Master's thesis

Master's Programme in Computer Science

Internal software startup within a university – producing industry-ready graduates

Saara Satokangas

May 9, 2022

Faculty of Science

University of Helsinki

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Saara Satokangas

Työn nimi — Arbetets titel — Title

Internal software startup within a university – producing industry-ready graduates

Ohjaajat — Handledare — Supervisors

Prof. T. Männistö, Ph.D. M. Luukkainen

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	May 9, 2022	58 pages, 8 appendix pages

Tiivistelmä — Referat — Abstract

Tertiary education aims to prepare computer science students for the working life. While much of the technical principles are covered in lower-level courses, team-based capstone projects are a common way to provide students hands-on experience and teach soft skills. Although such courses help students to gain some of the relevant skills, it is difficult to simulate in a course context what work in a professional software engineering team really is about.

Our goal is to understand ways tertiary education institutions prepare students for the working life in software engineering. Firstly, we do this by focusing on the mechanisms that software engineering capstones use to simulate work-life. A literature review of 85 primary studies was conducted for this overview. Secondly, we present a more novel way of teaching industry-relevant skills in an university-lead internal software startup. A case study of such a startup, Software Development Academy (SDA), is presented, along with the experiences of both students and faculty involved in it. Finally, we look into how these approaches might differ.

Results indicate that capstone courses differ greatly in ways they are organized. Most often students are divided in teams of 4–6 and get assigned with software projects that the teams then develop from an idea to a robust proof-of-concept. In contrast, students employed in the SDA develop production-level software in exchange for a salary for university clients. Students regarded SDA as a highly relevant and fairly irreplaceable educational experience. Working with production-quality software and having a wide range of responsibilities was perceived integral in giving a thorough skill set for the future.

In conclusion, capstones and the internal startup both aim to prepare students for the work-life in software engineering. Capstones do it by simulating professional software engineering in a one-semester experience in a course environment. The internal startup adds a touch of realism to this by being actual work in a relatively safe university context.

ACM Computing Classification System (CCS)
Social and professional topics → Professional topics → Computing education

Avainsanat — Nyckelord — Keywords

software engineering, capstone projects, teaching, students, internal startup

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Software study track

Contents

1 Introduction

Universities and other tertiary education institutions should provide their computer science students with sufficient skills and abilities before the students enter working life. In software engineering related programs, this entails having an understanding of the common principles and theory in computer science (ACM/IEEE, 2013; ACM/IEEE, 2014) and technical competencies and knowledge demanded by the industry (Radermacher et al., 2014; Garousi et al., 2019). Any recent graduate should also have the ability to apply this technical knowledge in practice (ACM/IEEE, 2013).

While much of the technical knowledge and theories are covered in lower-level courses, many institutions hold team-based capstone project courses to ensure students are ready to apply the knowledge in a workplace. The main goal of a capstone project is to provide hands-on experience in applying the tools, techniques, principles and best practices that are taught more theoretically in previous courses (Ziv and Patil, 2010; Majanoja and Vasankari, 2018; Panicker et al., 2020). Capstone projects are also regarded as crucial in teaching students the necessary soft skills such as teamwork (Keogh et al., 2007; Venson et al., 2016), verbal and written communication (Watkins and Barnes, 2010), time management (Dupuis et al., 2010), problem solving (Majanoja and Vasankari, 2018) and project management (Haddad, 2013). For students, a capstone project typically represents a culmination of their studies and is one of the last milestones before graduation (ACM/IEEE, 2013).

In software engineering related programs, the projects typically last one or two semesters (Ikonen and Kurhila, 2009; Adams and Kleiner, 2016; Schneider et al., 2020) and are sometimes organized with industry clients (Fornaro et al., 2007; Isomöttönen and Kärkkäinen, 2008; Majanoja and Vasankari, 2018; Spichkova, 2019). Although such courses help students to gain relevant skills and provide an important sneak peek into the work-life in software engineering, the essence of what work in a professional software team is all about, is hard to capture. Moreover, previous research has demonstrated that preparing students for the life in software engineering is not a trivial task. There are still clear gaps between what students have done in their software engineering projects at the university level and what they are expected to do in the industry (Begel and Simon, 2008; Radermacher and Walia, 2013; Radermacher et al., 2014; Garousi et al., 2019).

To gain a thorough understanding of the mechanisms universities use to prepare their students for the software engineering industry, we firstly present an literature review done on software engineering capstone courses. It gives an outlook of the organizatorial aspects of these courses and their reported outcomes. Any areas where capstone courses might fall short of a realistic industry-experience are also look into. Additionally, as a solution to these shortcomings, this research describes our experiences of running an internal software startup in the University of Helsinki, called Software Development Academy (SDA). SDA was founded in 2017 and is a team made of computer science students and lead by faculty. SDA both develops and maintains educational administrative software used within the University of Helsinki. It combines the best parts of capstone projects, external experiences like internships and creating software for the purposes of an academic institution. This research presents the framework for SDA and explains how it has helped us to bridge the gap between industry expectations and traditional software engineering education. We also present the ways it differs from capstone courses.

The thesis is organized as follows. The next chapter presents relevant concepts for understanding the context of this research. Chapter 3 describes the goal of this research along with the research questions formulated to reach that goal. The two-part approach used to conduct the research is also described and a framework for an internal software startup in the university context, Software Development Academy (SDA), presented. Chapter 4 outlines the results of the semi-systematic literature review and how the academic institutions traditionally conduct their software engineering capstones. Chapter 5 describes the experiences of students and faculty involved in the SDA. The differences to traditional capstone courses are also discussed. The results, their validity and related work are discussed in Chapter 6. Finally, Chapter 7 concludes the research and makes proposals for future work.

2 Background

This research focuses on describing the ways tertiary education institutions prepare their students for the working life in the software engineering (SE) industry. This chapter presents a brief overview of the central concepts and ideas related to the research area. Firstly, Section 2.1 explains what gaps earlier literature has identified between the skills of recent graduates and the industry expectations. Section 2.2 then describes a common way universities aim to ensure that their students have the necessary skills: capstone project courses. Finally, Section 2.3 gives a short overview of an internal software startup in a university context, Software Development Academy.

2.1 Gaps in industry-relevant skills

Increasingly employers and society are requiring universities to produce graduates with appropriate "employability skills" such as communication and self-management (Johns-Boast and Flint, 2013). The emphasis of each programme is unique, but some of the commonly mentioned soft skills that students should possess after completing their studies are teamwork (Keogh et al., 2007; Ziv and Patil, 2010; Delgado et al., 2017; Marques et al., 2017; Iacob and Faily, 2019), client negotiation skills (Keogh et al., 2007), project management (Haddad, 2013) and a general ability to function in the software engineering industry (Ziv and Patil, 2010; Mahnic, 2011). These skills are especially in the center of capstone courses' learning goals (ACM/IEEE, 2013; ACM/IEEE, 2014). Despite the perceived high importance and focus, these learning goals are still not fully met. In a systematic literature review on the skill gaps of computer science and software engineering graduates, written communication tied with oral communication as the most commonly identified knowledge deficiency (Radermacher and Walia, 2013). In the same review, project management came in third (Radermacher and Walia, 2013). Another systematic literature review on the gap between SE education and industry expectations found that professional practice, containing professionalism, group dynamics and communication skills, is perceived as highly important in the industry (Garousi et al., 2019). Yet it still presents a high knowledge deficiency and therefore is something that the educators should pay close attention to (Garousi et al., 2019).

What it comes to technical skills, the knowledge deficiences do not seem to be as imminent or all-encompassing since the top three positions in the systematic literature review go to soft skills (Radermacher and Walia, 2013). Similar notions have been made elsewhere (Begel and Simon, 2008; Stevens and Norman, 2016). The technical foundation has been found to be sufficient and many industry representatives acknowledge the need to extend the specific technological skills of graduates themselves (Stevens and Norman, 2016). The technical skills that industry representatives most report graduates not having, relate to software development tools and configuration management (Begel and Simon, 2008; Radermacher and Walia, 2013; Radermacher et al., 2014; Garousi et al., 2019). In a study of recent graduates' skills, industry managers indicated that their new employees had not been exposed to configuration management tools before (Radermacher et al., 2014). Several managers also stated that their recent hires likely had not used software tools in a production environment before and might have lacked an inherent understanding of why a production environment should be used (Radermacher et al., 2014). Configuration management has been identified as being of high importance in the industry while being the topic with the highest gap between industry expectations and software engineering education (Garousi et al., 2019).

In light of all this, there clearly are areas in which the current education on software engineering has room for improvement. Project management, communication skills and how to use software development tools in a production-environment are issues that should be better tackled in education.

