Reaching Steady State in Software Engineering Project Courses

Dora Dzvonyar
Department of Informatics
Technical University of Munich
Munich, Germany
dzvonyar@in.tum.de

Bernd Bruegge
Department of Informatics
Technical University of Munich
Munich, Germany
bruegge@in.tum.de

***Abstract*—Project courses provide students with a hands-on experience of software engineering practices in industry and prepare them for their later career. The first weeks of such a course typically involve change and uncertainty and are therefore challenging for students: project teams have to understand requirements that are often unclear, learn to work together and communicate as a team, as well as become accustomed to processes and workflows used in the course. In this paper, we summarize our experiences of how student teams master change at the beginning of a project course to reach a steady state in which the level of uncertainty is manageable. We describe the typical stages of a student project, go into detail on the challenges during the Launch Phase and provide suggestions of how instructors can help students overcome those challenges. We report both on our experience conducting project courses with clients from industry since 2008 and on the results of a qualitative evaluation of one instance of a large multi-project capstone course.**

I. Introduction

Project work helps students apply and develop their software engineering skills in a realistic environment, thus preparing them for their career in industry [1]. Project courses often involve industry partners to increase students' motivation and maximize educational effectiveness: by working on real-world problems, the course participants develop both their technical and soft skills, gain contacts to potential mentors in industry and get the satisfaction of working on a project that is not a purely academic exercise [2], [3].

However, the realistic setting also introduces challenges for the participants. The beginning of a project course is particularly difficult, as students have to familiarize themselves with development processes and tools, get to know the project's problem domain, understand requirements, and get acquainted with their fellow team members [4]. Some projects are more clearly defined than others, resulting in different initial situations and varying levels of uncertainty for the teams [5]. Instead of shielding students from these challenges, instructors should design environments that enable project teams to overcome them and learn from experience [6], [7]. This task is challenging and time-consuming, especially with large courses or inexperienced instructors [8], [9].

In this paper, we report on our experiences conducting large capstone courses with industry partners since 2008. We first describe our course structure and the phases a project typically goes through between the course kickoff and its termination. We then go into detail about the challenges students face in the Launch Phase during the first weeks of the course. While we do not aim to provide solutions, we give recommendations and examples of how instructors can support the teams in handling these challenges. Finally, we present and discuss the results of a qualitative analysis of students' perspective of the first weeks of a multi-project capstone course.

II. Course Structure and the Project Lifecycle

This publication is based on our experience conducting the large multi-project capstone course *iPraktikum* with customers from industry and 10-12 projects per semester. We have described the course structure, our teaching methods and other aspects of the course in further detail in [5], [8], [10]–[15].

Each instance of the course runs for one semester and starts with a Kickoff meeting in which all industry partners present their problem to all participants. The instructors of the course allocate project teams based on students' priorities with the goal of creating balanced teams regarding experience as well as other factors described in [13]. Each project team consists of 6-10 developers, one team coach who is an experienced student having participated in the course as a developer before, and a project leader who is a PhD candidate.

After the Kickoff, the teams go through a period of 2-3 weeks, which we call *Sprint 0*. During this time, the focus is on reducing uncertainty in the project: the teams discuss the requirements and verify them with the customer, conduct team building activities, set up the tools and test important workflows. Sprint 0 can be of varying duration depending on the initial level of definition of the project as well as the technological challenges involved.

Following Sprint 0, teams start developing based on our agile process model Rugby [10]. We hold two major events with presentations from all teams: in the Design Review 8 weeks after the beginning of the course, all teams present the result of their requirements analysis and system design activities, and the Client Acceptance Test marks the last milestone of the course, after which teams wrap up their project by finishing the documentation and conducting a retrospective.

