Designing and Implementing an Environment for Software Start-up Education: Patterns and Anti-Patterns

Fabian Fagerholm, Arto Hellas, Matti Luukkainen, Kati Kyllönen, Sezin Yaman, Hanna Mäenpää

PII: S0164-1212(18)30174-2
DOI: https://doi.org/10.1016/j.jss.2018.08.060
Reference: JSS 10215

To appear in: *The Journal of Systems & Software*

Received date: 2 November 2017
Revised date: 18 August 2018
Accepted date: 31 August 2018

Please cite this article as: Fabian Fagerholm, Arto Hellas, Matti Luukkainen, Kati Kyllönen, Sezin Yaman, Hanna Mäenpää, Designing and Implementing an Environment for Software Start-up Education: Patterns and Anti-Patterns, *The Journal of Systems & Software* (2018), doi: https://doi.org/10.1016/j.jss.2018.08.060

Highlights

- 16 patterns and 16 anti-patterns for experiential project based start-up education.
- Physical and virtual environments as well as pedagogics are covered.
- Student, teacher, and customer perspectives are considered.
- Based on seven years of experience with educational reform and course development.

Designing and Implementing an Environment for Software Start-up Education: Patterns and Anti-Patterns

Fabian Fagerholm[1,*], Arto Hellas[1], Matti Luukkainen[1], Kati Kyllönen[1], Sezin Yaman[1], Hanna Mäenpää[1]

[a]*Department of Computer Science, University of Helsinki, Helsinki, Finland*

Abstract

Today's students are prospective entrepreneurs, as well as potential employees in modern, start-up-like intrapreneurship environments within established companies. In these settings, software development projects face extreme requirements in terms of innovation and attractiveness of the end-product. They also suffer severe consequences of failure such as termination of the development effort and bankruptcy. As the abilities needed in start-ups are not among those traditionally taught in universities, new knowledge and skills are required to prepare students for the volatile environment that new market entrants face. This article reports experiences gained during seven years of teaching start-up knowledge and skills in a higher-education institution. Using a design-based research approach, we have developed the Software Factory, an educational environment for experiential, project-based learning. We offer a collection of patterns and anti-patterns that help educational institutions to design, implement and operate physical environments, curricula and teaching materials, and to plan interventions that may be required for project-based start-up education.

Keywords: start-up education, project-based learning, experiential learning, curriculum, software engineering, computer science

*Corresponding author

Email addresses: fabian.fagerholm@helsinki.fi (Fabian Fagerholm), arto.hellas@helsinki.fi (Arto Hellas), matti.luukkainen@helsinki.fi (Matti Luukkainen), kati.a.kyllonen@helsinki.fi (Kati Kyllönen), sezin.yaman@helsinki.fi (Sezin Yaman), hanna.maenpaa@helsinki.fi (Hanna Mäenpää)

1. Introduction

Entrepreneurship has been identified as a key solution for future employment. For example, to support the development of entrepreneurial skills, knowledge, and attitudes, the European Commission's Entrepreneurship 2020 Action Plan puts special emphasis on entrepreneurial education [1]. Even though software start-ups may be prime examples of the entrepreneurial world, higher-education institutions must still decide on their role in providing timely and useful education on *how to introduce entrepreneurial aspects alongside the core expectations of the computer science program*. Designing, implementing and operating environments for start-up education needs to be taken seriously, as poorly executed education can cause students to distrust themselves, the teacher, and the learning environment.

University courses do not often provide students with the chance to see the link between their actions and real-life outcomes, although it would be possible [2]. Many of the important relationships and effects in software engineering are learnt best when students gain personal experience of the practical application of the methodologies. Therefore, teaching start-up-related knowledge and skills requires an environment where students can experience the consequences of their actions. This both gives meaning to students' experiences and solidifies their prior knowledge, creating a fertile ground for posing questions that motivate further learning.

Since 2010, the "Software Factory" courses at the University of Helsinki [3, 4] have provided students with opportunities to experience software development in a start-up-like environment. Here, teams of Master's-level students use contemporary tools and processes to deliver working software prototypes in close collaboration with practitioners. The goal of the learning environment is to allow students to apply their advanced skills in an environment with working life relevance and to deliver meaningful results for their customers [4]. Software Factory projects face extreme constraints on schedule and resources, along with high ambitions on the innovativeness and attractiveness of the end product. They offer potential for very high pay-off in the event of success through, e.g., increased chances of future employment due to demonstrated abilities. The Software Factory environment offers a safe learning experience, as practical consequences of failure are limited, compared to being employed by an actual start-up. Some of the chaotic traits that many real start-ups have are not present in the educational setting, allowing students to focus mostly on the questions and uncertainties of software product development.

In keeping with the idea of reflective practice (e.g., [5, 6]), we look back at the past seven years of Software Factory projects and extract insights related to start-up education. The material is presented as a collection of patterns and anti-patterns for five purposes: designing, implementing, and maintaining i) physical and virtual environments, ii) course design and curricula, iii) learning materials, iv) teacher guidance and v) educational interventions for start-up education. This article extends a previous paper that concerned only the patterns [7].

The remainder of this paper is structured as follows. In Section 2, we discuss entrepreneurship and pedagogical theory. Section 3 describes our research context and approach. The main result, a collection of educational patterns and anti-patterns, is given in sections 4 and 5. We discuss the result in relation to theory and a framework for entrepreneurial education in Section 6, and conclude the paper in Section 7.

2. Background

Entrepreneurship combines "the mindset and process to create and develop economic activity by blending risk-taking, creativity and innovation with sound management, within a new or an existing organisation" [8]. In an era where the formerly dominating large firms are restructuring, downsizing and creating new strategies for growth and survival, fostering positive attitudes and entrepreneurial skills are key in creating new jobs and a society that values entrepreneurship and innovation [8].

To prepare for this, the European Union's Joint Research Centre's "Entrepreneurship Competence Framework" defines 15 competences that should be integrated into the educational fabric [9]. These include both motivation and team-work skills, but also several personal traits such as creativity, self-awareness, self-efficacy, and perseverance. Strengthening these in software engineering education is not straightforward and requires integrating new pedagogical approaches with knowledge from entrepreneurial research into the core subject areas.

2.1. Start-up education for software engineers

Previous research suggests that entrepreneurs are a very heterogeneous group [10] and thus new pedagogical approaches that allow both extremely creative as well as less creative individuals to thrive are called for [11]. Flexible educational structures that can cater to both group-level and individual needs are suggested [12]. While management and business research suggests general principles for entrepreneurship and start-up education, few papers have investigated the topic within the scope of software engineering. Here, we briefly outline the most important findings.

Entrepreneurship education should focus on developing personal attributes and skills as well as tasks [13]. Concrete experience through active participation should be offered, e.g. by project work. The literature clearly indicates that entrepreneurship can be taught, and that teaching methods can be enhanced through active participation. There is evidence to support the notion that educational programs can positively influence entrepreneurial attributes and that they can build awareness of entrepreneurship as a career option and encourage favourable attitudes towards entrepreneurship. However, Gorman [13] notes that there is strong evidence that small business owners and managers resist start-up training.

Inducing learning by involving students in start-up firms, e.g. as interns, may at first appear to be a good solution, but may not be beneficial in all cases. There are indications that start-ups with immature processes may be detrimental to undergraduate students' understanding of the software development process and its relationship to high-quality software products [14]. This may be worsened by the aforementioned resistance to training among entrepreneurs. It may be more beneficial to have start-up education occur in an environment where the host university has more control of the pedagogical content and quality.

Several universities have established project-based courses for teaching software engineering, with capstone courses being a typical educational pattern. There are a few recent examples of such courses with special focus on entrepreneurship and start-ups. Järvi et al. [15] report on experiences with organizing a course on Lean Start-up, focusing on ideation, innovation, and subsequent product and business development. They describe a course design that supports experiential teaching of product and service development using the Lean Start-up approach, and find that the course is promising for teaching software business and intrepreneurship skills to both software engineering and business students. Harms [16] reports on results from a B.Sc.-level Lean Start-up project. In that report, self-regulated learning was positively related to individual-level assessment, and team-based learning and psychological safety were positively related to group-level assessment. In other words, there were indications that self-regulated learning is favourable for individual students, and that peer collaboration and a safe peer group are favourable for learning as viewed on the group level.