2.2 Capstone project courses

While much of the technical and theoretical background is being covered in basic-level software engineering courses, a large portion of the necessary soft skills are being taught in project-based courses, such as software engineering capstones. A "capstone course" usually means a course that finishes an academic degree (Ikonen and Kurhila, 2009). In computer science (CS) and software engineering (SE) programmes, capstone courses generally last one or two semesters and they include assigning students into teams and having them work on various kinds of software engineering projects (Ikonen and Kurhila, 2009; Bowring and Burke, 2016; Paasivaara et al., 2019). In these projects they are expected to experience stages of the software development lifecycle from requirements solicitation to software maintenance (Buffardi et al., 2017).

ACM/IEEE Curriculum Guidelines for SE programs (ACM/IEEE, 2014) regard the capstone project as an essential element of a SE degree program. According to the guidelines, capstones are integral in ensuring that the curriculum has a significant real-world basis. To make the projects as realistic as possible, capstones should be held with clients that are external to the course staff (ACM/IEEE, 2014). The ACM/IEEE Curriculum Guidelines for CS programs (ACM/IEEE, 2013) align with these views and state that all graduates of CS programs should have been involved in at least one substantial project. Such projects should challenge students by being integrative, requiring evaluation of potential solutions and work on a larger scale than typical course projects. Students should also have opportunities to develop their interpersonal communication skills as part of their project experience (ACM/IEEE, 2013). The University of Helsinki also provides a mandatory capstone project course, called Software Engineering Project, for the students in Computer Science Bachelor Program. Students are expected to take the course after they have passed most of the Bachelor-level courses, typically, in their third year. The course lasts for 14 weeks, and students are assigned into teams of 4–6 students, based on their self-assessed skills and preferences regarding project topics. The clients for the projects comprise of local businesses as well as research groups and various departments of the University of Helsinki. All clients are external, and none of the teaching staff of the course act as clients. The manager and founder of the Software Development Academy, has also been the responsible teacher for the course since 2009.

2.3 Software Development Academy

Software Development Academy (SDA) is an internal non-profit software startup comprised of computer science students and lead by faculty within the University of Helsinki. SDA develops and maintains several educational administrative applications for the use of the entire university. Selected students work in the SDA for one year, consisting of part-time work during the academic term and full-time work during the summer. During the year, each student has a wide range of responsibilities in software development and maintenance. The team is managed and lead by a senior lecturer of computer science. Since its founding in 2017, the SDA has employed 29 students and successfully created and maintained 9 applications. A more thorough description of the SDA is given in Section 3.3.

3 Research approach

This chapter defines the research methodology used in this thesis. Firstly, Section 3.1 describes the goal of this research and the research questions derived from it. These are followed by a graph visualising the study design (Figure 3.1). Section 3.2 further discusses the process of conducting the semi-systematic literature review and how its results were analysed. Section 3.3 presents the methdology and the case used in the case study part of this research.

3.1 Research questions

The main goal of this research is to understand how universities and other tertiary education institutions prepare their students for the working life in software engineering. This goal is approached from two separate viewpoints. Firstly, we aim to describe how industry-relevant skills are generally being taught in software engineering capstones. Secondly, we investigate a more novel way of teaching these skills in a university-lead internal startup. Finally, we try to identify how these two approaches might differ and what common traits they share. To reach this goal, the following research questions were formulated:

RQ1: What mechanisms traditional capstone courses use to prepare students for the software engineering industry?

RQ2: What mechanisms does a university-lead internal software startup use to prepare students for the software engineering industry?

RQ3: How do traditional capstone courses and a university-lead internal software startup differ?

To answer RQ1, we aim to create an overview of how tertiary education institutions use capstone project courses to prepare their students for the working life in software engineering. As described in Section 2.2, capstone courses are the de facto method for teaching industry-relevant skills and giving students realistic software engineering experience. This overview is gained by conducting a semi-systematic literature review on the subject. Semi-systematic reviews are good for topics where the research questions are fairly broad and analysis and evaluation can also be qualitative (Snyder, 2019). This is in contrast to sys-

tematic literature reviews, where the research questions are expected to be specific and followed by a quantitative evaluation of the results (Snyder, 2019). A semi-systematic literature review can be especially useful for detecting themes or for synthesizing the state of knowledge for a certain research field (Snyder, 2019), which is why it suits the purposes of this research well. As the focus is on universities, any internships and co-ops, where the organizing party is not a tertiary education institution, are out of the scope of this research.

In order to answer RQ2, a case study of an internal startup in the university context, Software Development Academy (SDA) is investigated. By describing the framework, and finding out the experiences of both students and faculty, we hope to shed light on how the startup affects students' skills and knowledge. Finally, for RQ3, the study combines the knowledge gained from answering RQ1 and RQ2 and provides a comparison of the two approaches.

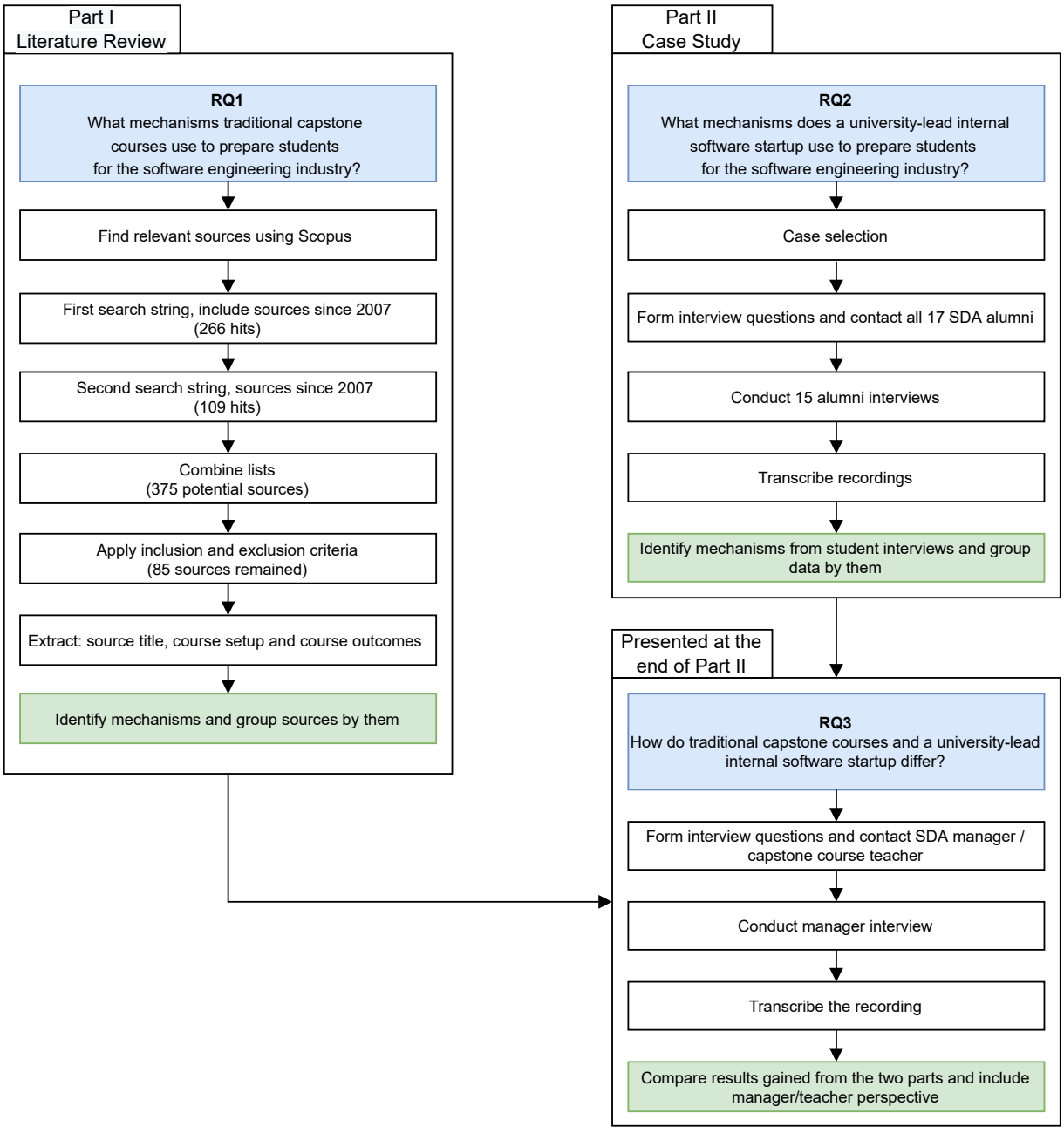

Figure 3.1: Research approach

3.2 Part I: Semi-systematic literature review

In order to get a comprehensive representation of how project-based capstone courses are traditionally organized, relevant literature and experience reports were searched. This sections describes that process. It is also summarized on the left side of Figure 3.1.

3.2.1 Constructing the search strings

The data collection was done by finding relevant sources from the abstract and citation database Scopus in March 2022. The search was limited to sources in their final stage (final) and within the Computer Science -subject area (COMP). The search included only articles (ar) and conference proceedings (cp). In order to have a complete picture of the project course landscape in software engineering, a second search was performed using the second search string presented. As not all relevant sources had the word “capstone“ in their title, abstract or keywords, this second search was deemed necessary.