The aforementioned milestones of the course are visualized in Figure 1 along with the typical phases we experience in the

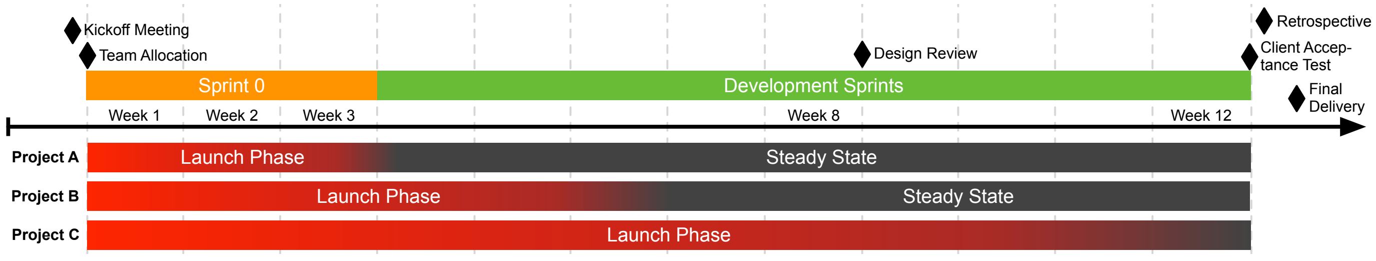

Fig. 1. Course timeline and project phases (amount of red color signifies level of uncertainty).

projects. After the initial team allocation, projects go through a *Launch Phase*, a period of typically high uncertainty in which students have to familiarize themselves with a large amount of information both general to software engineering as a discipline, as well as specific to the problem domain of their project. As the teams gradually reduce uncertainty, they reach what we call a *Steady State*: while requirements can be refined or disambiguated during iterations, there should not be any changes that fundamentally alter or even derail the project, setting the team back by several weeks [10].

The duration of the Launch Phase depends on the initial level of definition of a project. We visualize three examples in Figure 1. Project A reaches steady state during Sprint 0, which is the case for roughly half of the projects in the iPraktikum. This project is adequately defined by the customer so that the team can fully understand the requirements and decide on the initial system architecture as well as the main technologies involved, enabling them to start development with a strongly reduced level of uncertainty. In the case of Project B, the Launch Phase outlasts Sprint 0 and overlaps with the development sprints. This can be the case if the problem is more abstract or involves technological challenges. We describe the varying initial situations of student projects in our courses in further detail in [5]. In the past, an example of this type of project involved a NFC sensor that collected shock and humidity data that was not yet fully developed by the manufacturer, requiring the team to reverse-engineer the protocol used to read and write sensor data before they were sure that they could use it for their purposes.

Finally, as Team C shows, some projects might not even reach a steady state until the end of the course due to extremely high uncertainty and fundamental changes to the project throughout the semester. This situation should be prevented to avoid frustration of students and has happened very rarely in our courses. In such a case, the instructor should intervene by, e.g., limiting the scope of the project.

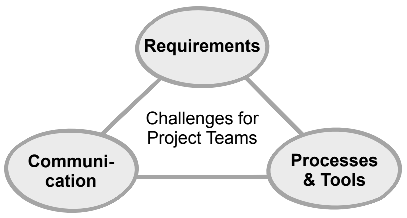

Fig. 2. Challenges for project teams in capstone courses.

III. Challenges of the Launch Phase

We have grouped the typical challenges of project teams during the Launch Phase into three categories visualized in Figure 2 and explain each in further detail in this section.

A. *Requirements*

Uncertainty resulting from factors around requirements is regarded as one of the highest risk factors in software projects [16], thus it is not surprising that we also experience it as one of the biggest challenges in our capstone courses. Involving industry partners in a course brings a more realistic experience for students, but it also introduces uncertainty compared to a project that is fully defined by the instructor from the beginning [17]. For instance, some customers do not have a technical background and thus are not able to specify the requirements sufficiently to be used as a basis for development. Students have to *understand* the visionary scenarios given by the customer, *clarify* ambiguous requirements, as well as *prioritize* and *negotiate* requirements.

In projects involving emerging or unstable technologies, students need to do research to decide on frameworks or products to use, and they might even find that the results envisioned by the customer are not entirely feasible. Technological challenges have been previously identified as a risk factor [18] and source of demotivation [19] in student projects. To name an example from a past course, one project aimed at saving blood glucose levels and other data of diabetic patients in Apple's secure HealthKit database had to change direction when the team found out that the API did not support all kinds of measurements; they had to implement their own database and merge the data from both sources for analysis.