2.2. Personal traits and motivation in education

Self-efficacy relates to the individual's emotional evaluation of their own worth [17, 18]. The belief that one's own actions have an influence is crucial for any learner, and also affects individual response to stressful situations [19]. Therefore, in flexible learning environments, the motivation of students is highly influenced by the degree of freedom to choose what to work on and with whom [20]. Self-efficacy influences choices when facing challenges: students with high self-efficacy tend to perform better than those with lower self-efficacy [21, 22, 23]. Consequently, self-efficacy contributes to whether a person can face the challenges given [17, 18]. It plays a role in whether a student believes that they can learn, and in whether they will invest themselves into learning [21, 24].

Luckily, self-efficacy can be improved through training [21]. One approach that has been shown to improve self-efficacy is behavioural modelling, which refers to a process where students are given a model of the process that is required to perform a given task [25]. This approach is prominent in many learning theories. Cognitive apprenticeship, the theory of how a master teaches a skill to an apprentice, suggests that modelling would be used as the first step when learning a new task [26, 27]. Cognitive apprenticeship outlines multiple teaching methods that are relevant to learning complex tasks. Here, modelling is used to provide the apprentice with an overview of the problem solving process. This is followed by coaching and scaffolding, a process where a teacher provides feedback and support, employing meaningful strategies and activities to support further learning. Once a student engages in learning, the support from the teacher is slowly faded and the teacher engages with new learners.

In general, the teacher acts as a guide, facilitating the development of expertise, instead of "handing out knowledge" [26, 27, 28]. This emphasis on the students' effort is echoed in situated cognition theory which is intertwined with the cognitive apprenticeship theory. Situated cognition theory posits that while knowledge is constructed by the student, it is always linked to the activity, context, and culture that the learning environment is surrounded by [29]. In general, it is suggested that learning occurs through social interaction and shared language [29]. While students may at first work on tasks individually, it is crucial that interaction with the community and a shared language is formed. Approaches such as project-based learning and problem-based learning can be used to adjust the level of exploration and to provide team-based experiences where students can articulate and reflect on their decisions [30].

3. Research approach

This article aims to answer the research question: *"What are the ingredients of successful software start-up instruction in higher education?"*.
To address this question, we extract potentially reusable patterns and anti-patterns for project-based software start-up education from our experiences with arranging both B.Sc. and M.Sc. level courses.

3.1. *Context of the study*

Higher-education institutions may lack the incentives to keep up with the fast-paced developments of the IT industry. This may lead to using outdated technologies and development practices no longer used by practitioners [31]. After recognizing this threat in 2009, our department started an ongoing reform of educational goals for the B.Sc. degree. The goal was set so that "any graduating bachelor from our department should be able to perform efficiently in a modern software development context". At the start of the reform, we interviewed our alumni, discovering the following problems in our education of the time:

- Programming skills of the graduates were not at an adequate level and good programming practices were barely known.
- Students lacked knowledge of software architectures and needed more experience in building web applications.
- The students' knowledge of modern software processes and agile practices was limited.
- Students lacked knowledge about quality assurance methods: behaviour- and test-driven development, continuous integration, and continuous deployment.
- Students lacked administration and maintenance skills.

The reform started with the most fundamental problem: the poor skills in programming. Our lecture-based "Introduction to programming" course was redesigned to create an active role for the students and to position the teacher only as an enabler of their learning. We used the cognitive apprenticeship theory [26, 27] to establish our foundational pedagogical approach, re-designing our courses to incorporate best practises, along with programming-related methods and activities. Subsequently, this approach was scaled up to the masses by empowering more experienced students to teach their peers [32]. The cognitive apprenticeship cycle was incorporated into multiple levels of instruction from introduction of elementary programming concepts to the highest level of the curriculum, giving the ultimate goal for instructors to "fade for good".

In parallel with the fundamental reform of our B.Sc. level education, we developed a spearheading course on the M.Sc. level. The first such "Software Factory" project course in 2010 was designed as a response to the need for a modern environment that would link software engineering education, research, and entrepreneurship in a university setting [3]. The course was offered as a short and intensive experiential project that required presence equivalent of full-time employment. Students would focus on the project for one period (half a term, i.e., roughly two months), yielding credit points equivalent to the amount of work performed. During this time, students could choose to take other courses simultaneously, usually only one. However, by opting for a shorter work week, they could manage additional studies, according to their preferences. By keeping the intensive project short, the impact on student workload across the academic year was kept reasonable. Still, the course was designed to be challenging and to require students to manage their personal schedule well.

At the time, a major concern for continuing our curriculum reform was the fact that most university teachers lacked recent industrial experience or direct contact with practitioners. Thus, awareness of emerging software engineering practises was lacking and teachers were not capable of conveying the methodologies in their teaching [31]. We were forced to employ an unusual strategy for a research university. The tenured associate professor that was driving the reform participated in a Software Factory project, assuming the role of a normal student and engaging in a real, industry-relevant project. Later, this faculty member spent a year-long sabbatical immersing himself in professional start-up software development. This experience was key in redesigning our curriculum and driving the idea of entrepreneurial education further [31].

As the software engineering skills of our B.Sc. level students have improved, the pre-conditions and expectations for the Software Factory course have also changed throughout the years although its core ideas have remained the same. These developments and related experiences have given us confidence to present emergent patterns, best practises and issues to avoid when arranging entrepreneurial education in the context of software engineering education.

3.2. Methodology

Design-based research [33, 34] is a dual-purpose methodology that studies education in its authentic context. It aims at developing both the design and the context of the education with an iterative process that begins by an initial problem analysis, followed by cycles of evaluation and improvement that deepen understanding [35, 36, 37]. Design-based research relates to educational action research [38] and design science, linking paradigms in use in the information technology field at large (c.f. [39]). Its emphasis, however, is on educational improvement [38].

Our research started in 2009 by building and implementing an initial design, followed by the first Software Factory course in 2010. The initial design comprised everything from the physical facility and visual materials to project management and course plans, and these have subsequently evolved. Each succeeding project can be seen as a single case in a multiple-case study, providing triangulation of viewpoints, data and researchers [40]. By reflective practice (c.f. [5, 6]), we abstract our experiences into education design patterns and anti-patterns that help teachers to implement similar learning environments.

3.3. Data collection and analysis

We have collected a longitudinal record of documents and observations that span from the initiation of the Software Factory, and the projects that followed, to this day. Table 1 outlines the projects that form the data set for this study, representing various types of start-ups and start-up-like environments. Our related research data comprises of:

1) Design documents for the facility and its work processes.

2) Software assets, their documentation, and project documents – these are created by student teams and their customers.

Table 1: Projects in the longitudinal data set.
Each project is one period long (7-8 weeks) unless otherwise indicated. Data is from January 2010 to May 2017.

Year	Projects	Students	Types of partners
2010	5	45	a) A start-up simulation, b) a project with a small start-up, c) early product development from a student's idea.
2011	4	33	a) Supporting an internal start-up, b) a 2-part project with a small start-up and c) one with an established company.
2012	5	26	a) An internal start-up, b) a 2-part project with a small start-up c) a collaboration with 2 established companies.
2013	2	15	a) 2 projects for participating in open source software development projects.
2014	3	21	a) Participating in open source projects, b) a 2-part prototype project for a small start-up.
2015	3	22	a) Participating in open source projects b) a 2-part project for working on an open source spin-off.
2016	3	10	a) 2 continuing projects on the spin-off software, b) a research driven prototyping project.
2017	1	6	a) Research-driven product prototyping.
Total	26	178	

3) Meeting memos and retrospective documents produced by the students and their customers during the execution of the projects.

4) Group interview data from post-project debriefing sessions – these provide a timeline of actions for each project. All participating students and at least one customer representative have participated in these sessions, with a small number of exceptions due to absences for personal reasons.

5) Individual interviews with students and customers. All customers have been interviewed at least once during their projects. Approximately 80% of students have been individually interviewed, with interviews occurring during or after the project. Some of the interviews have been conducted in conjunction with other research, as reported elsewhere (e.g. [4, 41, 42, 43]).

6) Anonymous feedback from students.

7) Personal notes of the course staff.

The present authors have a variety of viewpoints on the course of actions of the Software Factory. The first author coordinated the design and operation of the Software Factory throughout its existence. Two authors have coached the student teams and one has participated in projects as an observing researcher. In addition to their role as teaching and research staff, some authors have participated in the Software Factory projects as students. From these perspectives, we have gradually and iteratively synthesized the set of patterns and anti-patterns that are given in the following two sections.

