Tackling Real World Complexity in a Software Engineering Student Project - An Experience Report

Christian Köppe
HAN University of Applied Sciences, Arnhem/Nijmegen, the Netherlands
christian.koppe@han.nl

Leo Pruijt
HU University of Applied Sciences, Utrecht, the Netherlands
leo.pruijt@hu.nl

Abstract

Developing large-scale complex systems in student projects is not common, due to various constraints like available time, student team sizes, or maximal complexity. However, we succeeded to design a project that was of high complexity and comparable to real world projects. The execution of the project and the results were both successful in terms of quality, scope, and student/teacher satisfaction.

In this experience report we describe how we combined a variety of principles and properties in the project design and how these have contributed to the success of the project. This might help other educators with setting up student projects of comparable complexity which are similar to real world projects.

Categories and Subject Descriptors K.3.2 [*Computers and Education*]: Computer and Information Science Education —Computer science education

General Terms Design, Education, Software Engineering Project

Keywords Realistic Projects, Design Principles

1. Introduction

In 2012, we ran a semester-project with a class of 25 third-year students of the undergraduate CS program at HU University of Applied Sciences in the Netherlands. The whole class had to develop one open source tool for software architecture compliance checking, which includes a high complexity and several architectural challenges. Addressing these challenges and the complexity required a good work division and a high level of collaboration between the student teams working on different parts of the system.

The project turned out to be successful. The results were beyond our initial expectations: the developed tool worked well and was of sufficient quality. It was applied (by the students themselves) at four IT departments of industry partners, demonstrating its applicability and relevance for professional software architects. Furthermore, the students were highly satisfied with the project. They stated that they had never worked so hard before in a project and that they never had learned that much. 7 students of the group indicated that they want to continue working on the tool in their spare time after the project. As teachers, we observed that the students grasped a variety of architectural concepts and applied them on several cognitive dimensions.

In this experience report we look back at the project and analyze what factors of our project design contributed to its successfulness. By sharing our experience we might help other educators to also set up such a software engineering project of high complexity and with motivated and satisfied students.

This paper is structured as follows: In Section 2 we provide some background information on student projects in computer science (and more specific software engineering) education and discuss influential aspects of such projects. Section 3 gives an overview of work that is addressing these aspects in various ways and levels. In the following Section 4, we describe the design of our project, hereby focusing on how we addressed the influential aspects by combining several known approaches. The execution of the project is presented in Section 5. In Sections 6 and 7, we present the results of the project and evaluate them respectively. The paper concludes with a summary of our experiences.

2. Student Projects in CS Education

The main intention of computer science (CS) education is to prepare students as good as possible for their later work. To achieve this goal, students need to acquire knowledge and skills relevant for CS-related jobs. Ideally, these knowledge and skills can be acquired in a way that helps the students to easily apply them in their later workplaces. This means that not only basic CS topics like programming in a specific language or working with relational databases need to be taught to the students—discipline knowledge and skills—but also software engineering practices and general high-level skills, as communication, goal-setting, self-management, and team work skills—the workplace knowledge and skills. This is also emphasized in the Computer Science Curricula 2013 report by ACM/IEEE, where the knowledge areas explicitly include social issues and professional practice as well as project management with all its facets (the latter one as part of the software engineering knowledge area) [7].

The necessary workplace knowledge and skills are in first instance taught in apart courses, but also later in the program addressed in larger student projects. These projects form a common way to integrate knowledge and skill acquisition and practice in a larger and more complex setting, hereby striving to be as similar as possible to real world projects.

However, most of these student projects are limited in scope in order to fit the academic setting, and students will face other problems when entering the working field and taking part in larger projects with (much) more complexity [1, 20].

presented at SPLASH-E 2015, October 26, 2015, Pittsburgh, PA, USA.

There are many interrelated aspects—or forces—that influence the design of student projects (and the systems to-be-developed in these projects):

- *Teamwork* - Students need to learn working in teams, as this is what they likely will do in the work field. Successful teamwork is dependent on many factors, like student motivation, level of relevant expertise, student's expectations, prior teamwork experience etc.
- *Complexity* - Project tasks should ideally be in the "zone of proximal development" (as defined by Vygotsky, see [2]). This means that they should not be routine or impossible tasks, but tasks the students are almost able to solve [2].
- *Scope* - Related to complexity, as scope—the quantitative aspect of complexity—of projects is also limited by the available amount of time and manpower due to the educational setting.
- *Independent vs. interdependent teams* - Usually student project teams work on independent standalone projects, but in companies even small teams most likely have to interact and cooperate with other teams when developing complex (parts of) systems.
- *Standalone system vs. part of larger system* - Related to interdependent teams is also the dependency of the to-be-developed system with other new or existing systems. This translates to negotiation of and adherence to interfaces, obligatory usage of frameworks, protocols etc.
- *Artificial vs. realistic projects* - The purpose of student projects is mostly the project itself and the skills and knowledge required to realize it. This is different from projects where the purpose is to realize a valuable system which will be used after the project execution. Projects where the results actually will be used are usually higher valued by students than throwaway-projects.