First search string - Capstone projects

```
(
    TITLE-ABS-KEY ( software  AND  engineering  AND  "capstone" )
    AND  ( LIMIT-TO ( PUBSTAGE ,  "final" ) )
    AND  ( LIMIT-TO ( SUBJAREA ,  "COMP" ) )
    AND  ( LIMIT-TO ( DOCTYPE ,  "cp" )
    OR  LIMIT-TO ( DOCTYPE ,  "ar" ) )
)
```

Second search string - Project courses

```
(
    TITLE-ABS-KEY (
        software  AND  engineering AND  "project course"  AND NOT  "capstone"
    )
    AND  ( LIMIT-TO ( PUBSTAGE ,  "final" ) )
    AND  ( LIMIT-TO ( SUBJAREA ,  "COMP" ) )
    AND  ( LIMIT-TO ( DOCTYPE ,  "cp" ) )
    OR  LIMIT-TO ( DOCTYPE ,  "ar" ) )
)
```

3.2.2 Inclusion and exlusion criteria

The titles and abstracts of the initial sources were read and evaluated agains the inclusion and exclusion criteria. In few unclear cases, the introduction and conclusions presented in the study were read to make a justified decision. After selection, 85 studies remained. These were ought to give a reasonably comprehensive view on how the capstone courses are organized. These final sources are listed in Table 3.1, and with full publication details in Appendix C.

The source should meet all of the following inclusion criteria:

- The title or abstract strongly hints that the study presents frameworks or case studies of software engineering capstones or other large project-based courses in software engineering
- The study experiments with a method of teaching software engineering in project-based capstone courses
- The title or abstract indicates that the study assesses the outcomes of the course
- The study is not older than 15 years

The source should not meet any of the following exclusion criteria:

- The study focuses on describing an assessment tool for such project courses as the focus of this research was on course outcomes, rather than tool generation
- The study focuses on describing tools for forming student teams in such courses, as the focus of this research was on course outcomes, rather than tool generation
- The study contains only workshop proceedings
- The study is a work in progress
- The study does not have full text available

Table 3.1: Included sources for data extraction

SID	Title	Year, Publication venue
S1	A global and competition-based model for fostering technical and soft skills in software engineering education	2009, 22nd Conference on Software Engineering Education and Training
S2	A holistic capstone experience: Beyond technical ability	2017, 18th Annual Conference on Information Technology Education
S3	A look at software engineering risks in a team project course	2013, 26th International Conference on Software Engineering Education and Training (CSEE T)
S4	A Scalable and Portable Structure for Conducting Successful Year-long Undergraduate Software Team Projects	2007, Journal of Information Technology Education: Research 6(1)
S5	A scalable approach to graduate student projects: hundreds with industry every year	2009, 22nd Conference on Software Engineering Education and Training
S6	A scalable model of community-based experiential learning through courses and international projects	2018, World Engineering Education Forum - Global Engineering Deans Council (WEEF-GEDC)
S7	A software engineering senior design project inherited from a partially implemented software engineering class project	2007, 37th Annual Frontiers In Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports
S8	Academia-academia-industry collaborations on software engineering projects using local-remote teams	2009, 40th ACM technical symposium on Computer science education
S9	Academic education of software engineering practices: Towards planning and improving capstone courses based upon intensive coaching and team routines	2013, 26th International Conference on Software Engineering Education and Training (CSEE T)
S10	Academy-industry collaboration and the effects of the involvement of undergraduate students in real world activities	2016, IEEE Frontiers in Education Conference (FIE)
S11	Accommodating Shortened Term Lengths in a Capstone Course using Minimally Viable Prototypes	2020, IEEE Frontiers in Education Conference (FIE)
S12	Adopting industry agile practices in large-scale capstone education	2020, 42nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)
S13	Advantages of agile methodologies for software and product development in a capstone design project	2014, IEEE Frontiers in Education Conference (FIE)
S14	An academia-industry collaborative teaching and learning model for software engineering education	2009, 21st International Conference on Software Engineering and Knowledge Engineering
S15	An Agile embedded systems capstone course: Overview, experiences, and lessons learned	2013, IEEE Frontiers in Education Conference (FIE)
S16	An exploration of knowledge and skills transfer from a formal software engineering curriculum to a capstone practicum project	2011, 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE T)
S17	An industry-academia team-teaching case study for software engineering capstone courses	2008, 38th Annual Frontiers in Education Conference
S18	An innovative approach to Software Engineering term projects, coordinating student efforts between multiple teams over multiple semesters	2014, IEEE Frontiers in Education Conference (FIE)
S19	Assessing the capability and maturity of capstone software engineering projects	2008, Conferences in Research and Practice in Information Technology Series
S20	Capstone project: From software engineering to “Informatics“	2010, 23rd IEEE Conference on Software Engineering Education and Training

SID	Title	Year, Publication venue
S21	Capstone courses under the PBL methodology approach, for engineering	2019, IEEE World Engineering Education Conference: Modern Educational Paradigms for Computer and Engineering Career
S22	Collaborating with industrial customers in a capstone project course: The customers' perspective	2019, 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)
S23	Collaboration support in an international computer science capstone course	2016, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)
S24	Collaborative security risk estimation in agile software development	2019, Information & Computer Security 26(4)
S25	Competitive and agile software engineering education	2010 IEEE SoutheastCon (SoutheastCon)
S26	Deployment of Capstone Projects in Software Engineering Education at Duy Tan University as Part of a University-Wide Project-Based Learning Effort	2013, Learning and Teaching in Computing and Engineering, LaTiCE
S27	Designing a multi-disciplinary software engineering project	2009, 22nd Conference on Software Engineering Education and Training
S28	Discovering high-impact success factors in capstone software projects	2009, 10th ACM conference on SIG-information technology education
S29	Evaluating capstone project through flexible and collaborative use of Scrum framework	2015, IEEE Frontiers in Education Conference (FIE)
S30	Evaluating test-driven development in an industry-sponsored capstone project	2009, Sixth International Conference on Information Technology: New Generations
S31	Evolving a Project-Based Software Engineering Course: A Case Study	2017, 30th Conference on Software Engineering Education and Training (CSEE T)
S32	Experience report: A sustainable serious educational game capstone project	2013, CGAMES'2013 USA
S33	Experiments with adding to the experience that can be acquired from software courses	2010, Seventh International Conference on the Quality of Information and Communications Technology
S34	Exploiting multiplicity to teach reliability and maintainability in a capstone project	2007, 20th Conference on Software Engineering Education Training (CSEET'07)
S35	Exploring the gap between the student expectations and the reality of teamwork in undergraduate software engineering group projects	2019, Journal of Systems and Software 157
S36	Exposing Students to a State-of-the-art Problem through a Capstone Project	2020, IEEE Frontiers in Education Conference (FIE)
S37	Facilitating entrepreneurial experiences through a software engineering project course	2019, 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)
S38	Global vs. local - Experiences from a distributed software project course using agile methodologies	2015, IEEE Frontiers in Education Conference (FIE)
S39	How does participating in a capstone project with industrial customers affect student attitudes?	2018, 40th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)
S40	Impact of software development processes on the outcomes of student computing projects: A tale of two universities	2022, Information and Software Technology 144
S41	Improving Software Engineering education through an empirical approach: Lessons learned from capstone teaching experiences	2014, 45th ACM technical symposium on Computer science education
S42	Improving the capstone project experience: A case study in software engineering	2008, 46th Annual Southeast Regional Conference on XX

SID	Title	Year, Publication venue
S43	Industry agile practices in large-scale capstone projects	2020, 42nd International Conference on Software Engineering: Companion Proceedings
S44	Industry-emulated projects in the classroom	2015, 16th Annual Conference on Information Technology Education
S45	Industry-oriented project-based learning of software engineering	2019, 24th International Conference on Engineering of Complex Computer Systems (ICECCS)
S46	Infusing Design Thinking into a Software Engineering Capstone Course	2017, 30th Conference on Software Engineering Education and Training (CSEE T)
S47	Integrating fundamental and advanced concepts in a rounded capstone design experience in computer engineering	2007, 37th Annual Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports
S48	Integrating international students' contests with computer sciecnce capstone: Lessons learned and best practices	2012, IEEE Frontiers in Education Conference
S49	Introducing good design principles in an early system engineering course	2009, Information Systems Education Conference, ISECON 26
S50	Introduction of continuous delivery in multi-customer project courses	2014, Companion Proceedings of the 36th International Conference on Software Engineering
S51	Investigating Students' Metacognitive Skills while Working on Information Systems Development Projects	2017, 7th World Engineering Education Forum (WEEF)
S52	Lessons learned managing distributed software engineering courses	2014, Companion Proceedings of the 36th International Conference on Software Engineering
S53	One-semester CS capstone: A 40-60 teaching approach	2013, 10th International Conference on Information Technology: New Generations
S54	Overcoming limited resources: An academia-government partnership on Software Engineering and capstone projects	2007, 37th Annual Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports
S55	Practical Software Engineering Capstone Course – Framework for Large, Open-Ended Projects to Graduate Student Teams	2019, Communications in Computer and Information Science 1022
S56	Preparing software engineering graduates for an industry career	2007, 20th Conference on Software Engineering Education Training (CSEET'07)
S57	Reflections of nine years of interdisciplinary capstone courses	2010, IEEE Frontiers in Education Conference (FIE)
S58	Reflections on 10 years of sponsored senior design projects: Students win-clients win!	2007, Journal of Systems and Software 80(8)
S59	Reflections on teaching software engineering capstone course	2018, 10th International Conference on Computer Supported Education
S60	Relating student, teacher and third-party assessments in a bachelor capstone project	2017, Communications in Computer and Information Science 770
S61	Retrospectives in a software engineering project course: Getting students to get the most from a project experience	2011, 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE T)
S62	Scrumming with educators: Cross-departmental collaboration for a summer software engineering capstone	2015, International Conference on Learning and Teaching in Computing and Engineering