4. Patterns for Software Start-up Education

This section provides patterns for software start-up education. For each pattern, we provide contextual information and illustrative examples that should help applying and adapting the patterns to similar courses.

4.1. The physical and virtual environment

Simulating a software start-up requires attention to creating an authentic learning environment. To accomplish this, we constantly follow the evolving state of the art of software development tools and environments and keep the Software Factory infrastructure up to date accordingly.

Pattern 1: Team Room

- Have a team room.
- Equip it properly for start-up education.

A well equipped office room that exceeds the standard level of classrooms resembles a real workplace and increase the students' ambition. We have put emphasis on a functional, aesthetic interior design and providing students with modern equipment and ample wall space with whiteboards. This allows informal and transparent communication and co-creation. The room should also be furnished to encourage relaxing, taking breaks, and communicating casually.

Pattern 2: Studying is like work

- Create realism by simulating working life.
- Require working hours according to local conventions.

We give students the freedom to choose their working hours, but expect them to work an average of 6 hours per working day. They can choose between a full 5-day working week and a shorter 4-day working week. In special cases, an even shorter working week can be arranged if this fits with the project. The students should spend time together and be present in all meetings. We encourage students to work between 9 and 17. Supporting a regular schedule, with suitable flexiblity for individual preferences, tends to lead to increased productivity, a safe working environment, and an atmosphere for increased creativity.

We maintain norms and standards related to working life practices and skills. Examples of these range from relatively mundane to more intricate. As an example, we encourage clear and effective communication in both speech and writing. Students should inform others of being late or away on sick leave. They are expected to keep their promises and give notice if they are unable to achieve what has been agreed upon. Students are made aware that these aspects are vital both for team climate and for managing the project. Being pro-active and maintaining a professional attitude towards others are recommended. Good concentration on tasks at hand is indicated to increase productivity, creativity and to create a safe working environment. Finally, co-operation and asking for help are recognized as behaviours that can be benefited from in both the classroom and in work life.

Pattern 3: The virtual Development Environment

- Provide a common base infrastructure for all projects.
- Upgrade continuously and stay with latest versions.
- Choose technology pragmatically and based on evidence.

A common, up-to-date development infrastructure helps students to start projects more smoothly. With this provided, students are required to decide on their own project's communication and collaboration tools. While each project has particular needs with respect to its software development tools, students are expected to build and maintain their project's own tool-set as a part of their learning process. This typically entails integrating version control, automatic build and test suites, and deployment automation, but may vary greatly between projects.

A base infrastructure is also beneficial for the project itself. Using personal laptops, other equipment, or various incompatible tools, such as different code editors or personal git work flows, often lead to compatibility issues at some point. Solving these provides an excellent learning experience, yet is not productive at all.

4.2. Course design and role in the curriculum

Experiential, project-based learning in an open-ended environment brings about several challenges with respect to course design. The first concern is the placement of the course in the curriculum.

With respect to this, we have employed a flexible solution where learning goals are customized according to the students' current stage of studies. Early-stage students receive more mentoring and are motivated to choose their further studies based on their experiences. Later-stage students are directed towards gaining closure for their studies and finding inspiration for their M.Sc. thesis. We have previously reported on trials with assessment strategies [44]. The fit to the curriculum can be ensured through careful selection of the projects and customers.

Pattern 4: No Teacher, Only Learners

- Teachers should encourage self-directed learning.
- Think of teachers as participants in the start-up.
- Teachers should set their own learning goals.
- Guidance is needed to avoid detrimental effects.

In the beginning of each project, the problem and solution spaces are largely unknown, and no-one can tell students exactly what to do. Here, teachers should be posited as co-learners while more knowledge about the project's goals emerges. Teachers can bring in knowledge on software engineering practices, project management, and general approaches that support problem and solution space exploration. The modelling-scaffolding-fading cycle should be performed rapidly in several areas at once, and responsibility of running the project and meeting its goals should be transferred to students as soon as possible.

The teacher should initiate communication with the customer, ensuring that students set up communications tools and project tracking through, e.g., a Kanban board. From there, students should be responsible for running the project and setting their first goals. Teachers can use learnings from previous projects to help students overcome obstacles quicker. Coaching should help navigating the inherently uncertain and ill-defined start-up environment and emphasise deliberate learning as a value. For this, coaches can formulate and discuss problem statements, aid in gathering evidence for decisions and help evaluate the consequences of decisions.

Pattern 5: Continuous Formative Assessment

- Formative assessment provides feedback to improve the ability to execute the project.
- Everyone continuously assesses everyone.

Increasing self-efficacy of the students and helping them build their professional identity as software developers should be focal for startup education. Also, there are several non-technical skills that should be developed to help navigate the problem and solution spaces of the project. These include teamwork, the ability to lead and follow, communicating, self-awareness, perseverance and showing initiative. Linking assessment to such learning goals is difficult, but can be accomplished through reflective assessment for and by all project participants. We encourage such assessment not only as part of formal meetings, but also during everyday activities. A culture of constructive feedback is built by teachers by example.

Sprint retrospectives can play an important role in formative assessment. They provide regular opportunities for feedback. Students need to be helped to notice where they have been successful, as they often are stuck with incomplete artefacts or negative experiences. In cases of a failed sprint, it is helpful to have a moment to discuss it and reach closure. Then, the focus should be shifted back to learning – what would students now do differently – and then start a new sprint with a renewed enthusiasism.

Pattern 6: 360-degree Summative Assessment

- Include all project stakeholders in assessment.
- Assess learning goals through observable behaviour.
- Provide opportunities for expressing personal strengths and accomplishments.

In our summative assessment (c.f. [44]), we emphasise a 360-degree self-, peer-, and observer (teacher) perspective. Teaching staff often do not have a complete picture of the performance of each individual student, but multiple views can increase assessment accuracy. The assessment should focus on observable behaviour that indicates fulfilment of the learning goals. Here, self-assessment can help students to express their strengths and accomplishments.

In our Software Factory course, the learning goals are set to be general enough to apply to all projects, but to allow for them to be tailored to each project. Although it would be possible to focus on group assessment, we have chosen to focus on the individual student, as our aims are to advance their learning as individuals situated in a group. The learning goals are divided into a set of factors that begin with basic behaviours and progress towards higher levels of involvement and skill [44]. The basic factors include presence and activity, implying that students take part in and actively influence project activities. At a slightly higher level are an attitude of taking the initiative and committing to perform planned tasks. At the highest level are actual contributions which impact the project, in the form of code, documentation, or other deliverables, or in the form of project management, customer communication, or support tasks. These are assessed primarily through analysis of artefacts. Also at the highest level is how the person performed in their expert role and how they collaborated. These can be observed through each student's social behaviour in the team. Concretely, the factors are assessed by collecting ratings on behavioural descriptions and written statements from each project stakeholder.

In summary, we place emphasis on what the student does and how, but also on the results each of them produced. However, we place less emphasis on how the results are utilised outside the project. For example, the start-up partner may fail after the project, which we see as being largely outside the students' sphere of influence. It is thus not a focus of our assessment. We also note that this type of assessment is an an area for further research, and we continue to evolve our methods.

4.3. Learning materials

Learning materials need to be reconsidered for start-up-like learning environments. Their choice and creation is dependent on students' prior knowledge of topics arising in each project.

Pattern 7: The World is the Learning Material

- Learning materials should be discovered and created on demand.
- The coach has a key role in highlighting and critiquing potential learning materials.
- Maintain a basic set of learning materials common to all projects. Use student-created learning materials in future projects.
- Create a culture by collective material maintenance.

A majority of the learning materials is created during the project, which emphasises the knowledge acquisition skills of the students. They should also learn to assess the materials and manage knowledge repositories as needed. Here, the coaches can help students by highlighting and critiquing materials. A common base set of materials provides efficiency and consistency in projects.

Student-created materials from previous projects, such as learning diaries, presentations and notes from retrospectives and debriefing sessions, can aid subsequent projects. This is crucial especially if the same start-up company returns for a new project, but such materials can be beneficial for unrelated projects as well. A tradition has emerged where students prepare tips and cautions ("do's and dont's") to be delivered to students in future projects. We have found this to be useful and fun: students enjoy writing tips for the newcomers, perhaps because they leave a legacy behind. Also, we have observed new students receiving the materials with a slight sense of respect; it is as if they perceive a personal connection to the trials and tribulations experienced by their predecessors. Such materials can become part of a student-to-student culture in the learning environment even though they may never meet in person.