In order to help students overcome challenges surrounding requirements, instructors should encourage frequent communication between the customer and the team in the beginning of the project. We place a strong emphasis on communication models in Sprint 0 because we believe that they enable targeted, rich discussions between stakeholders, foster the creation of a shared mental model and expose misunderstandings in requirements [5], [15]. In addition, we introduced a continuous prototyping workflow and tool that allows developers to easily deliver even early mock-up prototypes to stakeholders, gather feedback and iteratively transition to the delivery of hybrid and code prototypes through the same channel [20]. Our student teams use the workflow to regularly deliver potential product increments to their customer and verify the progress with them.

B. *Processes & Tools*

As instructors, our goal is to introduce students to the processes and tools used in industry so that they can gain valuable knowledge for their later career [17]. However, this also increases the workload for students especially at the beginning of the course, as they need to familiarize themselves with a variety of new topics, from the principles of agile software development through workflows such as continuous delivery, software modeling, prototyping, code reviews or meeting management [10]. The tools used in the course are also new to most students: depending on their prior experience, they have to get used to an issue tracker, repository and merge request system, continuous integration system and more. Problems with tools have also been named as one of the top four risks in student projects, e.g., [18].

As these topics are relevant to all teams, we use centralized teaching methods to transfer the necessary knowledge to all participants. We hold a weekly course-wide lecture during the first month of the course in which we introduce all students to the principles of agile software development, continuous delivery and UML modeling, including the tools supporting these workflows. In addition, we have cross-functional teams that consist of one team member per project and concentrate on the major workflows of the course [11]. For instance, we have a cross-functional team *Release Management* that is concerned with the continuous integration and delivery pipelines of the teams: the cross-project members get in-depth knowledge about the workflows, set up the tools in their team, and get support from coaches and instructors as needed. They then present the information that is important for the other team members and make sure their team correctly adopts the workflows. This enables us to disseminate knowledge through an additional channel without burdening all team members with detailed information.

After giving course participants the necessary information, instructors should make sure that the workflows are correctly applied in the teams, which can be a time-consuming task with multiple projects. We use a set of metrics to get a high-level view over the projects and only investigate further if the metrics show issues, making our courses more manageable [8].

C. *Communication*

Issues around communication, team work and personal relationships are also among the top challenges in student projects [18], [21]. We do not find this surprising: on top of the high workload in terms of understanding requirements and getting used to processes and tools, students need to get acquainted and start working productively with their fellow team members, most of whom they did not know prior to the course. Cultural and language issues can also get in the way of establishing team synergy [19]. Customer communication is often an additional challenge if the customer is rarely available for questions or does not have a technical background.

Additional complexity is introduced by the fact that the development teams do not spend every day working at the same location: many course participants work next to their studies or attend various other courses. This results in scheduling difficulties when trying to find a suitable time for the team meeting or time to work together on the project. Scheduling has also been reported by fellow researchers as a major source of annoyance in student projects [18].

As instructors, we take cultural and scheduling factors into account when composing the teams [13]. However, we cannot do much to support teams in overcoming these hurdles in a centralized way, as establishing communication processes and synergy is a very team-individual process. Instead, we use the coaches and project leaders who work closely with the developers. We urge the coaches to organize at least one icebreaker event in Sprint 0, in which they participate in a joint activity outside of the course context to get to know each other. Furthermore, we teach them facilitation methods such as dealing with shy team members or agile retrospectives to establish a constructive feedback culture and react to issues arising in their team.

IV. Case Study

In order to get a more accurate picture of how students perceive the challenges in the Launch Phase, we conducted a qualitative evaluation in one instance of the iPraktikum. 78 developers and 12 team coaches participated in the course, which ran between October 2017 and February 2018. We sent all participants a repeat questionnaire one, two, four and six weeks after the Kickoff on 19 October and asked them to describe at least one problem or challenge they had been facing in their team since the last time they received the questionnaire. Students had one week to answer the open-ended question, with the promise that we would analyze their responses anonymously and not use them for assessing their contribution to the course in any way. The response rate decreased slightly with each round, from 93.3% down to 70% with the last questionnaire, which was sent out two weeks before the Design Review. We coded the responses using a provisional coding technique [22], grouping codes into the three categories described in Section III. Figure 3 shows the frequency of each code in the responses by questionnaire.