Many approaches as described in the literature address these aspects partly, these are reviewed in Section 3. However, we think that projects can be made more successful if most if not all of these aspects are addressed appropriately. This is unlikely to be realistic for projects in an early phase of the study, but projects situated in later phases of the study could benefit from such holistic approach.

3. Survey of Prior Efforts

Oakley et al. describe best practices for team work projects in general engineering education, which can also be applied to CSE [15]. Their main findings are that students' perception of how successful a course was is correlated with the satisfaction on their team experience. The team satisfaction increases with the amount of instructor guidance on teamwork, is dependent on the size of the teams (at least three to four), and the absence of slackers. Research also shows that instructor-formed teams mostly perform better than self-selected ones if the team members are diverse in ability levels [14]. Instructor-formed teams have a higher chance of diligent isolates (students who take over the tasks of others or work alone) and social loafers (also called slackers of free-riders) as suggested by Pieterse and Thompson [16]. This chance can be lowered by using well designed peer ratings [14].

Another important factor for the success of a student project is the student motivation. Furthermore important is how well problems—technical and teamwork-related—are handled during the project. Many of the real world problems only occur in larger projects with larger teams and higher complexity [20].

There are many descriptions of successful approaches on how to set up student projects of higher complexity as well as being motivating for students. Some of these approaches cover holistic aspects of real-world projects, while others focus more on either technical issues or project management issues [1, 5, 20]. In the next paragraphs we describe some of these approaches and pedagogic styles which are also related to our project design.

Schocken [20] identified four guiding design principles which can be applied to include a higher complexity in large-scale system projects. These principles should be applied while respecting the *upper limit* pedagogical value (related to Vygotsky's concept of Zone of Proximal Development): the students should be almost able to solve the required tasks [2]. Otherwise the tasks will be either routine or too complex. The four principles are:

- *Modularity* through decomposition of the target system into smaller and partly independent parts,
- *Abstraction* by excluding all unnecessary details,
- *Staging* through decomposition of the required tasks (in relation to the modularity), which allows a better planning, and
- *Focus* on the important aspects only.

Even though these principles were described by Schocken for, and successfully applied in, a course on the development of a general purpose computer system—including hardware and software—we found these principles also in other project design approaches (which are partly described below). We therefore think that these principles are generally helpful in realizing more complex and realistic software engineering student projects.

Many student projects used agile methods like Scrum (e.g. [4, 19]). Sanders reports positive experiences with using a modified version of Scrum in student projects. Sanders states that "Scrum enforces individual responsibility, forces the team members to set and achieve short term goals, gives the team a sense of ownership over the project, and helps them produce more work in the same amount of time. Scrum also forces students to monitor their progress, which in turn, helps them improve their estimation skills." [19]. Our assumption is that agile methods in general have these properties and are therefore suited for educational settings. They support the design principles of staging and focus. The standard short iterations also ensure a continuous activity of the students [10].

There are reports on successful collaboration with real world customers in student projects. In some cases the customers only *play* the role to make the project seem more realistic, as in [5]. Katz reports on successful collaboration with a real customer [8]. The motivation of the customers is a specific need and the lack of resources for running the project internally. We assume that the success of customer collaboration projects is the highest when it is the most realistic—a real world software system—and if it is of long term value for the customer. This also improves the motivation of students to work on the project, as the results will be used afterwards. The open source approach addresses this aspect implicitly, which might be another reason for its success.

Liu describes experiences with using open source systems for student projects in order to expose the students to large, complex, real-world software engineering projects [13]. An important tool are issue trackers, as they help student teams to follow the process (and therefore support the staging principle). In the educational team open-source development process (named GROw) proposed by Liu he suggests to cover the complete requirements specification and design, but to let students implement only some features, which limits the realized technical complexity in some ways. Süß and Billingsley use a modular open source project [21] and the students were given relatively clear features to implement, therefore supporting modularity, abstraction, and focus.

Deursen et al. describe a project that combined open source and real world customers [3]. Their students had to contribute to active open source projects from GitHub. Their contributions needed to be of a certain complexity, enforcing the students to actively work with various aspects of software architecture.