4.4. Teacher guidance

As previously mentioned, our pedagogical approach is that the teacher and teaching assistant are facilitators for building expertise. The patterns in this section concern how teachers can be guided to contribute to the learning environment.

Pattern 8: Role-play the Start-up

- Roles of teachers are defined by the start-up context.
- Realism in role naming promotes the role function.

As teachers are participants in the start-up (see Pattern 4), it is also beneficial to consider what their role entails. The *coordinator* role corresponds to a senior executive, e.g. a CEO or COO of a small company proving software development services to customers. The coordinator manages projects on the portfolio level, handles project selection and initiation, and leads the communication with the customer. The coordinator role would usually be filled by a senior teacher.

Coaches work with students on a regular basis and correspond to roles with the same name in companies. They are mediators between the student teams and the customer and thus assist the coordinator in customer communication. The most important task for the coaches, however, is to guide the students in executing the project. Coaches usually are teaching assistants. *Mentoring* entails reflection and takes a longer and sometimes more personal perspective than the coaching. Thus, mentor is a crucial role for promoting learning.

Pattern 9: An Exit From the Simulation

- Teaching staff must sometimes intervene to ensure that the project can proceed.

Situations sometimes arise that would either not occur in a real start-up or that would require more time to solve in a real situation than what is available in the learning environment. The teacher must step forward to handle such situations and to override project priorities. For instance, students may have to withdraw from the project temporarily or permanently due to personal circumstances. The roles suggested in the previous pattern must then be set aside – but often, it is possible to use the seniority of the coordinator role to provide a positive way to "exit the simulation".

An example of a situation where this pattern can be activated is when a student does not contribute to the project despite repeated attempts at bringing them into the team. It may then be necessary to interrupt their involvement in the project in order to find out what the actual problem is. In some cases, students have overcommitted themselves and have already realised that they need to focus on other courses. In these cases, providing an amicable way for them to drop the course can be the best option.

Pattern 10: The Reflective Teacher Team

- Establish a community of teaching practice.

- Teaching staff must coordinate behind the scenes.

- Teaching staff need a safe environment of their own.

Whereas students learn by being embedded into their project, the continuous cycle of improvement requires that teachers maintain a larger perspective. In order to achieve learning in the teaching organisation, it is important to establish a community of teaching practice. In addition, the intensive nature of the course means that teachers also need a safe environment of their own, where they can vent feelings and reflect on their own efforts to facilitate teaching and improve the learning environment.

In the Software Factory, teaching staff meet during and between projects as needed to discuss both ongoing projects and future ones. There is often a need for teachers to coordinate behind the scenes in their own environment while projects are ongoing, in order to validate observations as well as to prepare for and carry out project-level interventions. A reflective teacher team for this kind of learning environment may, in the best case, propagate insights to other courses, and we have seen examples of such transfers as discussed in Section 3.1.

4.5. Educational interventions

A set of project-level interventions have been developed to address some recurring situations. These are structural elements on the syllabus level that can be repeated from one course instance to the next. Some of the interventions are used in all projects, while others can be activated when needed. The interventions are not meant to be mechanically followed, and we suggest that they, too, need to be learnt by teachers in the same manner as students learn: through activities in the authentic context guided by more experienced teachers. We have discussed some of the interventions in a previous publication [4].

Pattern 11: The Project Blueprint

- Develop and maintain a blueprint to give structure to projects and enable their enactment with minimal effort.
- Address at least project and student selection as well as project start and end activities.

- Customise the blueprint for each project as needed.

The Software Factory project phases are depicted in Figure 1. The process starts with project selection (1) followed by a student selection phase (2). After the selection, a project kick-off event (3) is organized, starting the agile software development project with all parties present (4). A final demo (5) is typically held on the last day of the project, followed by the project debriefing session (6).

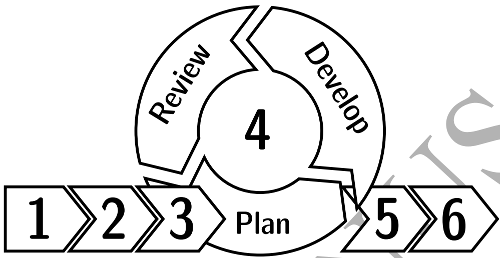

Figure 1: Software Factory Project Blueprint. 1: project selection, 2: student selection, 3: kick-off, 4: project execution, 5: end demo, 6: debriefing.

Pattern 12: Student Selection As Job Interviews

- Adapt start-up job interview elements to student selection.
- Even those who are not selected can learn something from the feedback.

Unlike most other courses in our department, the Software Factory applies a student selection process before admission. The process mimics some elements of the hiring process we have observed in software companies: hands-on skill demonstrations and an element of character assessment. We ask students to implement a programming task and give a self-assessment of their skills. The purpose is to assess students' ability to perform a basic task promptly and correctly and to be able to make a first formulation of their identity as a software developer. The programming task is tailored to each project and students are usually free to use any programming language. The difficulty level of the selection is set to include all students who are likely to be able to participate meaningfully in the course. It also aims to provide feedback to students on how to develop their knowledge and skills to that basic level. Additional interviews can be conducted depending on project-specific requirements. The selection is done collaboratively by the coordinator and the coaches based on a ranking scheme. However, selection is not based on a quota – all students who pass the selection are admitted.

Pattern 13: Kick-off!

- Focus on igniting students' entrepreneurial spirit and starting team building.
- The kick-off session is an early and easy opportunity to enhance students' self-efficacy beliefs.

Students often meet each other, the coaches, and the customer for the first time in the kick-off meeting. This event is crucial as the onboarding requires a large amount of information to be assimilated, including introduction of each participant and the industry project as well as performing formal tasks such as signing NDAs. In order not to overwhelm students, the kick-off has to be carefully planned. We found it particularly useful to organize a pre-meeting with students to focus on team building before meeting the customer. We have also combined team-building with a dry run of the development process: using a Kanban approach to define sensibly-sized tasks for carrying out a pizza order, necessitating steps such as learning team members' names, determining tasks, and executing them, giving the students a feeling that they can master collaboration.

Pattern 14: Start the Process Engine

- Transfer the responsibility of enacting the development process to students.
- Keep repeating the basic process until it becomes routine.

The first two weeks of the project are critical for transferring the enactment of the process, grooming students into their project roles and transferring project ownership to students. Students are encouraged by the coaches and the coordinator to see the project as their own. Students must be active themselves in identifying tasks that would create customer value. They must also assimilate a large amount of material. To frame this activity, it is advisable to create time-boxed tasks. In our experience, guiding students to work in pairs has been an efficient way of overcoming disbelief in personal skills, and also acts to balance differences in skill levels. We also observed that rotating the responsibilities among students gives them a chance to step out of their own comfort zone and often results in self-realization of their personal strengths and weaknesses. Noticing this and giving credit to students can positively impact their self-efficacy. Moments of success nourish a positive spirit within the team.

In practice, unexpected situations occur that threaten to disrupt the flow of the project. Changes in participants have occurred in the early weeks of several of our projects, when students re-prioritise their commitments or register late for courses. This needs to be managed carefully and possible newcomers need special attention to be integrated well into the team.

Pattern 15: Let Them Talk

- Put effort into connecting the students and the customer.
- Continue to nurture the communication and promote the idea of a single team.

Building effective communication between the customer and student team is essential for project success. Students are often hesitant to contact the customer, and coaches should encourage them. Contrary to what students might think, customers often appreciate being asked questions which lead to "clever discussions", as one of the customers put it. More urgent issues have been handled via phone or video calls. As complementary communication channels, students use messaging services, phone calls, voice over IP, e-mails, file-sharing services, electronic task boards as well as physical kanban boards. The main responsibility of the communication is given to the students already during the kick-off event (see Pattern 13), and the role of coaches is to act as enablers when needed. We have tried to minimize coaches' interventions when the customer is present to avoid bypassing the students in communication. Sometimes, however, coaches have intervened to help to focus the discussion.

Apart from daily communication, we have encouraged weekly customer meetings where students present their progress, typically through a demo. To make the occasion slightly more formal, the coordinator has also been present. Weekly demos might not be prepared perfectly by students in the beginning. They might fail, and often last-minute changes in functionality have to be implemented. However, the demos tend to improve throughout the project, and experiencing the failures makes it obvious to students that they need to think and plan ahead. It is useful to rehearse the demo internally before the final demo session with the customer, and coaches can help here.