SID	Title	Year, Publication venue
S63	Simulating industry: An innovative software engineering capstone design course	2013, IEEE Frontiers in Education Conference (FIE)
S64	Skills Development Through Agile Capstone Projects	2021, Communications in Computer and Information Science 1523 CCIS
S65	SLPC++: Teaching software engineering project courses in industrial application landscapes - A tutorial	2011, 24th IEEE-CS Conference on Software Engineering Education and Training, CSEE and T
S66	Software creationworkshop: A capstone course for business-oriented software engineering teaching	2018, ACM International Conference Proceeding Series
S67	Software engineering education: A study on conducting collaborative senior project development	2011, Journal of Systems and Software 84(3)
S68	Software engineering practicum course experience	2010, 23rd IEEE Conference on Software Engineering Education and Training
S69	Software engineering problems and their relationship to perceived learning and customer satisfaction on a software capstone project	2018, Journal of Systems and Software 137
S70	Software engineering project courses with industrial clients	2015, ACM Transactions on Computing Education 15(4)
S71	Software engineering senior design course: Experiences with agile game development in a capstone project	2011, International Conference on Software Engineering
S72	Splat! er, shmup? A postmortem on a capstone production experience	2016, IEEE Frontiers in Education Conference (FIE)
S73	Students' perceptions of scrum practices	2012, 35th International Convention MIPRO
S74	Teaching advanced software design in team-based project course	2013, 26th International Conference on Software Engineering Education and Training (CSEE T)
S75	Teaching code review management using branch based workflows	2016, 38th International Conference on Software Engineering Companion
S76	Teaching students global software engineering skills using distributed Scrum	2013, 35th International Conference on Software Engineering (ICSE)
S77	Ten years of capstone projects at Okanagan College: A retrospective analysis	2016,21st Western Canadian Conference on Computing Education
S78	The Effect of Real-World Capstone Project in an Acquisition of Soft Skills among Software Engineering Students	2020, 21st Western Canadian Conference on Computing Education
S79	The impact of undergraduate mentorship on student satisfaction and engagement, teamwork performance, and team dysfunction in a software engineering group project	2020, 51st ACM Technical Symposium on Computer Science Education
S80	The real world web: How institutional IT affects the delivery of a capstone web development course	2014, Western Canadian Conference on Computing Education
S81	The value of a real customer in a capstone project	2008, 21st Conference on Software Engineering Education and Training
S82	Towards an ideal software engineering project course	2015, 15th Koli Calling Conference on Computing Education Research
S83	Use of agile methods in software engineering education	2009, Agile Conference
S84	Use of role-play and gamification in a software project course	2017, IEEE Frontiers in Education Conference (FIE)
S85	What can students get from a software engineering capstone course?	2017, 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)

3.2.3 Data extraction and analysis

We developed a classification scheme for the sources based on the RQ1. These were important in mapping the mechanisms used in the capstone courses and therefore providing answers to RQ1. Therefore, after applying the inclusion/exclusion criteria, the following properties were extracted from the remaining 85 studies.

- The title of the study
- Publication venue and year
- Short description of the perceived outcomes of the course, focusing on the learning outcomes
- Key aspects of the course setup: duration and workload, types of projects used, phases of the software cycle gone through in the course, perceived quality of the produced software, clients for the projects, technologies used and the project team composition.

The data was extracted to a common datasheet by the above mentioned aspects. Short descriptions of the perceived course outcomes were also included in the datasheet. Chapter 4 summarizes the results of this mapping and discusses how the mechanisms presented have affected the course outcomes.

3.3 Part II - Case Study

3.3.1 Case selection

The case presented here was selected by convience and personal interest. At the time of writing this research, the author had been working as a software developer at the Software Development Academy for 1.5 years. Therefore describing the basic function of the team, in terms of the number of students, software being developed, technologies used and organization of work could be done without any external sources. The manager and one colleague working at the SDA had also previously indicated interest in refleting the role of the SDA in a university context. As this aligned with interests of the author, the case was selected.

3.3.2 Case introduction

The SDA (toska.dev) was found in 2017 and over these last five years, has undergone various changes to suit the needs of the team and its stakeholders. The framework given here represents SDA as it is in spring 2022 and has been generalized as much as possible. For some aspects of the SDA which are essential to report, but not inherently visible to the author, the managing and founding member of the SDA was interviewed. This includes things such as the recruitment process, client acquisition and funding. These parts have been clearly indicated in the text to separate them from pre-existing knowledge of the author.

Staff

The team size of the SDA is not fixed and varies between 4–10 students depending on the need. By default, each student is given a one-year contract. Rapid circulation of staff ensures that no software is left as a responsibility of a single employee for too long. Knowing that the time at the SDA is limited, forces the students to demonstrate their work more often and pass any relevant information to the remaining team. It also aims to motivate the students to continue their studies to gain further employment after their time at the SDA. Student staff members are all given the same salary at the corresponding experience level.

There is no specific year or phase of studies, which qualifies a student to be hired, instead the competencies of the student are in the center. Few have had pre-existing degrees and several years of work experience behind them, whereas for some, the SDA is their first real job. Some have only recently started their studies, some are in the final steps of their Master's degree. Through their wide responsibilities as a teacher in the Computer Science Department, the manager responsible for the recruitment has a very good overview of the entire student population in the CS programs. They are the responsible teacher for several compulsory and elective courses on software engineering and related technologies. This overview gives the manager a general idea of who might be a potential recruit for the SDA. In addition, the students working in the SDA can propose or second any selection based on their experiences with their fellow students. The recruitment process therefore has remained a fairly straight-forward one. Generally one interview with a potential recruit has been sufficient to determine whether that person gets hired.

During the academic year, all students work at the SDA half-time and are expected to

continue their studies while working. The SDA differs from traditional summer internships in the sense that students are hired throughout the year and normally only one new team member starts at a time. In practice, many times when one student member leaves, a replacement for them is hired. A sufficient overlapping period is also reserved, where the leaving staff member hands over the responsibilities and software they have been working on to the new staff member. This procedure also ensures that the existing staff members are not overwhelmed by the need to guide and mentor a lot of new team members at once during summertime, which is a risk with hiring plenty of summer interns. The spread of starting dates also balances out the proportion of more and less experiences developers in the team at any given moment.

SDA is managed by a single faculty member who simultaenously is a lecturer at the University of Helsinki. Approximately 30% of their monthly work time is allocated to SDA-related issues. Their main responsibilities are supervising the team of students, participating in client meetings, negotiating for the funding and start of any new projects, keeping updated on the status of each project and recruiting new students. The manager is very involved in the daily life of the SDA. In addition to the manager, the SDA also has one faculty member, working partly as a software developer and providing technical continuity for the team. But their main duties are in teaching, which reduces the amount of time that can be allocated to helping the students in the SDA.

Developed software and clients

All projects that the SDA has undertaken are web applications uniquely specific to the University of Helsinki. The size of the user base for the software varies from few dozen faculty members to the entire student and teacher body of the university. Largest software, in terms of its user-base, is a course feedback system used by every teacher and student at the University of Helsinki. Other software include a self-assessment form provided for the studyprograms in the university, an analytics tool for investigating the statistics and demographics of studyprograms and courses, and a tool automizing course completion registrations. All the aforementioned software have gained dedicated users especially from the administrative and planning staff.

The software includes few projects that were handed over for maintenance to the SDA by other teams or faculty members, and several projects which were originally developed at the SDA. The manager actively promotes the team within the university and finds suitable projects for the team to work on. At the moment of writing this, there are three

client organizations with software under development and maintenance. All three are organizations within the University of Helsinki and for two of these, the SDA manages several software.