Principle/Aspect	Description
Modularity	Decomposition into smaller and partly independent parts..
Abstraction	Exclusion of unnecessary details.
Staging	Decomposition into smaller tasks for a better planning.
Focus	Not too many things at the same time.
Teamwork	Working in teams and not in isolation.
Instructor Guidance	Getting enough guidance from the instructors.
Real Customer	Building a system that fulfills a real need.
Open Source	The internals of the system are also visible for—and reusable by—others.
Continuous Integration	Preventing a big-bang integration at the end and ensuring an ongoing focus on the system as a whole.
Agile Process	Working incrementally in short iterations.

Table 1: Overview of principles and properties relevant for student projects in CSE

Süß and Billingsley also used continuous integration practices and automated metrics as approach [21]. According to their findings, this allows to expose students to more realistic problems and does not require significant additional staffing. However, Gestwicki states that it requires significant time investment to maintain such a system (the servers and their configuration) [4], so this approach seems to be dependent on the institutional infrastructure possibilities.

There also are reports on successful projects that do combine some of the earlier mentioned approaches. Gestwicki used continuous integration, test-driven development and an agile software process for setting up an undergraduate game development studio [4]. Süß and Billingsley used a modular open source project and continuous integration to make a software project more realistic [21]. Pieterse et al. combined different approaches to integrate software engineering, technical skills, and teamwork skills [17].

Table 1 provides an overview of all identified principles and properties, covering the aspects of influence on the project design as described in Section 3. In general, many of the described approaches are overlapping. However, we expect that using a combination of different approaches while following the design principles as we did in our project design, it is possible to execute student projects of real world complexity and with motivated and satisfied students.

4. Project Design

During project design, we discussed which goals we have for the project and what we need to do to achieve them. We wanted to tackle a high complexity—both technical and process related—in this project, where the complexity was higher than in other described projects and approaches (see survey in section 2). Our idea was to achieve this by combining the described approaches, so that they support the guiding design principles (in varying degrees). We hereby also built on the experience we made with a similar project given a year earlier. Most of the initial ideas worked well, while some had to be adapted during the execution. This will be discussed in more detail in the following sections.

The project was a single semester project lasting 19 weeks. There were 25 students and three instructors involved and it was part of the third year of an undergraduate computer science program. As stated before, the design of the project was initially based on the experiences we made with earlier projects. The results of these earlier projects, where 4 different teams developed 4 smaller-scoped architecture compliance checking tools with a limited set of requirements, were used to define a mature and demanding set of requirements for the new tool.

As we wanted to give the students a real world experience, all students had to collaborate to develop one tool. The teams were carefully formed by the teachers, taking care of diversity in ability levels as suggested by Oakley et al. [14]. We addressed the possible issue of diligent isolates and slackers by using peer ratings at the end of the project.

The design of the project followed the earlier described design principles: modularity, abstraction, staging, and focus. We used the designs of the earlier projects to define a modular start architecture that enabled us to assign clear responsibilities for specific system parts to different teams. The responsibilities were described in abstract high-level terms and therefore open to diverse ways of realization.

The main goal of the project was the development of an open source tool for Software Architecture Compliance Checking (SACC). (Software) Architecture Compliance Checking is defined by Knodel and Popescu as "a measure to which degree the implemented architecture in the source code conforms to the planned architecture" [9]. Due to the implicit complexity of this domain, we also considered the complexity of the tool to-be-developed as realistic and similar to systems usually developed in larger projects. Furthermore, having software architecture as application domain of the tool *and* as area needed for implementing the tool likely led to an improvement of knowledge acquisition in this area on multiple levels. This is an application of the pedagogical pattern MULTI-LEVEL ASSIGNMENT [12].

We divided the responsibilities into six main areas, each assigned to one student team of 4-5 members. These responsibilities allowed the teams to work independently on the internal workings of their components, but required the teams to communicate with other teams about e.g. the required interfaces or coding style guides. The six main areas are:

1. Architecture Definition (incl. mapping of physical elements to logical architecture elements)
2. Source Code Analysis - Java
3. Source Code Analysis - C#
4. Architecture Validation (for the actual compliance check)
5. General GUI & Control (incl. application flow, Maven- and Eclipse-plugins)
6. Graphics

Please note that the responsibilities were given to the students in more detail. Some teams also got project-related responsibilities like setting up a Version Control System or developing a test application which was used as benchmark for this project. Teams 2 and 3 started together with setting up a design that made it easy to include other programming languages without having to modify the general analysis mechanism.