Pattern 16: Closure: The Story of Our Project

- Hold a debriefing session after the project has ended.
- Use a participatory method to build a collective story of the project.
- Leave students with a feeling of accomplishment.

After each project, we organise a debriefing session for all project participants. Placing the debriefing session after the end of the project helps delineate and separate it from the project, giving students some distance to the project. Debriefings involve a retrospective look at the project both from the customer and team points of view. First, we solicit the often diverging memories of how the project unfolded, then process different perspectives and underlying reasons, and arrive at a collectively accepted version of "our project story". The aim is not to arrive at objective fact, but rather to increase understanding of why events occurred and decisions were made in a certain way. A successful debriefing session leaves participants, and particularly students, with some degree of crystallised learning – insights into their own actions and their consequences for the project in a form that applies elsewhere – and a heightened, healthy sense of self-efficacy

ACCEPTED MANUSCRIPT

stemming from the experience of a completed project. From the perspective of the teaching staff, the debriefing session also provides information that can be used in summative assessment (see Pattern 6).

5. Anti-patterns for Software Start-up Education

This section provides anti-patterns for software start-up education. As with the patterns in the previous section, we provide contextual information and illustrative examples that should help applying and adapting the patterns to similar courses. We also mention related patterns and anti-patterns. The related patterns are possible solutions that could be applied to avoid the anti-pattern or reduce its effect. The related anti-patterns are manifestations of problems with similar causes or effects. We do not have solutions for all the anti-patterns, but we believe that being aware of them may be a step towards avoiding them.

5.1. The physical and virtual environment

An experiential course of the kind we have described cannot thrive unless the learning environment supports immersion. The anti-patterns in this section alert educators to some of the threats against an authentic learning environment.

Anti-pattern 1: The pointless corner

- Allocating an inadequate space from leftovers.
- Failing to equip the team room properly.
- Related: Team Room

The pointless corner is a team room that no-one wants to visit. A table and two chairs do not make a team room unless they stand out. An authentic team room is created by allocating valuable space to it. This demonstrates that the university is serious about supporting its students in their education and its external partners in their collaboration. That in turn establishes the level of ambition for education and learning.

Valuable space and equipment is a subjective concept. A well-functioning team room does not need to be expensive. If it is not possible to allocate dedicated space to a team room from university premises, it does not need to be a show stopper for the whole project. Other alternatives require organizing efforts and flexibility from the teaching staff, customer and students. According to our experiences, the office space at customer premises can also be an effective learning environment. This requires that not only students, but also the teaching staff is able to access the location in a reasonable way to ensure they can provide guidance. A "garage approach" where students, teachers, and customers agree on where to meet for each project should be considered as a last alternative. It often imposes high demands to students self-regulation skills and limits the complexity of the development environment. Avoid creating a space which students and project stakeholders do not feel comfortable in or that restricts the project scope.

Anti-pattern 2: Rules for the sake of rules

- Simulating the wrong kind of working life.
- Treating rules as an end in themselves.
- Related: Studying is like work

The purpose of imposing rules, such as working hours and required methods and practies, is to create realism and simulate working life. Unfortunately, rules can become detached from their educational purpose and start living their own life. Rules that contribute to realism are motivated because they further learning goals – the ability to improve one's own work environment and to experience the link between the goals and aims of the project and the means of reaching them.

Rules must not be mindlessly chosen but must be motivated by their existence in a relevant working life context where they serve a function. If the purpose is to simulate a start-up in a specific domain, the conventions of similar companies in that domain should be followed. Each rule and practice should be motivated by a project need rather than being an end in itself. Rules, methods, and practices should be chosen and adapted because they contribute to the project's goals.

Anti-pattern 3: The decaying development environment

- Getting stuck in "approved" tools which are no longer up to date.
- Failing to keep support systems in shape.
- Related: The virtual development environment; The dream development environment

We have observed a tendency to try to stick with a certain set of default tools and development frameworks for each project. For some time, we tried to ensure that projects used the same tools and frameworks, but we soon realised that this is a fool's errand. The fallacy may stem from the otherwise sound idea of reusing technical knowledge and optimising project setup, but it fails because in this environment, each project is supposed to start from scratch, teachers do not actually retain project knowledge, and modern tools have all but eliminated time-consuming setup. Trying to maintain a standard set of tools and frameworks puts unnecessary strain on teachers and university IT, slows down projects, and means the technical infrastructure is not synchronised to what students have skills in using.

Simultaneously, certain systems are necessary to provide in a somewhat standardised manner. For example, Continuous Integration systems are complex to set up and a ready-made system can let projects start from day one using proper testing and deployment techniques. This applies especially if the systems need hardware components such as dedicated mobile devices for realistic automated testing. However, these few crucial systems must be kept up to date. Setting them up once and thinking that they will work is a different fallacy that can lead to a decaying development environment.

Anti-pattern 4: The dream development environment

- Getting stuck in tool selection and evaluation.
- Being unable to resist trying new tools for the sake of novelty.

- Related: The virtual development environment; The decaying development environment

Technology selection and evaluation is an important task that can have long-lasting effects for a company. However, the goal of start-ups and start-up education is not to build the final system. Learning is the most important goal. For start-ups, the goal is to learn what the market and customers need and how to provide a product or service that fulfils that need. For start-up education, the goal is thus to provide skills that contribute to such start-up goals. We have experienced a few projects where significant time was invested on exploring technology options, ultimately leading to a set of many options but lack of criteria that would allow for a choice. Establishing those criteria first with a reasonable hypothesis regarding the technology has, in other projects, proven more fruitful.

Sometimes, both students and customers have had a strong desire to try a new tool or development framework that supposedly fixes shortcomings that older alternatives have. However, unless the project goal is specifically to evaluate those new tools and frameworks, it may be better to stick with something that is not on the very bleeding edge. Typically, halfway through the project, the shortcomings of the new tool become apparent, and it may be difficult to find the support to resolve problems with it. Rather than striving for the dream development environment, we seek to be pragmatic and use tools and frameworks that students are comfortable with, and avoid the very latest ones until they have matured a bit.

5.2. *Course design and role in the curriculum*

It is a challenge to integrate experiential, project-based learning in a curriculum which is otherwise organised more traditionally. We consider here particularly the risks involved with such a combination

Anti-pattern 5: The disconnected course

- Failing to establish connections between experiential project courses and other courses.
- Avoiding to adjust the curriculum to support the experiential project course.

Having a course that is disconnected from the rest of the curriculum means that there is a lack of awareness in curriculum planning. No matter how exciting an experiential, project-based course is for those involved, it does not contribute to a long-term learning arc for students unless it is supported by the rest of the curriculum and unless it feeds into a more advanced level of studies. Although the connection can be dynamic and the course can support learning goals at different levels, establishing connections to other courses requires deliberate effort. It is easy to forget that such connections require not only consideration in course plans, but also that teachers of other courses are aware of what happens in the project course. Also, not all prerequisites can be provided or make sense to provide on a continuous basis. For example, specific technical knowledge of a development framework may be too narrow to warrant recurring courses. If they are scheduled as regular courses, they are never timely with respect to the fast-moving experiential course. However, such courses can be provided as pop-up courses that serve changing needs. We believe keeping the curriculum cohesive requires continuous effort, and this is an area where we are still learning.

5.3. Learning materials

It may be hard to shed old conventions regarding learning materials. Here, we consider what may happen when trying to combine a traditional view of learning materials with the dynamic requirements for start-up education.

Anti-pattern 6: The sacred textbook

- Requiring students to read a textbook because every course has to have one.

Textbooks justify arranging courses on a subject. If there is a textbook, there is one argument less against the topic being suitable for higher education. But in experiential, project-based learning, the role of textbooks is the same as any other learning material. Still, it may be tempting to bring in a textbook as required reading just because "every course has to have one".

Textbook material should be covered before this kind of course starts. Some knowledge is better learned outside project courses, yet that knowledge can be reconstructed by learners in experiential projects, giving it more meaning and giving them new insights. Establishing the connection between what happens in the project and what prior courses have taught is something the teacher can help with. This is not to say that textbooks or other kinds of written materials cannot be learning materials and in fact, we have established a small library of books in our Software Factory premises. The initiative to use the books, however, should be derived from the students themselves: from learning needs that arise durig the course of the project.

