Overall, the questionnaire therefore has helped us in getting some feedback and insights on how to improve the project's management. It will be interesting to compare future results.

## 5.4. Consequences of using a real project

It was noted that a consequence of choosing a real project, and even more challengingly a web project, is to raise the deployment issue. After testing their system on their own machines, students were required to install it on a shared server accessible only within ETH. This turned out to be one of the most delicate parts of the project.

Apart from some server setup issues, it happened that some groups (as we found out, only a small number, but this did not make things any better) had not properly tested their programs, which either took up all memory or entered infinite loops, preventing others from testing on the server. Although some of the problems will go away with better planning and the benefit of this first experience, it is clear that any deployment on shared resources carries such risks. The choice of an Internet-oriented application carries other risks that we had not envisioned. For example when testing their applications students used emails with addresses such as (naturally enough) some_name@csardas.org, with inevitable bouncing. The assistants noticed this pattern early enough to stop it before the site administrators of the unsuspecting (and charming) Hungarian dance site had time to complain or — we hope — notice. This illustrates a general lesson: using a real-life project topic forces the instructor to consider, and if possible anticipate, risks that would not arise otherwise, including the risks of devising all-too-clever acronyms.

## 6. Lessons drawn

The technical issues that arose during the deployment and testing of the project can and should be solved. They had a negative impact on the general student satisfaction and led to underestimate the weight effectively assigned to the development phase. We are therefore repeating the experience for this year (2008) course. We also hope that other universities will try a similar set up for their software engineering courses.

## 7. Acknowledgments

Our thanks to all the ETH students of the Software Engineering course, Spring Session 2007, who put all their efforts in achieving a remarkable result, and to Michela Pedroni for help preparing the project questionnaire.

## 8. References

[1] Ali, Muhammad Raza. Imparting effective software engineering education, ACM SIGSOFT Software Engineering Notes, v.31 n.4, July 2006.

[2] Ghezzi, C., Jazayeri, M., and Mandrioli, D. *Fundamentals of Software Engineering*. Prentice Hall, 2002.

[3] Jaccheri, M. L. and Lago, P. *How Project-based Courses face the Challenge of educating Software Engineers.* Proceedings of the joint World Multiconference on Systemics, Cybernetics and Informatics (SCI '98) and the 4th International Conference on Information Systems Analysis and Synthesis (ISAS '98), Orlando, USA, 1998.

[4] Horning, J. J. *The Software Project as a Serious Game.* In Wasserman, A. I. and Freeman, P. editors. Software Engineering Education: Needs and Objectives, 71-77. Springer Verlag, 1976.

[5] Pedroni M., Bay T. G., Oriol M. and Pedroni A., Open Source Projects in Programming Courses, in SIGCSE 2007, ACM, Covington, Kentucky, USA.

[6] Pfleeger, S. L., and Atlee J. M., *Software Engineering.* Third edition. Prentice Hall, 2005.

[7] Pressman, R. S. *Software Engineering. A practitioner approach.* Sixth edition. McGraw-Hill 2005.

[8] Rheinberg, F., Vollmeyer, R. and Burns, B. D. QCM: A questionnaire to assess current motivation in learning situations. *Diagnostica*, 47: 57-66, 2001.

[9] SIGSOFT *Software Engineering 2004.* Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. Downloadable at http://sites.computer.org/ccse/

[10] Sommerville, I. *Software Engineering.* Addison-Wesley, 2004.

[11] http://www.informatics-europe.org/

[12] http://www.informatics-europe.org/cgi-bin/informatics_europe.cgi

[13] http://se.inf.ethz.ch/teaching/ss2007/252-0204-00/project.html

[14] http://www.swebok.org