The number of software has grewn over the years as earlier launches have been successful, and at the moment totals at 9. Out of these, a little over half have been in active development within the past year, meaning that they have had large new features implemented. The remaining four are smaller software in fairly low-key maintenance mode and rarely require attention from the staff. All the software developed in the SDA is open source. Having the source code open to public, gives the student members also possibility to use it as a reference point when applying for jobs after their time at the SDA.

Organization of work

The SDA is divided into several subteams of approximately 2–5 students. Each subteam is responsible for one software in active development phase. There are no working times set in stone for any of the subteams, and they are very self-organized what it comes to the ways of working. Since the working hours are not fixed, the students are able to make their own schedules so that they can attend to any lectures, group tasks or practice sessions they might have in their studies. The entire staff shares one weekly meeting at the end of the week to sum up the past week and also to free-formly share ideas and thoughts.

Currently, the team works both locally in the office located on campus and remotely from home. Each student is free to choose where they wish to work. Instant messaging platforms are in very active use at the SDA for both off- and on-topic conversations. It is worth mentioning that the global COVID-19 pandemic affected significantly the location of work in the SDA. Between spring 2020 and fall 2021, the entire team was forced to work remotely, whereas before the pandemic students worked mostly on campus.

The SDA has no separate account managers to conduct the communication with the clients. While the manager maintains the big picture of the developed software, each staff member is also responsible for communicating with the client on the software they are developing. This includes the whole process of finding out the requirements of the client and creating and presenting a solution. Each project has weekly meetings where the Product Owner from the client-side, the developers of the software and the SDA manager are present. The meetings entail going through the future tasks as well as lately implemented fixes and features.

Technologies

The technology stack used at the SDA has been consolidated over the years. React is used in the frontend and Node.js in the backend. The logic for these choices goes back to the initiation of the SDA in 2017. According to the manager, the combination of React and Node.js was seemingly becoming the "go-to stack" in the web development industry. According to the manager, it seemed like "the world had selected this stack", which made it a natural selection to be the building blocks of the SDA software.

All software is containerized and runs on virtual machines located on servers, which are maintained by the university. Each software uses the same version control system, have similarly configured CI pipelines and employ end-to-end testing. The consolidated stack helps the team in many ways. Firstly, staff members can fairly easily fluctuate between different projects or get a grasp of multiple software at once. Different subteams working on different software can easily share best practices with each other. It also allows the staff to gain knowledge of one commonly used stack fairly deeply, rather than scratching the surface of multiple technologies. The manager also states that at this level of experience, the continuity in the technologies used from project to project is integral. The breadth of skills for junior developers is not wide enough for them to be able to jump from a project to another within one year, if each project is made with different technologies.

3.3.3 Qualitative data from student interviews

In order to gain the student perspective of the SDA, semi-structured theme interviews were conducted. Summarization of the interview process can also be found in Figure 3.1. Students who have previously worked at the SDA had first-hand knowledge of what skills they have gained during their time at the SDA as well as how the SDA has affected their future employment. Thus they were the only ones who could reliably give an assessment of how the SDA has prepared them for the life in software engineering. Interviews were conducted between May 2021 and March 2022. In March 2022, there was a total of 17 alumni student members who all were contacted and out of whom 15 participated in the interviews. The author of this research conducted 8 interviews and a another staff member working at the SDA the remaining 7. Students currently working at the SDA were not considered as interviewees, as they had not yet experienced their full year at the SDA and were not yet recruited elsewhere. Therefore most of the questions reflecting their time at the SDA or how it possibly affected their employment could not have been answered.

The setting of the SDA is quite unique in the sense that it does not fully correspond to any capstone project and neither to any common industry internship. For this reason no readymade questionnaires for conducting the interviews could be found and the questionnaires were designed specifically for the purposes of this research. The questions were purposefully open-ended and wide-scoped to get a general idea of what purposes the SDA serves and how it might differ from what the university has to offer. Third parties from the university and the manager of the SDA were involved in designing the questionnaire. The interviews were conducted in Finnish and lasted between 30 minutes and 2 hours. All the interviews were recorded and transcribed in order to ease up the analysis process. The translated questionnaire can be found in Appendix A.

The interview questions were roughly divided into three parts. The first part covered the background of the interviewees before their time at the SDA. The idea was to gain an idea of the level of their skills and understanding of the industry prior to working at the SDA.

The second part consisted of questions relating to the time at the SDA. The questions covered the responsibilities that the students had at the SDA as well as their perception of the SDA. The second part also included questions in which the interviewees were asked to compare the SDA to project courses. These questions were asked in order to find out not only if there exists any differences between regular project courses and the SDA but also whether the SDA could be replaced by or expanded into a course.

The final part was focused on the time after working at the SDA. The questions inquired about the potential differences between SDA and the interviewees' current place of employment. There were also questions regarding the industry-relevant skills the interviewees had learned both at the SDA and elsewhere at the university.

3.3.4 Characteristics of the interviewees

Table 3.2 summarizes the characteristics of the interviewees. A few of the interviewees had completed studies or degrees in other disciplines prior to starting in the CS program. However, for the purposes of this research, no comparison to studies outside of CS programs was done. In order to keep interviewees' individual answers anonymous, the backgrounds of the interviewees are categorized and presented only as aggregate numbers. For the same reason, any quotations taken from the interviews are not relatable to any specific interviewee or their background.

Table 3.2: Characteristics of the interviewed SDA alumni

	Number of interviewees
Professional software engineering experience in years before the SDA	
None	6
$< 1/2$ year	4
1/2 year - 2 years	3
> 2 years	2
Phase of studies when starting at the SDA	
2nd year Bachelor student	5
3rd year Bachelor student	4
1st year Master's student	4
2nd year Master's student	1
Software engineering capstone project done before the SDA	
Yes	9
No	6
Worked partly or entirely remotely at the SDA, due to the pandemic	
Yes	5
No	10
Had found employment in the industry after the SDA	
Yes	14
No	1*

*One interviewee was not employed at the time of the interview due to their own choice. At the time of writing this thesis, they too were employed in the software engineering industry.

3.3.5 Qualitative data from the faculty member interview

The interview of the faculty member and manager of the SDA was conducted by the author in March 2022. Lower right corner of Figure 3.1 summarizes this interview process and how the manager perspective was included in the study design. As with the questionnaire for the student members, the questions for the faculty member interview were designed specifically for the purposes of this research and were fairly open-ended.

The questionnaire for the faculty member consisted of four parts. The first part had fairly general questions about the technology, recruitment process and client-base of the SDA. The questions were formulated to clarify some aspects of the SDA and how it is run, which were not visible to the author as a regular student member. Second part included questions about the Software Engineering Project Course. The manager of the SDA has

also been the responsible teacher in the said course since 2009, so they were able to provide an unique perspective into the course objectives, its outcomes and its comparability to the SDA. Third set of questions aimed to shed to light on the process of course selection and development at the Computer Science department as well as how the SDA might have affected it. And the final part consisted of questions about the future of the SDA.

3.3.6 Data analysis

Firstly thematic analysis was applied to interviews of student members to gain the key concepts that are evident in the data. Thematic analysis was chosen as it is a commonly used method for describing, analyzing and reporting themes and patterns in data (Alhojailan, 2012). It also provides a systematic element to data analysis and allows the researcher to associate a frequency of a theme recurring in the data (Alhojailan, 2012). Each interview was read, after which each answer was extracted to a common data sheet and organized by question. On the data sheet, the essential sections of each answer were coded, and the main themes were obtained. These themes were viewed as essential in describing the educational aspects of the SDA and its relation to capstone courses. In addition, relevant quotations from the interviews were chosen and translated into English to illustrate the main findings.

The analysis of the faculty member interview was done next. As there was only one interview the process was considerably more light-weighted than that of the student interviews. Firstly, the missing details for the case presentation were extracted. Secondly, the faculty member's perceptions of the differences and similarities of capstone courses and the SDA were extracted along with descriptive quotations.

4 Semi-systematic literature review

This chapter presents the results of the semi-systematic literature review. All sources selected for the review describe ways capstone courses have been organized in their respective institutions. As educators often like to modify their courses over time to find the best ways of teaching, sources here too reflect on the changes done to the courses. To generalize and clarify the results, only the principal aspects of the final course iteration are presented and generalized in Table 4.1.

Some authors have also written multiple articles based on the same capstone course. All these duplicate articles provide important complementary information of the courses. However, in order to keep the statistics clean, each separate capstone course has been categorized only once in Table 4.1. In such cases the most recent article was chosen. The sources left out in this procedure S17, S39, S43, S46, S54, S59, S69 and S75. This leaves a total of 77 sources to be presented in Table 4.1. We will now go through each of these mechanisms presented.