5.4. Teacher guidance

Experiential learning projects require teachers to balance involvement with other concerns. These anti-patterns could help self-reflective teachers improve as coordinators of active learning. They could also help in discussing the role of the teacher among colleagues.

Anti-pattern 7: The bureaucrat-teacher

- Teachers being disconnected from the project.
- Assuming a role of only managing the course paperwork.
- Related: Role-play the start-up; The reflective teacher team

Teachers must be able to adapt to the style of learning in an experiential course. However, teachers can become disconnected from the project and fail to gain an understanding of what is really going on. If they are not connected to the project and its stakeholders, and take the role of mere administrators or bureaucrats, the educational benefits will be lost. Teachers will not be able to drive the educational aspects of the project, as they will not have the required insight to connect educational interventions to project events and goals. Also, the educational priorities sometimes include making choices for the sake of learning rather than for the sake of the customer, but such choices require insight into the project dynamics. When failing to connect with the project, teachers will not be able to do meaningful formative and summative assessment, and they will not understand the project aims.

Anti-pattern 8: The takeover-teacher

- Teachers taking over the project.
- Becoming too engaged in the project goals.
- Related: Role-play the start-up; The vanity project; The reflective teacher team

If teachers become too engaged in the goals of the project, they may start to perform project tasks and "take over" the project from students. This can occur in particular when the teacher believes that the project is at risk to fail in some sense. For example, they may believe that students are bypassing the process when in fact they either would require greater insight into how it should be connected to the real project tasks, or when the process is simply not suited to the project. Teachers may then be tempted to "rescue" the project – but this is not a good idea from a pedagogical perspective, since learning opportunities are missed when focusing only on surface implementation of predefined goals. This pattern is common if the project is a vanity project.

5.5. Educational interventions

Not all structures and interventions which intend to advance educational priorities work well. Similarly, lack of critical interventions and checks can lead to suboptimal learning and failed projects. Some of these patterns concern customers, but we emphasise that they are also ultimately the teacher's responsibility, as customers must also be trained and their expectations managed.

Anti-pattern 9: The vanity project

- Projects that are selected based on attractiveness alone.
- Too much politics in the project selection.

It is sometimes tempting to select projects that are high-profile and that could generate lots of visibility but that are badly conceived, too broad, too vague, or have too much political baggage that students should not have to deal with. One situation in which this may arise is when projects are proposed with research partners. Satisfying research partners through student projects is seldom a good idea. Such projects inevitably fail in one way or another, and impact especially students for whom the project can be a unique event during studies. It is better to select a project which perhaps looks slightly more boring at the outset, but is better suited for the educational environment. This depends also on the profile of the environment, which may vary from implementation to implementation. However, this is not to say that high-profile projects should be rejected on principle. Rather, if the project looks important, then it deserves the background work needed to prepare it for the project course. We try to avoid vanity projects by ensuring that there is wide-ranging interest towards projects among all stakeholders.

Anti-pattern 10: Planned to fail

- Projects that do not aim for the real desired outcome.
- Demotivating students by telling them there is already another plan.

For a project to be motivating, it must be believable that the outcome has some impact or value. In some situations, the project carried out in the educational setting is only one of many options that can be explored. This is a perfectly valid approach to learn in a start-up setting. However, when executed badly, this may mean that the educational project does not actually provide any valuable outcome – it is already known that the outcome will be discarded. Miscommunication of the chosen strategy can have a similar effect. It is important to ensure that the project aims for a real desired outcome, no matter how small. Otherwise, all stakeholders will merely go through the motions for the sake of finishing the project rather than actually engaging with the exploration of problem and solution spaces. We believe a key factor is how to arrange the customer's learning in an experiential course setting, but we are still exploring that question.

Anti-pattern 11: Initiation-expectation mismatch

- Projects that are started in a manner that does not support the expected outcome.
- Lack of clarity regarding how to work for a specific type of outcome.
- Lack of initial communication regarding expectations.

The desired outcome of a project must be taken into account when considering how to initiate it. A project may start in a very exploratory mode, but as the end approaches, it becomes clear that the customer actually wanted something specific all along. If a concrete outcome is desired, the project should start with a well-defined deliverable. In other words, it should operate more in the solution space. In some projects, the customer is genuinely unsure of what outcome they expect, and this pattern does not refer to those situations where the customer realises what they wanted as a result of the project. Rather, this pattern is about the situation where the customer does not communicate their true wishes from the beginning. This may occur out of politeness, as customers may believe that they must let students explore and have fun in the beginning of the project. It may also occur because the customer is afraid to reveal their true goals. In any case, the customer then initiates the project in a manner that leads down the wrong path. As in the previous pattern, we believe the customer's learning is a key factor in ensuring that the initiation matches expectations.

Anti-pattern 12: Feeding the strong

- Educational interventions and teacher attention placed too much on the most knowledgeable students.
- Failing to introduce interventions that balance student team skills.

Teachers may inadvertently nourish learning mostly or only among the most knowledgeable students. Also, they may contribute to a team hierarchy and conduct that lets only the strongest students steer and decide, and suppresses communication from the weaker students. This will lead to a situation where the strongest students perform nearly all project tasks, and the weaker students withdraw and try to hide the fact that they are not contributing much. This is unfair to all students. The stronger students will have an unreasonable workload as they must carry the entire project. Also, they will not accomplish as much as they could if they had the full force of the team behind them. The weaker students will feel left out of the project and their learning is likely to stop. The teacher must be alert and use educational interventions to balance the skills in the team. For example, we have helped students to form sub-groups where stronger students collaborate with weaker students, and introduced learning tasks where knowledge and skills can be transferred between students.

Anti-pattern 13: The customer who doesn't know students

- Customers that are not familiar with the conditions of student life.
- Overinterpreting the working life simulation.

Customers may have little understanding of what it means to be a university student. Although we emphasise an intensive project during which few other studies and duties should be performed, students' lives are different than that of paid workers. While we strive to simulate relevant conditions of working life, there are many aspects that are not possible or even desirable to simulate. These include monetary compensation for work, extra compensation and regulation of overtime, regulation of working conditions, and the existence of a legal entity and superiors which handle issues of legal responsibility. Customers may come to believe that they can deal with students and the student project in the same manner as they would deal with a contractor. They may, for example, exert pressure in inappropriate ways, start monitoring the project constantly, or demand detailed work reports that are sensible only as a way to confirm that billing corresponds to work delivered. It is important to communicate the nature of the project to customers beforehand, but in our experience, this is sometimes not enough. Fortunately, we have only encountered a few cases of this anti-pattern, and have been able to correct the situation by constructive dialogue.

Anti-pattern 14: The dissociative customer

- Customers being disconnected from the project.
- Assuming a role of only attending meetings.
- Too many or too incoherent customer representatives.

Some customers do not realise their responsibility and role in driving projects. A customer may have multiple representatives who do not coordinate among each other. Each time a representative meets the student team, they have no idea what the team agreed with the previous representative, and rather than being able to evaluate the work so far and decide on a coherent next step, they come with new and incompatible ideas. Customers should have one single contact person who is responsible for the communication and decision-making. Other representatives can contribute as experts on specific topics, but they should coordinate with the main contact person and their role should be advisory.

Anti-pattern 15: Believing it will never end

- Customers that do not understand that the project ends.
- Not being able to adjust to project phases.

Since our projects are intensive but relatively short, it is important for the customer to be able to adjust to different project phases. Especially the last phase of the project, where customers must choose between new features and stabilising existing features, is particularly important. However, some customers may believe that since they have flexible project deadlines, the university course can also be flexible in that regard. Customers need to understand that project will end on the designated date, and students are not going to continue after that. Customers must be prepared to focus on the level of stability and packaging that they require at the end of the project.

Anti-pattern 16: The helpdesk

- Customers contacting the university for software support after the project.
- Failing to properly hand over the project deliverables and assets.

It is not sustainable to provide support for customers after projects, even as a courtesy. When the project ends, it is useful to explicitly hand over the project deliverables and any assets and resources that the project has produced or used. Such proper handoff and delivery of the end result signals that the customer is now responsible for any further actions. Handoff must include all technical assets but also access and administration responsibility to all systems where assets are stored. If any university systems have been used, they must either be scoped out of the delivery or assets must be moved out of university-owned systems. For this reason, we strive as much as possible to use third-party code repositories and other systems where transferring assets to the customer is easy and there is then no connection to the university. Becoming a perpetual point of support for the customer is both time-consuming and can lead to disappointment when it becomes obvious that true support cannot be given.