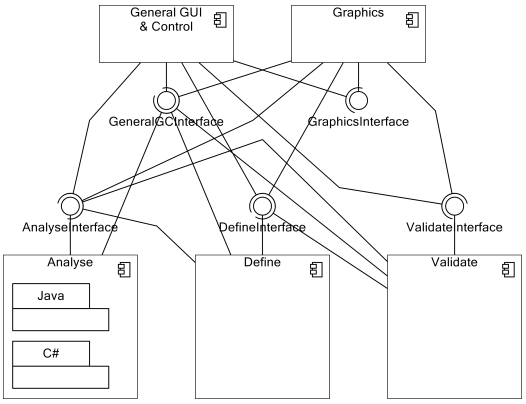

Figure 1: The start architecture

Some students got clearly defined roles that also constrained the communication ways. The architects of all teams were discussing all system-wide issues with each other and shared the outcomes afterwards with their teams. The project leaders (of the teams) made planning decisions for their own teams, but with respect to the overall progress of the total project.

Figure 1 gives an overview of the start architecture including components and general interfaces. Please note that these interfaces were not defined at this moment, as interface negotiation was an important aspect of the project.

We decided to use OpenUP[1] as software development process. OpenUP includes many aspects of agile methods like incremental and iterative working. This fits our project and supports the staging design principle. This was accompanied by weekly meetings, which were intended for presentation of interim results, discussions, making decisions, giving feedback, provision of feedback, or teamwork guidance if necessary. The combination of having to present interim results and getting new information needed for going on with the project was in our experience a good motivator for ensuring a continuous activity of the students, as also suggested in [10].

Initially we wanted to use continuous integration in combination with test-driven development. This would have required setting up a technical infrastructure, which is quite costly [4]. We therefore decided to deploy a general integration process and to emphasize test-driven development by letting the students write failing tests first before they start to work on the implementation.

Some parts of the project included aspects of problem-based learning (PBL), an instructional methodology intended to enhance learning by requiring learners to solve authentic and ill-structured problems [6]. We left it for example open how the general integration process could be designed best, how the source code analyzer should be implemented, or what the best visualization algorithm would be. These problems had to be explored by the students themselves while we as teachers only had the role of facilitator.

We found four industry partners that showed a high interest in such a tool and were willing to participate in the project as real customers. They offered to let the students apply the developed tool in their companies, giving them the possibility to check the architecture compliance of real commercial applications. This was of value for the companies, also in long terms as the development of the tool did not stop at the end of the project. Our idea was that this will also increase the motivation of the students.

[1] `http://epf.eclipse.org/wikis/openup/`

5. Project Execution

Beginning with the provided modular start architecture and the team responsibilities of the identified six main areas, the student teams first negotiated the interfaces between their own components and the components of the other student teams. These interfaces were shared and discussed with all students during a few sessions so that all students were aware of the status of interface negotiation. At the same time general decisions regarding the interfaces could be made collaboratively, like naming issues or the question if data transfer objects are used as protocol.

A first prototype of the system was implemented using service stubs—based on the negotiated interfaces—which led to a completely integrated and partly executable architecture prototype. After that, the student teams began to implement the stubs and deliver new versions at the end of each construction phase iteration. If adaptations to the interfaces were necessary, they were first discussed in the weekly meeting with all students and after that integrated in the system.

A group of students was asked to set up a version control system (VCS) and an infrastructure that offers some of the features of continuous integration. After doing some research the students chose for GIT and defined an underlying structure. This structure allowed the teams to work independently on their own system parts and to integrate them iteratively into one system. It was also decided to use the public service GitHub[2], which also meant that all source code is visible to the outside world and not only to the students themselves. As this service is freely available and all configuration tasks were done by the students themselves, it required only a small time investment from the teachers for supervising this part. The issue-tracking capabilities of GitHub were used for the registration of bugs and features to be implemented and were also used for planning the iterations during the construction phase of OpenUP.

For testing purposes, one student team took the responsibility to develop an application—initially in Java only—that included all previously described possible architecture violations. This system was then used for testing purposes and later accompanied with a similar application written in C#. Another aspect of quality assurance were code reviews, where teams had to publicly review the code of other teams including suggestions for improvement.

All students and teachers participated in weekly meetings where all relevant issues were discussed with the whole group and in many cases also collaboratively decided. After that the teachers met with the separate teams to discuss implementation issues only relevant for the teams themselves. These meetings were also used to take care of possible problems related to teamwork and to guide the students with solving them if necessary.

6. Project Result

6.1 The HUSACCT tool

The tool was implemented in Java and covers the whole process of (static) software architecture compliance checking. The size of the main tool at the end of the project was 33.143 non-comment lines of code in 486 classes and 98 packages (plus additional 70.896 ncloc of code generated by ANTLR). The code for the unit-tests comprises 6.623 ncloc in 66 classes and 16 packages. The following functionality has been implemented, the accompanying user interfaces are shown in Figure 2:

1. Defining an architecture - The logical architecture can be defined in the tool, incl. architectural rules like e.g. *Domain Layer is not allowed to use View Layer* or *All classes in module database have to implement the IDatabase interface*.