Table 4.1: Aspects of software engineering capstones

	Number	%	Sources by SID
Duration			
Less than one semester	7	9%	S9, S11, S15, S50, S62, S64, S76
One semester	48	62%	S1, S3, S6, S7, S8, S14, S16, S18, S21, S22, S23, S24, S25, S27, S28, S31, S32, S33, S36, S37, S38, S40, S41, S42, S45, S46, S47, S48, S49, S51, S52, S53, S57, S58, S60, S65, S66, S68, S70, S71, S72, S73, S78, S80, S81, S82, S83, S84, S85
Two semesters	21	27%	S2, S4, S5, S10, S12, S13, S19, S20, S26, S30, S34, S35, S44, S55, S56, S59, S61, S63, S67, S74, S77, S79
More than two semesters	1	1%	S29
Clients			
External clients	45	58%	S1, S3, S4, S5, S6, S7, S8, S10, S11, S12, S13, S16, S19, S20, S22, S24, S25, S26, S28, S29, S30, S33, S34, S37, S41, S45, S50, S53, S55, S56, S57, S58, S61, S62, S63, S65, S67, S68, S70, S76, S77, S78, S81, S82, S85
Course staff acts as clients	6	8%	S2, S31, S47, S64, S73, S83
No mention of clients	26	34%	S9, S14, S15, S18, S21, S23, S27, S32, S35, S36, S38, S40, S42, S44, S48, S49, S51, S52, S60, S66, S71, S72, S74, S79, S80, S84
Projects			
Teams work on the same project idea	20	26%	S1, S7, S14, S21, S27, S32, S36, S37, S40, S44, S49, S51, S60, S64, S65, S72, S73, S74, S76, S83
Teams work on separate project ideas	57	74%	S2, S3, S4, S5, S6, S8, S9, S10, S11, S12, S13, S15, S16, S18, S19, S20, S22, S23, S24, S25, S26, S28, S29, S30, S31, S33, S34, S35, S38, S41, S42, S45, S47, S48, S50, S52, S53, S55, S56, S57, S58, S61, S62, S63, S66, S67, S68, S70, S71, S77, S78, S79, S80, S81, S82, S84, S85
Main way of sourcing projects			
Industry or university proposed projects based on a real need	46	60%	S1, S3, S4, S5, S6, S7, S8, S10, S12, S13, S16, S19, S20, S22, S24, S25, S26, S28, S29, S30, S33, S34, S37, S41, S44, S45, S48, S50, S53, S55, S56, S57, S58, S61, S62, S63, S65, S67, S68, S70, S76, S77, S78, S81, S82, S85
Course staff provides project specifications	22	29%	S2, S9, S11, S14, S18, S21, S27, S32, S36, S38, S40, S49, S51, S52 (OSS), S60, S64, S72, S73, S74, S80, S83, S84
Student's generate their own project proposals	9	12%	S15, S23, S31, S35, S42, S47, S66, S71, S79

	Number	%	Sources by SID
Main way for technology selection			
Teams use same technology stack, set by the course staff	19	25%	S1, S7, S25, S27, S31, S32, S36, S37, S40, S44, S49, S60, S64, S65, S72, S73, S74, S76, S80
Technology selections done in teams	46	60%	S2, S3, S4, S6, S11, S12, S15, S16, S18, S20, S22, S23, S24, S26, S28, S29, S30, S31, S33, S35, S38, S41, S45, S46, S47, S48, S50, S52, S53, S55, S56, S57, S58, S61, S62, S66, S70, S71, S77, S78, S79, S81, S82, S83, S84, S85
Not described	11	14%	S5, S8, S9, S10, S13, S14, S19, S21, S34, S42, S51, S67, S68
Assigned mentoring positions*			
Industry experts	7	9%	S14, S38, S44, S52, S63, S72, S84
Faculty or course staff	58	75%	S1, S2, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S16, S19, S21, S22, S23, S24, S25, S26, S28, S29, S30, S31, S32, S33, S34, S37, S38, S40, S41, S45, S47, S48, S49, S50, S51, S53, S55, S56, S57, S58, S61, S62, S63, S64, S67, S68, S70, S72, S73, S74, S76, S77, S78, S82, S83, S85
More experienced students	14	18%	S1, S4, S12, S22, S33, S40, S50, S63, S70, S76, S79, S80, S81, S85
Not described	12	16%	S3, S15, S18, S20, S27, S35, S36, S42, S60, S65, S66, S71
Team sizes*			
1	1	1%	S5
2	5	6%	S2, S49, S60, S66, S77
3	16	21%	S6, S7, S8, S10, S14, S16, S18, S19, S26, S33, S58, S64, S66, S67, S68, S77
4	33	43%	S1, S6, S7, S8, S10, S13, S14, S16, S18, S19, S23, S26, S27, S28, S30, S33, S38, S40, S42, S45, S52, S53, S55, S58, S64, S66, S67, S68, S73, S77, S78, S81, S85
5	33	43%	S1, S9, S10, S15, S16, S18, S24, S25, S26, S27, S28, S30, S31, S35, S37, S38, S40, S42, S45, S51, S52, S53, S55, S62, S63, S65, S66, S77, S78, S82, S83, S84, S85
6	25	32%	S1, S3, S15, S24, S25, S28, S29, S35, S36, S37, S40, S45, S51, S52, S55, S63, S65, S66, S74, S77, S78, S79, S82, S84, S85
7	16	21%	S15, S22, S24, S25, S29, S32, S37, S50, S52, S55, S65, S76, S77, S79, S82, S85
8	10	13%	S22, S24, S41, S50, S52, S55, S65, S70, S76, S77
9	3	4%	S22, S41, S70
10	3	4%	S12, S41, S70
11 or more	3	4%	S12, S56, S72
Not described	13	17%	S4, S11, S20, S21, S34, S44, S47, S48, S57, S61, S67, S71, S80

*For categories that are not mutually exclusive, the percentages do not add up to 100%.

4.1 Duration

The duration of the courses varies between one period and one year. Few sources have given the duration of their course in months or weeks. These durations have been rounded to the nearest amount of semesters in Table 4.1. Courses lasting less than 4 months are in the category “less than one semester“, 4–6 months in the category “one semester“ and anything between 6–10 months in the category “two semesters“.

Clear majority of institutions conduct capstone courses that last one semester. Interestingly, this is in conflict with the ACM/IEEE Curriculum Guidelines for Software Engineering (SE), which state that the capstone project should span a full academic year (ACM/IEEE, 2014). The ACM/IEEE Curriculum Guidelines for Computer Science (CS) do not specify the duration in a similar manner, but emphasize that the project needs to be substantial and in a larger-scale than other projects during studies (ACM/IEEE, 2013). However, the unfortunate reality is that not all curricula can absorb a full-year implementation (S78). The capstone courses often are also very labour-intensive for the teaching staff, with many teams to manage and evaluate throughout the projects (S70). Students might have full- or part-time work which makes the longer courses harder to arrange (S45). Students also perceive two-semester capstone courses laborous (S45, S74), some even the one-semester ones (S26). In order to provide an intensive and realistic experience, many of these courses take up at least half a work-week (S19, S26, S74, S85). This again, might make other courses taken simultaneously suffer (S26), which limits the possibilities for an intensive, year-long capstone.

However, educators who have experiences with both, shorter and longer duration, have shifted to the longer duration since they felt it is not possible to reach the wanted depth in just a few months. S2 describes how they switched to a two-semester capstone as they found the one-semester projects inadequate in skill coverage and depth. S20 have had experiences with one period and three period courses and state that the change to longer version received overwhelmingly positive feedback from all the participating parties. Students were able to gain more hands-on experience in applying new and familiar tools and project management. Additionally, they learned to act when faced with unanticipated events as the teams experienced surpises – regarding both technologies and people – multiple times during the year. Industrial clients received more ambitious and polished products as a result of the course, and the course staff felt that the learning objectives for the course were finally truly met.

4.2 Clients

Almost half of the institutions choose to conduct their capstone courses without clients that are external to the course (see Table 4.1). In these courses, the course staff may act as clients or Product Owners for the projects or alternatively, the student teams work on their own and only report progress regularly to the course staff. S2 explains that they have the instructors playing clients due to the difficulty of finding suitable clients. Being a small program in a rural institution makes the businesses and organizations suited for such collaboration far and scarce. Also the institution's wish to own the intellectual property rights for the developed products puts off potential clients. For institutions, that would have suitable clients available, there is always the upfront investment in time and effort that the course staff has to make to contacting said clients, guiding them through creating project proposals and assigning the students to these projects (S4, S79). S27 aimed to create a course with students from five different technical and non-technical disciplines, such as computer science and business informatics. S27 mentions that they need to be careful in how they organize the course, so that it would suit the needs of all disciplines. Bringing an external client to the mix might not fulfill the learning goals for all students. Some educators also explain how the outcomes of the course are less predictable with multiple external clients. S77 have experienced several cases when the project sponsors did not show up for the bi-weekly meetings with the students. Such client behaviour caused very low motivation in the student teams and some capstone projects projects failed due to client unavailability. S82 made similar observations and stresses the importance of finding committing clients to ensure a good experience for the students.