Our results show that students perceived unclear requirements as a challenge at the very beginning of their project, with 17.9% and 15.7% of responses containing a reference to this issue after week 1 and 2 of the course. The frequency of this code dropped below 5% in the following questionnaires, suggesting that most students felt they had reduced uncertainty in terms of unclear requirements after Sprint 0. However, they began to sense issues with the technologies involved in their project from week 2 onward. Communication with fellow team members is the code with the highest frequency in all questionnaires, ranging from 17.5% to 27.1% of responses. Another challenge in this category was "Synergy & Investment", the code under which we group responses referring to a lack of trust or team spirit as well as differences in dedication and time investment of team members; the frequency of this code increased dramatically in the last questionnaire, which was

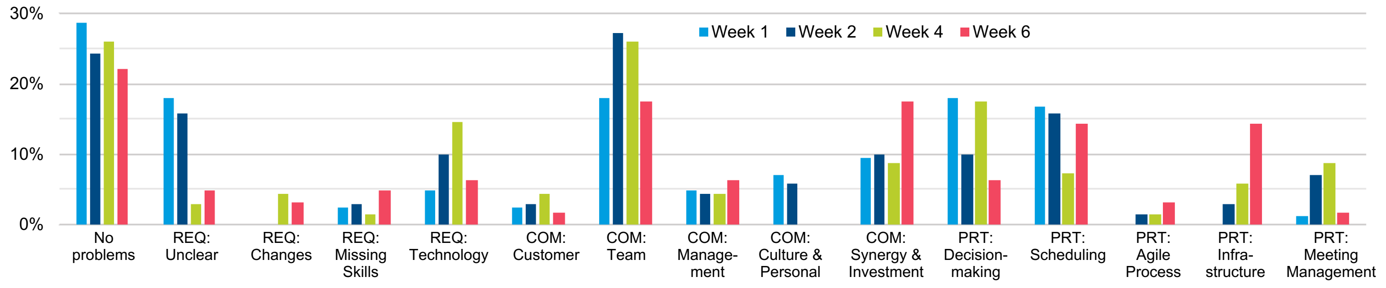

Fig. 3. Frequency of codes in students' responses, categorized by challenges in Requirements (REQ), Communication (COM) and Processes & Tools (PRT).

sent out during a high-workload period filled with preparations for the first major event. In terms of processes and tools, students perceived challenges around the decision-making practices of their team, which includes answers referring to unnecessarily long discussions or a lack of decisiveness and compromise. Issues around finding times to meet and work as well as punctuality and time management, grouped under the code "Scheduling", were also recurring in the projects, which is in line with the findings of, e.g., [18]. Problems referring to the course infrastructure increased as the course progressed and students had to integrate and deliver a growing amount of features to their customer.

Overall, our case study provided us with a deeper understanding of students' experience at the beginning of our project course. While it is debatable whether students can accurately assess and predict problems in a software project, we were interested in their subjective perception of challenges and how these evolve over time. Each of our three clusters Requirements, Communication and Processes & Tools posed challenges to participants of the course. More analysis is necessary to examine the impact of teaching methods described in Section III on students' ability to tackle these challenges.

V. Conclusion

In this publication, we described how projects in capstone courses reach a steady state by reducing change and uncertainty, and the challenges they need to overcome to do so. We summarized how we experience the evolution of projects throughout our multi-project capstone course and described typical challenges along with suggestions of how instructors can empower students to tackle those challenges. Finally, we gained a deeper understanding of students' experience during the first weeks of the course.

We do not believe that there is a one-size-fits-all toolbox for instructors to ensure that student projects run smoothly, as the course structure, nature of the projects and background of the people involved can differ widely. While we did not provide concrete solutions, we hope that by reflecting on the Launch Phase from both an instructor's and a student's perspective, we inspired instructors to help students overcome the challenges in their projects. In the future, we want to examine the impact of instructors' teaching on the ability of students to tackle these issues, and we are also analyzing the influence of team composition on teamwork and students' experience at the beginning of the course.