[2] `https://github.com/`

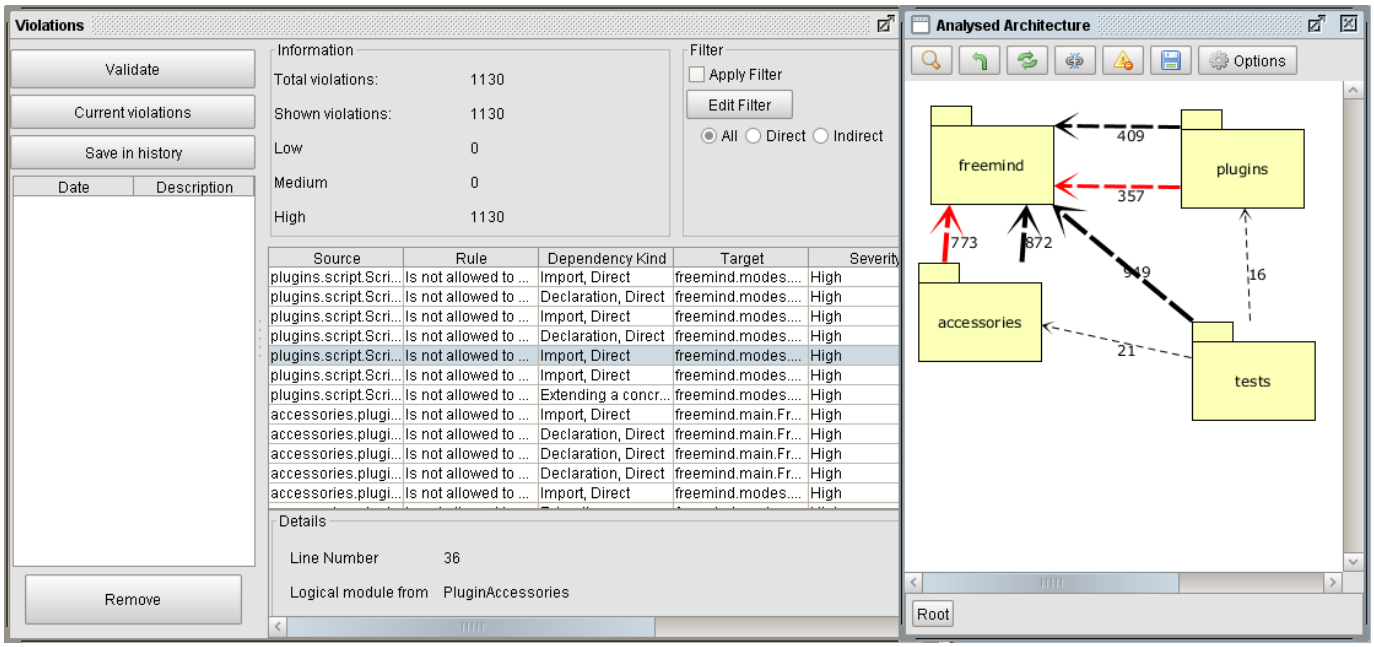

Figure 2: Screenshot HUSACCT - report and visualization of architecture violations

2. Analyzing an existing application - The source code of an existing Java or C# application is analyzed, then the physical structure of the application and dependencies between the system parts can be examined in a dependency browser, switching between different levels (from packages to classes).
3. Mapping the defined architecture - Map the physical entities found by analyzing to the defined logical modules.
4. Execute compliance check - The tool checks if there are violations against the earlier defined rules. The violations can be analyzed using a violation browser or exported as report in different formats.
5. Graphical representation - The physical structure, the defined architecture, and the result of the compliance checking including all violations can be graphically examined. This includes comfortable options like multi-zoom and optional hiding of system parts.

Additional to the realized functionality of the tool itself, the students also managed to develop an Eclipse-plugin. This plugin offered jumping directly to the source code where a violation occurred. Furthermore, an initial version of a Maven plugin was developed by one team, enabling that the software architecture compliance check can be integrated in a build process.

The tool HUSACCT itself was developed further, based on this result of this project, and is described in more detail in [18]. It is open source and can be downloaded from `http://husacct.github.io/HUSACCT/`. This site also contains the documentation and an introduction video.

6.2 Tool Application in Companies

The tool was applied by the students in different companies, among which RDW (the Dutch traffic authority), Belastingdienst (Dutch tax and finance authority), and Schiphol Airport. The companies provided an architecture description of one of their (sub-)systems and its source code. This was then analyzed by the students, the findings were summarized in a report and presented to the company's architects. The students had hereby to explain and defend their findings and possible suggestions for improvement given to the architects.