6. Discussion

Throughout our Software Factory projects, we have observed recurring patterns of success and encumbrances. Our educational patterns and anti-patterns encapsulate insights that can contribute positively to the whole journey from project planning to completion; and to learning objectives of the course, thus providing an answer to our research question. A pertinent question is how the patterns and anti-patterns contribute to entrepreneurship and start-up education. In this section, we discuss our findings with respect to both the contextual and pedagogical backgrounds of our study and consider the implications for arranging similar learning experiences.

6.1. Entrepreneurship education

Previous work suggests that entrepreneurship education benefits from concrete experience through active participation [13]. This motivates our overall approach to software start-up education: an immersive learning environment in which students carry out projects. Previous research also shows that learning in an actual start-up could be detrimental to more junior students [14]. Placing the learning environment within university control – both administratively and physically – has the benefit of providing a safe environment where educational quality can be ensured, but with enough realism for participants.

By mapping our patterns and anti-patterns to the 15 competences defined in the EntreComp Entrepreneurship Competence Framework [9], and considering them in relation to the pedagogical frame, we can give a theoretically motivated reasoning for how they contribute to start-up education.

The (anti-)patterns for the physical and virtual environment and teacher guidance strive to create a learning environment that feels authentic for students. They do not directly contribute to any of the competences, but can be motivated from the perspective of situated cognition theory. Since knowledge construction is linked to the activity, context, and culture surrounding the learning environment, it is important to promote a context and culture that conveys the feeling of a real start-up environment. The patterns contribute to taking such steps.

The (anti-)patterns concerning course design and role in the curriculum, and those concerning educational interventions, attempt to address several competences: self-awareness and self-efficacy, motivation and perseverance, mobilising resources, mobilising others, taking the initiative, and learning through experience. Previous work in entrepreneurship education suggests focusing on personal attributes as well as tasks (c.f. [13]). Our patterns and anti-patterns in this category strive to direct learning precisely to personal attributes rather than focusing on technical tasks. That is not to say that the latter cannot also be part of the learning in start-up education; however, due to the diverse nature of our projects, it is difficult to extract general principles related to the technical learning. Many of the more technical tasks are perhaps better learned in other courses, and our department curriculum reform shows that they can be successfully taught at an earlier stage. Future work could consider how technical aspects should be considered in software start-up education.

Several of the anti-patterns we have presented do address issues that are threats also in real-life software projects. This illustrates the level of realism we strive towards: our educational environment operates on projects that are not toy projects but that include several aspects of real-life software projects. The anti-patterns concern issues of project governance – the kinds of issues that students cannot generally be expected to take responsibility for, and that teachers and the university must address.

6.2. Self-efficacy as the primary learning goal

In our view, strengthening students' identity as software developers, and improving their self-efficacy in relation to that identity, is the most important learning goal in our course. Behavioural modelling has been shown to improve self-efficacy (see Section 2.2). Several of our patterns include behavioural modelling. Also, the patterns 5, 6, 13, 14, and 16 explicitly address nurturing self-efficacy: they direct teachers to observe and react to student success when it occurs. Conversely, anti-patterns 1, 3, 7-10 and 12-14 strive to protect the development of self-efficacy beliefs by avoiding demotivation among students.

One important factor that may support the development of positive self-efficacy beliefs is psychological safety. An environment that is safe for interpersonal risk-taking should support individuals to communicate and pursue ideas that they would otherwise keep to themselves. A supportive environment rewards expression of such ideas, and when they are pursued, there are opportunities for learning. These positive experiences can be part of fostering self-efficacy. Naturally, some ideas will fail, but in an environment with high psychological safety, those failures can also be utilised as learning experiences.

Many of our patterns and anti-patterns strive to strengthen psychological safety. For instance, pattern 2 strives to create a foundation of routines and norms that contribute to psychological safety. Pattern 4 removes project decision-making authority from teachers, instead emphasising communication and evidence as the basis of decisions, meaning that contributing to decision-making is possible regardless of seniority or rank. Pattern 10 extends the psychological safety goal to cover teachers as well as students. Anti-patterns 9 and 12 are examples that strive to increase psychological safety by avoiding to build unsafe conditions in the first place. Anti-pattern 9 strives to avoid projects which, from the outset, are conflict-prone. Anti-pattern 12 strives to avoid damaging team cohesion by favouring some students at the expense of others. In one way or another, most of the patterns and anti-patterns presented can be seen as contributing to self-efficacy through strengthening of psychological safety. Simultaneously, the patterns do not strive to shield students from difficulties or failure, as realism is also among the main aims.

Ultimately, self-efficacy is complex and very hard to influence. To what extent a single course can have a lasting impact on self-efficacy remains an open question. However, feedback from students both immediately after projects as well as after 1–2 years provides anecdotal evidence that the Software Factory experience is memorable and has a positive effect on students' willingness to work with developing innovative software-intensive products. Negative feedback given by students focuses more on difficulties of completing other studies during the intensive project, the short time available and the resulting constraints, and technical obstacles that students were frustrated by during the course. Further study is needed to understand how and to what extent the course improves students' self-efficacy beliefs. Inflated self-efficacy beliefs can hinder learning and should also be considered.

6.3. Limitations of the study

The results of this study are based on a retrospective, reflective analysis of several Software Factory projects. The internal validity and trustworthiness of the results should be high due to the use of 26 projects as cases, triangulation through multiple types of data, researcher triangulation, and traceability to evidence in the analysis. The external validity and transferability of the results is limited by several factors. Most importantly, it is uncertain whether the patterns and course set-up described here can be successfully enacted without a B.Sc. program similar to the one described in the background section. Students' prerequisite knowledge needs to be at a high level, and they must be used to the mode of teaching that we employ. Furthermore, implementing the patterns places demands on staff. It requires a special stamina in coaching the several, iteratively improving areas that contribute to students' self-directedness and motivation, as can be seen in patterns 7 and 8. Finally, we note that the patterns and anti-patterns presented here are inductively derived from the source material, and not empirically tested in different conditions. Taking these limitations into account, we suggest that teachers can apply them in their own project-based start-up courses.

7. Conclusion

Software engineering students often have a strong technical background in CS subjects, but lack the knowledge and skills to enact group work projects in an entrepreneurial environment. In this paper, we retrospectively analysed seven years of educational projects with start-up-like traits and developed 16 educational patterns and 16 anti-patterns for enhancing software start-up instruction in higher education. The patterns and anti-patterns cover the physical and virtual environment, course design and placement in the curriculum, learning materials, and teacher guidance.

Besides the patterns, we discuss the prerequisites for software start-up education. A thorough reform of the curriculum may be needed to achieve the desired learning outcomes, and prepare students for a world where entrepreneurship may be a dominant form of employment. Future studies could address how software start-up education can help build students' developer identities and enhance their self-efficacy beliefs, as well as examine how technical knowledge and skills should be considered in start-up education. Further extension and validation of the patterns and anti-patterns, as well as in-depth study on the customer's learning and integration into university-led experiential, project-based education, are among the potential future directions in this area.

References

[1] European Commission, Entrepreneurship 2020 Action Plan, online: http://bit.ly/2oBRevI [Retrieved: 2017-04-12] (2012).

[2] D. Rombach, et al., Teaching disciplined software development, Journal of Systems and Software 81 (5) (2008) 747–763.

[3] P. Abrahamsson, P. Kettunen, F. Fagerholm, The set-up of a software engineering research infrastructure of the 2010s, in: Proc. of the 11th International Conf. on Product Focused Software, ACM, 2010, pp. 112–114.

[4] F. Fagerholm, N. Oza, J. Münch, A platform for teaching applied distributed software development: The ongoing journey of the Helsinki software factory, in: 2013 3rd International Workshop on Collaborative Teaching of Globally Distributed Software Development, 2013, pp. 1–5.

[5] J. Dewey, How we think: A restatement of the relation of reflective thinking to the educative process, DC Heath, Boston, MA, 1935.

[6] D. A. Schön, The reflective practitioner: How professionals think in action, Basic Books, New York, NY, USA, 1983.

[7] F. Fagerholm, A. Hellas, M. Luukkainen, K. Kyllönen, S. Yaman, H. Mäenpää, Patterns for Designing and Implementing an Environment for Software Start-Up Education, in: 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2017, pp. 133–140.

[8] Commission of the European Communities, Green Paper – Entrepreneurship in Europe, online: http://bit.ly/2pDkvUj [Retrieved: 2017-04-12] (2003).