References

[1] C. Ghezzi and D. Mandrioli, *The Challenges of Software Engineering Education*. Springer Berlin Heidelberg, 2006, pp. 115–127.
[2] C. Wohlin and B. Regnell, "Achieving industrial relevance in software engineering education," in *CSEET '99*. IEEE, 1999, pp. 16–25.
[3] R. Chatley and T. Field, "Lean Learning - Applying Lean Techniques to Improve Software Engineering Education," in *ICSE '17*. IEEE, 2017, pp. 117–126.
[4] B. Boehm and D. Port, "Educating software engineering students to manage risk," in *ICSE '01*. IEEE, 2001, pp. 591–600.
[5] D. Dzvonyar, S. Krusche, and L. Alperowitz, "Real Projects with Informal Models," in *MODELS '14 EduSymp*. ACM, 2014.
[6] S. A. Hernandez, "Team learning in a marketing principles course: Cooperative structures that facilitate active learning and higher level thinking," *Journal of Marketing Education*, vol. 24, no. 1, pp. 73–85, 2002.
[7] A. B. Kayes, "Experiential learning in teams," *Simulation & Gaming*, vol. 36, no. 3, pp. 330–354, sep 2005.
[8] L. Alperowitz, D. Dzvonyar, and B. Bruegge, "Metrics in Agile project courses," in *ICSE '16*. ACM, 2016, pp. 323–326.
[9] D. Coppit and J. M. Haddox-Schatz, "Large team projects in software engineering courses," in *SIGCSE '05*. ACM, 2005, p. 137.
[10] S. Krusche, L. Alperowitz, B. Bruegge, and M. O. Wagner, "Rugby: an agile process model based on continuous delivery," in *RCoSE '14*. ACM, 2014, pp. 42–50.
[11] B. Bruegge, S. Krusche, and M. Wagner, "Teaching Tornado: from communication models to releases," in *MODELS '12 EduSymp*. ACM, 2012, pp. 5–12.
[12] D. Dzvonyar, D. Henze, L. Alperowitz, and B. Bruegge, "Algorithmically Supported Team Composition for Software Engineering Project Courses," in *EDUCON '18*, accepted for publication.
[13] D. Dzvonyar, L. Alperowitz, D. Henze, and B. Bruegge, "Team Composition in Software Engineering Project Courses," in *SEEM '18*, submitted for publication.
[14] S. Krusche, D. Dzvonyar, H. Xu, and B. Bruegge, "Software Theater - Teaching Demo Oriented Prototyping," *Transactions on Computing Education*, 2018.
[15] L. Alperowitz, J. O. Johanssen, D. Dzvonyar, and B. Bruegge, "Modeling in agile project courses," in *MODELS '17 Satellite Events*. CEUR-WS.org, 2017, pp. 521–524.
[16] L. Wallace, M. Keil, and A. Rai, "Understanding software project risk: a cluster analysis," *Information & Management*, vol. 42, no. 1, pp. 115–125, 2004.
[17] B. Boehm, A. Egyed, D. Port, A. Shah, J. Kwan, and R. Madachy, "A stakeholder win-?win approach to software engineering education," *Annals of Software Engineering*, vol. 6, no. 1/4, pp. 295–321, 1998.
[18] T. Ahtee and T. Poranen, "Risks in Students' Software Projects," in *CSEET '09*. IEEE, 2009, pp. 154–157.
[19] I. Bosnić, I. Čavrak, M. Orlić, M. Žagar, and I. Crnković, "Student motivation in distributed software development projects," in *CTGDSD '11*. ACM, 2011, pp. 31–35.
[20] L. Alperowitz, A. M. Weintraud, S. C. Kofler, and B. Bruegge, "Continuous prototyping," in *RCoSE '17*. IEEE, 2017, pp. 36–42.
[21] C. Bastarrica, D. Perovich, and M. M. Samary, "What can Students Get from a Software Engineering Capstone Course?" 2017.
[22] M. B. Miles, A. M. Huberman, and J. Saldana, *Qualitative data analysis*. Sage, 2013.