7. Evaluation

We experienced the project as successful. The feedback we got from the students at the end of the project was nearly completely positive. The students reported on a high work pressure, but also unanimously described it as the best project they did during their whole study. We also asked the students to evaluate the project, but the response was quite low (n=4). However, the results support our feelings about the successfulness of the project, as the relation with the professional practice was rated with an average of 8.3 and the importance of the project for the students' study with a 9.0 (on a scale of 1 to 10, where the highest possible rate is 10).

All students passed the project with an average grade of 8 (on a scale of 1 to 10). This grade is far above the usual average grades of other courses and projects. 7 students (27% of the participants) decided to work further on this project in their spare time and during holidays. This high motivation right after the end of the project can be seen as a clear indication that this project was highly valued and motivating.

Table 2 gives a summary of how the earlier described principles and properties were realized in our project. It also describes additional aspects that in our experience contributed to the success of the project. The applied principles and approaches are evaluated in more detail in the remainder of this section.

The usage of service stubs as dummy implementations of the negotiated interfaces played an important role in the early iterations of this project, as they offered an immediate integration of all system parts. The interfaces followed the abstraction principle and supported modularity, as they hid all details of the modules and therefore allowed independent development of them. They also supported staging, both on project level and on team level, as they allowed that planning decisions could be made per team.

Staging was also realized by the application of OpenUP as development process and its artifacts in this project. The definitions

Principle/Aspect	Application in project
Modularity	Modular architecture, where the modules have clearly defined responsibilities and interfaces. Each group was responsible for one module and the negotiation of the interfaces with other modules.
Abstraction	Only knowledge about the internals of the group's own module/s is needed, as well as on the required and provided interfaces of them.
Staging	The first smaller tasks were the negotiation of the required and provided interfaces, which was followed by their incremental implementation. All tasks were small enough to being finished in less than two weeks and therefore always fitted into one iteration.
Focus	Clear goals were defined per iteration. As the first goals were the negotiation of the interfaces, the students could afterwards focus on implementing these interfaces one by one. Service stubs were used for the not yet implemented interfaces so that less dependencies from other modules were present.
Teamwork	The students had to work in teams of 4-5 students with a total of 6 teams. Besides the weekly meetings they had to plan their activities themselves. As all teams worked on one system, the whole class also was seen as a team with a common goal and collective ownership of the result.
Instructor Guidance	Weekly meetings of the whole class where the instructors always were present and offered general guidance plus additional meetings with each team and where all team-specific issues were discussed.
Real Customer	Four well-known companies showed interest in the tool and offered the possibility to apply it at their location using real production systems.
Open Source	The project was openly hosted on GitHub. Knowing that everybody can see their code created in our experience a higher awareness on code quality.
Continuous Integration	One team was responsible for integrating and testing (using the unit tests provided by the other teams) all work at the end of an iteration, leading to a well-tested system every two weeks.
Agile Process	OpenUP as process with iterations of two weeks. The elaboration phase (with the architectural prototype as milestone) lasted 2 iterations, as there was a strong focus in this project on architectural aspects. All students had the role of team member, while one student of each team also fulfilled the role of architect. The architects also discussed higher level issues in a smaller group.

Table 2: Summary of how the principles and properties were applied in the project.

of the work products to be delivered helped with planning the iterations. Furthermore, OpenUP encourages incremental and iterative working on a system as well as other agile practices like test-driven development. Successful usage of agile methods in student projects has been reported. In our believe OpenUP can be added to this list.

Schocken describes some concessions that help with realizing the principle of focus: no exceptions, no efficiency, no special features, and no design uncertainty. This might be valid for building computer systems, but was different in our project. We consciously included exception handling (and unit-testing) in our project in order to increase the quality of the final tool. We also consciously worked with a high level of design uncertainty as part of our problem-based learning approach. This was because design and architecture were skills to be improved in this project. We interpret the focus principle in our case as having the students knowing at all times what they have to do or what they have to discuss and decide.

The point that the tool had to be open source, and therefore visible to the outside world, seemed to motivate the students to regularly improve the code. That they were administrating the VCS and handled the integration in an open and agreed on way by themselves might have led to a higher commitment to the project.

There was a high amount of instructor guidance throughout the whole project as also suggested by Oakley et al. [15], especially in the weekly meetings with the whole class and with all teams. This guidance comprised of supporting the students in making decisions. Helping students with solving teamwork-related problems was surprisingly rarely necessary. One reason could be the experience the students had with many earlier teamwork assignments. We used pattern mining of the students' collaboration patterns [11] to create an higher awareness of the students for possible problems and the way these problems were solved earlier by the students themselves or their peers. This combination of both instructor guidance and the mining and application of the collaboration patterns worked in our opinion complementary and intensifying.