However, whenever possible, ACM/IEEE Curriculum Guidelines for SE programs recommend having real, external clients other than the course supervisor for the projects (ACM/IEEE, 2014). These clients can be from other units within the university (S3, S33, S62, S68), local businesses (S3, S13, S63, S68, S70, S82) or various non-profit organizations (S3, S6, S7, S9, S33). Graduates of the program who already work in the industry are also a convenient way for finding clients (S22, S68). To mitigate the risks related to unexpected client behavior, S70 proposes first trying out with one reliable customer and deriving several project ideas from them and only then pursuing multicustomer model with differing project ideas.

Working closely with real-world clients has often received overwhelmingly positive feedback from students (S4, S39, S45, S78) and organizing staff alike (S4, S39, S50, S53, S70, S78).

It has been found to increase the motivation and commitment of students, when there is an actual client with a real need behind the project (S4, S45, S53). It has helped to keep the experience more realistic and credible in the students' eyes (S12, S50). Having industry clients improves the students' technical and nontechnical skills and better prepares them for the challenges they will face in the work-life (S70). In a survey conducted by S4, 19 out of 20 students answered "*yes*" to the statement "*It has been great dealing with a real client*". S39 conducted a study on students' attitudes before and after the capstone course. According to S39, it was evident that the students saw first-hand how software engineering is much more than just coding and clearly better understood the importance of teamwork, quality assurance, version control and customer collaboration.

The collaboration has several benefits for the client too. S22 conducted a study to find out reasons for why clients participate in such project courses. The reasons included getting a tailored software product, to research new technologies and, as a clear number one, recruitment. Recruiting students could happen directly from the team or more indirectly by adding visibility among the students as a potential employer. Others have noticed this benefit too, it is not uncommon for students to get hired by the industry partner who sponsored their capstone project (S8, S19, S45, S53, S77). S45 report having at least 60 out of few hundred students gaining full- or part-time job offers based on the capstone project outcomes, in mere few years. Keeping the experience positive also for the clients, might make them come back with further project ideas (S4, S22). This helps to reduce the client acquistion overhead for years to come. Some organizing institutions have even managed to attract more external clients than there are student teams, which has enabled them to collect a small fee from the ones participating in the course (S22, S81).

4.3 Projects

Three main ways for project sourcing were identified based on the literature. As the majority of courses have multiple external clients, the project ideas in these courses mainly are derived from the needs of the customer. In these cases, the organizing staff often performs some pre-screening and scoping in collaboration with the clients, to ensure that the expectations for the projects are realistic and that the project scopes suit the intended learning outcomes (S4, S6, S22, S55, S62, S81). Externally sourced projects should generally not be on the critical path of any external organization, as the course is intended to remain a safe learning place for the students (S4). Some educators also emphasize that

students are not working for these clients, but are in collaboration with them (S55, S81). Thus the projects need to remain such that the students can have a say in the ways the project will be developed.

Out of the sources, 12% mentioned that the students themselves are the main source for project ideas. These educators have realised that students are more motivated if they get to choose the project idea rather than have teachers assigning the projects (S22, S23, S31). According to S31, if the team selects and defines the projects, their level of commitment and excitement to the project rises as the software system grows. At the end of the semester the students have a strong sense of ownership towards the project, rather than feeling that they have just done one additional assignment (S23, S31). However, there are some potential pitfalls with this approach that educators should be aware of. S77 states, that students should not be allowed to bring project ideas from the companies they work at or from their own businesses. S77 have found that it causes a conflict of interests for the student with the proposal and creates an unfair situation for the rest of the team. S23 lets students to form their own teams and generate their own project ideas, but states this might not accurately reflect the situation in the students' future professional lives. If the project idea comes from the team itself, all complexities associated with requirements elicitation and analysis are eliminated (S45) making the experience less realistic. Real-life projects come with challenges relating to contradicting expectations coming from various external and internal stakeholders (S45).

For 29% of the courses, the course staff provides the project spefications. Some educators have assigned the same project idea to all the student teams (S44), or even in some cases all the students work on the exact same project in one team (S72). Having the same project has the benefit of giving the course staff a consistent basis for grading and teaching (S44) and providing technical assistance to the students (S31). In such cases, all teams will need to deal with the same complexities, project management issues and technology demands as in a typically constructed course, which makes the experience more predictable (S44). Having one project idea also opens up the possibility for competition amongst the teams, e.g., which team will create the best design and implementation (S1, S25, S44, S72). It also possibly allows the course to focus more on the quality of the developed software (S1). S1 has experimented with both approaches, having multiple project ideas and having only one project. They have found it more productive and rewarding to focus on doing one project really well rather than juggling multiple projects and obtaining partial results.

Thus letting the students freely choose differing project topics increases motivation and

ownership of the project, but decreases the realism in regards to what students will face in the work-life. A successful way to combat this tradeoff has been combining a multicustomer and multitopic course with students' project preferences (S26). Students for instance list the available topics in order of preference and the teacher assigns students into groups based on their skills and stated preferences (S26, S53, S59). Tools can also be created to manage the increasing workload that comes with diverse projects and teams (S70, S79). These include tools for assessing the knowledge and preferences of students and tools for assigning them to teams (S79). Also semiautomated workflows for the provisioning the course infrastructure have been created (S79).

Projects from all three sourcing categories generally proceed from an idea to a proof-of-concept or a product with few core requirements implemented, but which will not see real production-use during the time of the course (S2, S4, S15, S27, S37, S46, S47, S55, S62, S64, S68, S72). There are some cases where some of the projects have been production-ready at the end of the course, but these too were then handed over to the customer (S1, S6, S28, S39). This practice leaves students without the experience of working with existing products in the maintenance phase of the software lifecycle (S52). Students are also less motivated to work on the project if they know it will never see actual use and the project assignment is artificial (S52). Assigning students to contribute to Free and Open Source Software (FOSS) projects is an emerging approach to remedy these shortcomings. The idea is to allow students to deal with existing codebases, often large and complex, such as the one they will face when working in the industry (S52). S52 have built a large program where 30 Canadian schools provide suitable open source projects for the students to work on. However, they do mention that there are some restrictions on this model. Only selected students, who can demonstrate high motivation, get to work with these projects. As the projects are located scattered around the country, there are travelling costs involved, if the students wish to meet the project mentors.

4.4 Technologies

Not all sources describe the technology choices made in the course, or who gets to make the choice (see Table 4.1). In multicustomer courses, or in other courses with very differing project ideas, the technology choices are made based on the project (S55) and possibly based also on the client's infrastructure (S22). In these cases the course staff does not impose an entirely common technology stack for all the projects. Then again, having

common technologies for all projects is fairly common in cases, where the teachers provide students with the project requirements (S27, S36). In some cases, the evaluation methods focus heavily on the technical implementation and the course graders might for example have sets of tests they like to run on each project to determine the quality (S74). Some educators have the students compete on the same project proposal which makes choosing a common stack justifiable (S25).

On the other hand, having the teams decide on the tools and technologies makes the students explore available options and justify their selections (S53). This not only gives them autonomy but also makes them responsible for their own successes and failures. S53 has witnessed how students take pride in their work when they exceed expectations in the technology selections. For courses that have have differing technologies, this often entails students learning also new ones. In some cases students themselves have said that they have chosen a particular project so that they had the opportunity to learn some skills that otherwise they wouldn't have covered in their degree (S4). There is a value in exposing students to situations where they need to self-learn new skills beyond a structured teaching environment (S4). Students will face situations in their work-life where they need to learn new technologies and students should be encourage to continue professional development even after they graduate (ACM/IEEE, 2014). Even though the majority of technologies would be selected based on the project and client, some sources recommend having some shared infrastructural tools and technologies (S31, S70, S71). Version-control (S71), project management tools (S31) and tools for continuous integration and delivery are examples of these (S70). It has been found to make the evaluation and distribution of projects easier (S70).

4.5 Team sizes

The team sizes vary a great deal, ranging from 1 to 15. Educators have found that in very small groups, e.g., 2–3 students, the teams are unlikely to generate the dynamics and issues that are common in collaborative software development (S23, S31, S33). Such small team size does not present enough of a challenge (S23). Smaller groups are also unable to complete substantial projects in the typical one semester timeframe (S31). Slightly larger teams of 5–6 students have been found to be more successful in this regard (S28). Having very small teams might also be unmaintable in large programs of hundreds or even a thousand students due to the extra organizatorial overhead each team causes (S20).

Going to the other extreme, larger groups with 7 or more students have often found to be facing other kinds of problems, such as, inability to meet all together and other management and coordination issues (S23, S31). “Free-rider“ problem is also reportedly common in larger teams, where it is possible for few students take the bigger responsibility for ensuring the overall success, and the small contribution of others might go unnoticed (S33, S35, S55). S35 studied the gap between student expectations and reality of teamwork in group projects. They found that even in teams of 4 to 5 students limited contributions at the last minute, sporadic attendance in group meetings, or submitting unnecessary or poor quality work by some team members caused major stress and frustration in the team (S35). This has gone to the extent that some students have made pre-emptive plans for future project courses on how they can complete the work alone if necessary (S35). This unlikely is what the course staff intended. The risk for such behavior grows, if the group is large (S33, S59). In larger teams ensuring an equal balance of work and responsibilities requires more attention from the course staff (S55, S59).