[9] M. Bacigalupo, P. Kampylis, Y. Punie, L. Van den Brande, EntreComp: The Entrepreneurship Competence Framework, Publications Office of the European Union, Luxembourg, 2016, EUR 27939 EN.

[10] W. B. Gartner, Who is an entrepreneur? Is the wrong question, American Journal of Small Business 12 (4) (1988) 11–32.

[11] W. Fletcher, The Management of Creativity, International Journal of Advertising 9 (1) (1990) 1–37.

[12] H. Berglund, K. Wennberg, Creativity among entrepreneurship students: comparing engineering and business education, International Journal of Continuing Engineering Education and Life-Long Learning 16 (5) (2006) 366.

[13] G. Gorman, D. Hanlon, W. King, Some Research Perspectives on Entrepreneurship Education, Enterprise Education and Education for Small Business Management: A Ten-Year Literature Review, International Small Business Journal 15 (3) (1997) 56–77.

[14] S. Chenoweth, Undergraduate Software Engineering Students in Startup Businesses, in: 21st Conf. on Software Engineering Education and Training, IEEE, 2008, pp. 118–125.

[15] A. Järvi, V. Taajamaa, S. Hyrynsalmi, Lean Software Startup – An Experience Report from an Entrepreneurial Software Business Course, in: J. M. Fernandes, R. J. Machado, K. Wnuk (Eds.), Software Business: 6th International Conf., ICSOB 2015, Braga, Portugal, June 10-12, 2015, Proceedings, Springer International Publishing, 2015, pp. 230–244.

[16] R. Harms, Self-regulated learning, team learning and project performance in entrepreneurship education: Learning in a lean startup environment, Tech. Forecasting and Social Change 100 (2015) 21–28.

[17] A. Bandura, Self-efficacy: toward a unifying theory of behavioral change., Psychological review 84 (2) (1977) 191.

[18] A. Bandura, The explanatory and predictive scope of self-efficacy theory, Journal of social and clinical psychology 4 (3) (1986) 359–373.

[19] T. Judge, et al., Are measures of self-esteem, neuroticism, locus of control, and generalized self-efficacy indicators of a common core construct?, J. of Personality and Social Psych. 83 (3) (2002) 693–710.

[20] J. R. Engelsma, Best Practices for Industry-sponsored CS Capstone Courses, J. of Computing Sciences in Colleges 30 (1) (2014) 18–28.

[21] K. D. Multon, S. D. Brown, R. W. Lent, Relation of self-efficacy beliefs to academic outcomes: A meta-analytic investigation. (1991).

[22] B. Hasan, The influence of specific computer experiences on computer self-efficacy beliefs, Computers in human behavior 19 (4) (2003) 443–450.

[23] V. Ramalingam, D. LaBelle, S. Wiedenbeck, Self-efficacy and Mental Models in Learning to Program, in: Proc. of the 9th Conf. on Innovation and Technology in CS Education, ACM, 2004, pp. 171–175.

[24] A. Bandura, Perceived self-efficacy in cognitive development and functioning, Educational psychologist 28 (2) (1993) 117–148.

[25] M. E. Gist, C. Schwoerer, B. Rosen, Effects of alternative training methods on self-efficacy and performance in computer software training., Journal of applied psychology 74 (6) (1989) 884.

[26] A. Collins, J. S. Brown, S. E. Newman, Cognitive apprenticeship: Teaching the craft of reading, writing and mathematics, Thinking: The Journal of Philosophy for Children 8 (1) (1988) 2–10.

[27] A. Collins, J. S. Brown, A. Holum, Cognitive apprenticeship: Making thinking visible, American educator 15 (3) (1991) 6–11.

[28] G. Norman, H. Schmidt, The psychological basis of problem-based learning: a review of the evidence, Acad. medicine 67 (9) (1992) 557–65.

[29] J. S. Brown, A. Collins, P. Duguid, Situated cognition and the culture of learning, Educational researcher 18 (1) (1989) 32–42.

[30] B. Barron, et al., Doing with understanding: Lessons from research on problem-and project-based learning, J. of the Learning Sciences 7 (3-4) (1998) 271–311.

[31] M. Luukkainen, et al., Three years of design-based research to reform a software engineering curriculum, in: Proc. of the 13th annual conf. on Information technology education, ACM, 2012, pp. 209–214.

[32] A. Vihavainen, et al., Extreme apprenticeship method: key practices and upward scalability, in: Proceedings of the 16th annual joint conf. on Innovation and technology in computer science education, ACM, 2011, pp. 273–277.

[33] The Design-Based Research Collective, Design-based research: An emerging paradigm for educational inquiry, Educational Researcher 32 (1) (2003) 5.

[34] P. Bell, On the Theoretical Breadth of Design-Based Research in Education, Educational Psychologist 39 (4) (2004) 243–253.

[35] D. C. Edelson, Design Research: What We Learn When We Engage in Design, Journal of the Learning Sciences 11 (1) (2002) 105–121.

[36] H. Burkhardt, A. H. Schoenfeld, Improving Educational Research: Toward a More Useful, More Influential, and Better-Funded Enterprise, Educational Researcher 32 (9) (2003) 3–14.

[37] P. Cobb, et al., Design experiments in educational research, Educational Researcher 32 (1) (2003) 9.

[38] T. Anderson, J. Shattuck, Design-Based Research, Educational Researcher 41 (1) (2012) 16–25.

[39] A. Hevner, et al., Design Science in Information Systems Research, MIS Quarterly 28 (1) (2004) 75–105.

[40] K. M. Eisenhardt, Building Theories from Case Study Research, The Academy of Management Review 14 (4) (1989) 532–550.

[41] J. Münch, F. Fagerholm, P. Johnson, J. Pirttilahti, J. Torkkel, J. Järvinen, Creating Minimum Viable Products in Industry-Academia Collaborations, in: B. Fitzgerald, K. Conboy, K. Power, R. Valerdi, L. Morgan, K.-J. Stol (Eds.), Lean Enterprise Software and Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 137–151.

[42] F. Fagerholm, A. S. Guinea, H. Mäenpää, J. Münch, Building Blocks for Continuous Experimentation, in: Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, RCoSE 2014, ACM, New York, NY, USA, 2014, pp. 26–35.

[43] O. Liskin, K. Schneider, F. Fagerholm, J. Münch, Understanding the Role of Requirements Artifacts in Kanban, in: Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2014, ACM, New York, NY, USA, 2014, pp. 56–63.

[44] F. Fagerholm, A. Vihavainen, Peer assessment in experiential learning Assessing tacit and explicit skills in agile software engineering capstone projects, in: Frontiers in Education Conference (FIE), IEEE, 2013, pp. 1723–1729.

Fabian Fagerholm is a postdoctoral researcher at the University of Helsinki, Finland. His research interests include developer experience, human, behavioural, and psychological aspects of software engineering, continuous experimentation and evidence-driven software product development, open source software development, and experiential and project-based software engineering education. He has coordinated the design, planning, implementation, and operation of the Software Factory laboratory for experimental software engineering research and education since its inception. He received his PhD in computer science from the University of Helsinki, Finland.

Arto Hellas is a university instructor at the University of Helsinki, Finland. His research interests include computer science education, learning analytics, and MOOC learning. He received his PhD in computer science from the University of Helsinki, Finland.

Matti Luukkainen is a university lecturer at the University of Helsinki, Finland. His research interests include computer science education, the extreme apprenticeship method, and curriculum development. He received his PhD in computer science from the University of Helsinki, Finland.

Kati Kyllönen is a project manager at Digia Plc, and was previously a research and teaching assistant at the University of Helsinki, Finland. She has more than 15 years of experience working in the software industry, and has coached multiple projects in the Software Factory laboratory. She was awarded with the junior teacher award at the University of Helsinki in 2016. She holds a vocational qualification in Business IT from the ADP Institute, Finland, and is currently finalizing her M.Sc. studies in Computer Science at the University of Helsinki, Finland.

Sezin Yaman is a doctoral student at the University of Helsinki, Finland. Her research interests include data- and experiment-driven software engineering and customer involvement in software development. She received her M.Sc. in computer science from the University of Helsinki, Finland.

Hanna Mäenpää is a doctoral student at the University of Helsinki, Finland. Her research interests include open innovation, open source software development, and problem- and project-based education. She received her M.Sc. in computer science from the University of Helsinki, Finland.

ACCEPTED MANUSCRIPT