The inclusion of the students in the decision making during the project was in our opinion also a motivating factor. The fact that they had to negotiate the interfaces between the modules themselves and had to collaborate with other teams to reach this goal made them also the owners of the results. On the other hand were they exposed to high technical complexity and additional, because of the interface evolution, also to problems of project management, planning and system integration.

The cooperation with the companies was made easier by the fact that the students developed an open source tool and that in the future another group of students will go on with further developing and extending the tool (but likely in a different project setting than the presented one, as some important aspects such as interface negotiation and the building of something new will be missing). So it was seen as a long term investment by the companies, and not a one-off project. The feedback of all companies was positive. They valued the quality of the advice the students gave to the present company employees (mostly software architects). This advice was based on their findings when checking one or more of the company's systems on their compliance with the intended architecture. The involved software architects furthermore were impressed by the high level on which the students discussed architectural issues during the visits. Some students with jobs beside the university also introduced the tool in the companies where they worked.

Important part of the project was the explorative character of the project as part of the problem-based learning design. There were many assignments included were students had to sort things out by themselves and present the results to the group in the weekly meetings. We assume that this sharing of self-acquired knowledge led to a higher identification and commitment of the students.

The modularity and abstraction principles were realized in a problem-based learning way: the students were only given some high level requirements (as shown in Figure 1) and guidelines. This required them to identify and evaluate possible solutions in a

collaborative way. We believe that this approach supported a better acquisition of the technical skills.

8. Conclusion

In this paper we reported on a student software engineering project that we experienced as successful. The design of this project combined a variety of principles and properties, which was in our opinion the reason for its success.

The four principles described by Schocken—modularity, abstraction, staging, and focus—were applied for addressing the high complexity of the project. These principles were slightly adapted to fit a software engineering project as described in Section 4. Additionally to these principles, we found that real-world projects where also industry partners are involved are highly motivating for students. If these real-world projects are also open source projects which are publicly available and have a lifespan that extends the students project, then an extra motivation, triggered by identification of the students with the project, and a higher commitment to the project was observed by us. Agile methods like OpenUP are well suited for student projects and help students to experience and handle a realistic project complexity.

We believe that explicitly addressing such variety and properties can indeed help with realizing systems of larger-scale, real world systems of higher complexity. This would make such projects similar to projects the students will likely be involved in after their study, and is therefore in our opinion a better preparation for their later career.

Acknowledgements

First and most we want to thank all students who followed this semester. We also want to thank Wiebe Wiersema, lector at the HU University of Applied Sciences for his support during the semester. Last, but not least, we want to thank the companies which helped in making this a real project for the students.

References

[1] G. Bavota, A. De Lucia, F. Fasano, R. Oliveto, and C. Zottoli. Teaching software engineering and software project management: an integrated and practical approach. In *Proceedings of the 2012 International Conference on Software Engineering*, pages 1155–1164, Zurich,Switzerland, June 2012. ISBN 978-1-4673-1067-3. URL http://dl.acm.org/citation.cfm?id=2337223.2337375.

[2] J. Bennedsen and O. Eriksen. Applying and developing patterns in teaching. In *33rd Annual Frontiers in Education, 2003. FIE 2003.*, volume 1, pages 2–7. IEEE, 2003. ISBN 0-7803-7961-6. . URL http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1263358.

[3] A. V. Deursen, A. Nederlof, and E. Bouwers. Teaching Software Architecture: with GitHub!, 2013. URL http://avandeursen.com/2013/12/30/teaching-software-architecture-with-github/.

[4] P. Gestwicki. The entity system architecture and its application in an undergraduate game development studio. In *Proceedings of the International Conference on the Foundations of Digital Games - FDG '12*, pages 73–80, New York, New York, USA, May 2012. ACM Press. ISBN 9781450313339. . URL http://dl.acm.org/citation.cfm?id=2282338.2282356.

[5] M. Gnatz, L. Kof, F. Prilmeier, and T. Seifert. A Practical Approach of Teaching Software Engineering. In *Proceedings of the 16th Conference on Software Engineering Education and Training*, page 120, Mar. 2003. ISBN 0-7695-1869-9. URL http://dl.acm.org/citation.cfm?id=794194.794947.

[6] W. Hung, D. Jonassen, and R. Liu. Problem-based learning. In J. Spector, J. G. van Merriënboer, M. Merrill, and M. Driscoll, editors, *Handbook of research on educational communications and technology*, pages 1503–1581. 2007. URL http://faculty.ksu.edu.sa/Alhassan/Handbookonresearchineducationalcommunication/ER5849x_C038.fm.pdf.

[7] Joint Task Force on Computing Curricula ACM/IEEE. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, Inc, Jan. 2013. ISBN 9781450323093. . URL http://dl.acm.org/citation.cfm?id=2534860.

[8] E. P. Katz. Software Engineering Practicum Course Experience. In *2010 23rd IEEE Conference on Software Engineering Education and Training*, pages 169–172. IEEE, Mar. 2010. ISBN 978-1-4244-7052-5. . URL http://dl.acm.org/citation.cfm?id=1796177.1796659.

[9] J. Knodel and D. Popescu. A Comparison of Static Architecture Compliance Checking Approaches. In *Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*, pages 12–21. IEEE, Jan. 2007. ISBN 0-7695-2744-2. . URL http://dl.acm.org/citation.cfm?id=1264360.1264992.

[10] C. Köppe. Continuous Activity - A Pedagogical Pattern for Active Learning. In *Proceedings of the 16th European Conference on Pattern Languages of Programs - EuroPLoP '11*, volume 2011, Irsee, Germany, 2011. ACM Press. ISBN 9781450313025. . URL http://dl.acm.org/citation.cfm?doid=2396716.2396719.

[11] C. Köppe. Using pattern mining for competency-focused education. In *Proceedings of Second Computer Science Education Research Conference - CSERC '12*, pages 23–26, Wroclaw, Poland, 2012. ACM Press. ISBN 9781450318587. . URL http://dl.acm.org/citation.cfm?doid=2421277.2421280.

[12] C. Köppe and L. Pruijt. Improving Students' Learning in Software Engineering Education through Multi-Level Assignments. In *Proceedings of Fourth Computer Science Education Research Conference, CSERC14*, Berlin, Germany, 2014.

[13] C. Liu. Enriching software engineering courses with service-learning projects and the open-source approach. In *Proceedings of the 27th international conference on Software engineering - ICSE '05*, page 613, New York, New York, USA, May 2005. ACM Press. ISBN 1595939632. . URL http://dl.acm.org/citation.cfm?id=1062455.1062566.

[14] B. Oakley, R. M. Felder, R. Brent, and I. Elhajj. Turning Student Groups into Effective Teams. *Journal of Student Centered Learning*, 2 (1):9–34, 2004. URL http://oxfordbrookes.ac.uk/services/ocsld/resources/group_work/turnin_student_groups_into_effective_teams.pdf.

[15] B. A. Oakley, D. M. Hanna, Z. Kuzmyn, and R. M. Felder. Best Practices Involving Teamwork in the Classroom: Results From a Survey of 6435 Engineering Student Respondents. *IEEE Transactions on Education*, 50(3):266–272, Aug. 2007. ISSN 0018-9359. . URL http://dl.acm.org/citation.cfm?id=2254939.2256197.

[16] V. Pieterse and L. Thompson. Academic alignment to reduce the presence of 'social loafers' and 'diligent isolates' in student teams. *Teaching in Higher Education*, 15(4):355–367, 2010.

[17] V. Pieterse, L. Thompson, L. Marshall, and D. M. Venter. An Intensive Software Engineering Learning Experience. In *Proceedings of the 2nd Computer Science Education Research Conference, CSERC 2012*, pages 47–54, Wroclaw, Poland, 2012.

[18] L. Pruijt, C. Köppe, J. van der Werf, and S. Brinkkemper. HUSACCT: Architecture Compliance Checking with Rich Sets of Module and Rule Types. In *Proceedings of the 29th IEEE/ACM International Conference on Automated Software Engineering (ASE 2014)*, pages 1–4, 2014. ISBN 9781450330138. . URL http://dl.acm.org/citation.cfm?id=2642937.2648624.

[19] D. Sanders. Using Scrum to manage student projects. *J. Comput. Small Coll.*, 23(1):79, 2007. ISSN 1937-4771. URL http://portal.acm.org/citation.cfm?id=1289280.1289295.

[20] S. Schocken. Taming complexity in large-scale system projects. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education - SIGCSE '12*, page 409, New York, New York,

USA, Feb. 2012. ACM Press. ISBN 9781450310987. . URL http://dl.acm.org/citation.cfm?id=2157136.2157259.

[21] J. G. Süβ and W. Billingsley. Using continuous integration of code and content to teach software engineering with limited resources. In *Proceedings of the 2012 International Conference on Software Engineering*, pages 1175–1184, Zurich, Zwitserland, June 2012. ISBN 978-1-4673-1067-3. URL http://dl.acm.org/citation.cfm?id=2337223.2337377.