The course conducted in S72 presents an outlier case what it comes to team sizes. The course had 15 students working the same game project in one team. The idea was to simulate what large-scale game development in a diverse team feels like and what it takes to create production-quality games. The authors share that their approach was not really successful. In the aftermath of the course, it came up that some students wanted explicit direction while others felt that they wanted more autonomy and control. According to the authors, for the latter group of students, it was clear that they were uncomfortable following the leadership of the vision team and would have preferred to work on project of their own design. However, the authors also mention that getting to work with your own project vision is a very unlikely case for any recent graduate, which is why they did try to come up with such a real-world teamwork scenario.

Majority of educators do seem to opt for the middle ground of team sizes, and have 4 to 6 people working in any single team. This size is perceived as the sweet spot, cancelling out the negatives of the two extremes (S23, S31, S33). Additional measures for combating any non-productive and opportunistic group behavior such as social loafing and free riding has also been proposed. Conducting peer reviews has been proven to mitigate the risk for such behavior (S35, S44, S63). Some periodic monitoring should also be done by the course staff to ensure working team dynamics (S35).

4.6 Mentoring

Many sources specifically mention that the teams should not be left entirely on their own to complete the course project (S2, S4, S6, S31, S44, S70). Just as there are students who are self-motivated and conscientious, there are also students, who without mentoring, lack the skills or motivation they need to succeed in the course (S2). Also to provide consistency across the experiences, some form of supervision and mentoring is crucial (S4). Rows describing assigned mentoring positions in Table 4.1 reflect this. Sources where there are no mention of the teacher, or anyone else, having an active role in how the teams work during the course, fall into the category "Not described". If course staff only passively receives reports of the students progress and evaluates the course outcomes after its completion, these are not the active mentoring we were looking for. Some courses have several types of mentoring present, in which case the source has been listed under each corresponding category in Table 4.1.

The most popular way of mentoring is to have the course staff acting as mentors. The intensity of mentoring between these courses, or even within these courses, vary a great deal. Sometimes course staff provides oversight in a more supervisory role and intervenes the team's work if any conflicts arise or team includes clearly non-contributing students (S2, S4). At the other end, some instructors have weekly meetings with the students where the teachers actively propose solutions and guide the teams with technical and non-techical issues and team dynamics (S2, S4, S56). Some teachers prefer even to practically manage the team (S4). S31 explain that the most successful changes made on the course were those that allowed the course staff to take a more active role in each team. The grades of students improved and the teams were able to complete more functionality to the software products.

Another, often complementary, way of mentoring is to have industry experts occasionally participating in the course. This can be seen as an especially relevant way of mentoring when the course projects are focused around a common theme, for instance the gaming industry (S72). However, finding the correct balance in this type of mentoring, without a client relationship, has sometimes proven to be tricky. The authors of S72 had industry experts from the gaming and software industries participating as advisors on their course. The advisors' feedback on the students game product was mainly positive and encouraging. While the staff took their remarks to mean that the game concept and development for the moment was commendable, the students took the feedback to mean that the prototype

was, as presented, worthy of praise. This presented a dichotomy that never really resolved: students felt that the project was near-complete, whereas the instructors felt that the project was, at best, a rough sketch.

Many sources have noticed the updsides of having more experienced students, outside of the course staff, mentoring the students in the course. These can be for instance students who have completed the project course themselves in the past year (S4, S82). This has been found to benefit both the project implementation and group dynamics: an active coach can for example help students ask clarifying questions of the customer, overcoming a fear of these being stupid and saving days or weeks (S82). Student mentors can also act as external process auditors and document reviewers, ensuring that students are performing as expected on the course (S4). These peer audits have proven to be productive and positive experiences for the reviewer and the team being reviewed (S4).

Forming a team of the final year students with similar skill levels is in accordance with the ACM/IEEE Curriculum Guidelines for SE programs (ACM/IEEE, 2014), but leaves out an integral part of the software developer team experience: junior and senior positions. This discrepancy has been noticed by some educators (S4, S12, S33, S63), who have gone beyond having senior students just as mentors. In their capstones less-experienced students work as junior developers and more-experienced students as senior developers or team leaders. S12 organized their capstone course in a way that students are required to work two course units on the same project, one unit as a junior member and one unit as a senior member. Each unit lasts one period, but the periods do not have to be consecutive to allow some flexibility for students in organizing their studies. In order for such arrangement to work, the projects in the course are large, long-term products, which undergo enhancements over a number of semesters. S12 found that for junior students, this setup allowed a smooth transition to the project, up-skilling on relevant skills and acquiring the necessary orientation from senior students. Senior students, on the other hand, were enthusiastic about mentoring junior students and finding answers to their questions ranging from project requirements to the technology stack. S63 have similarly split their capstone project into two parts with junior and senior positions. They also had faculty mentors with industrial experience mentoring the student teams working with external clients. According to S63, having this course design enabled them to create an effective industrial simulation. S63 reports that students used tools and practices prevalent in industry but frequently not taught in university and were able to develop professional and team working skills more intensively.

5 Case Study

This chapter presents the results of the interviews of student and faculty members involved in the Software Development Academy (SDA). Section 5.1 is based on the student interviews and describes the perceived educational aspects and challenges of the SDA. Section 5.2 is based on the student and faculty interviews and presents the experienced differences between the SDA and the capstone course in the University of Helsinki. Finally, Section 5.3 provides a summary and comparison of the results from the literature review and the case study.

5.1 Educational mechanisms

As the student interviews were semi-structured and many of the questions were somewhat overlapping and open-ended, no single question provided a full description of the educational mechanisms of the SDA. Instead the topics summarized in Table 5.1 were found as a synthesis of all the interview questions. These topics present the aspects with most mentions by the 15 interviewees. Some of these emerged when asked about the pros and cons of the SDA on its own, and others when the SDA was compared to capstone projects. We will go through these topics in the next sections.

Table 5.1: Identified educational mechanisms and challenges in the SDA

	Interviews
Educational mechanisms	
Production-quality software	15
Wide responsibilities and autonomy	15
Employee status and salary	14
Strong community and networking possibilities	13
Easy integration with studies	11
Selectiveness of team members	11
One-year duration	10
Working with external stakeholders	6
Perceived challenges	
Lack of clear seniority	9
Asynchronized working schedules	6

5.1.1 Production-quality software

The most commonly mentioned positive aspect of the SDA is that the software developed goes to production and actual use by end-users. All 15 interviewees made a note of this, especially compared to software projects they had worked on during their studies. Project-based courses commonly include developing software from an idea to a robust prototype, and do not include work in a production environment (see Section 4.3).

According to the interviewees, actual usage of the software brings out many important things. Firstly, it forces students to focus on the quality of their work. Even though the developed software might not have millions of users, it is never-the-less expected to be up and running without interruptions. Thus any changes introduced to the software should not leave it in a broken state. Secondly, it teaches students how to handle and maintain development pipelines and infrastructure resources, keeping in mind that the software is in production use at all times. Thirdly, several interviewees indicated that developing software, which actually benefits the users, gives the students motivation to work on it. Finally, it gives students the ability to say in a job interview, that they have worked on a production-level software for a year.

> "(Developed features) to production as fast as possible. That is something that you don't necessarily get to experience in other environments. And you don't understand the value of that until you witness it first-hand. The long-term nature of the development is important. It is not enough, that the application works for six weeks and then explodes on the eight (week). It adds seriousness to the process."

> "The sense of meaning in the work and releases in the SDA is high. There are actual users who are happy with the updates made in the software."

5.1.2 Wide responsibilities and autonomy

Another thing that all the interviewees mentioned as a positive, is that the students are given wide responsibilities in contrast to narrow and simple tasks. This made students feel that they better understood software development as a whole. In practice, most of the interviewees describe how they started at the SDA by working on simple front-end issues, such as minor bugs or feature requests. From there on they progressed deeper into the

backend of the software, and built larger and more comprehensive features. Building new features often included also refactoring older parts of the codebase and making changes to existing databases. Most interviewees also ended up taking some responsibilities related to the infrastructure that the software runs on. This meant working on the deployment pipelines or setting up software on new virtual machines. Some spent a lot of time integrating the developed software to the systems used in the university. By the end of their year, an average student had had a lead role at least in one software, and also acted as a mentor to new-comers. The wide variety of tasks and areas enabled students to gain highly valuable skills for their future work life.

> "You get a really vast experience from different areas of software development really fast. You get to do everything. And you get to do it almost immediately."

> "It is not that I didn't learn any new skills at the SDA. It is that there are so many skills, I think it is worthless to start listing all of them."

> "Coding. Infrastructure. Designing software architectures. Testing. People skills. Working with colleagues. Trunk-based development. That's about it. All sorts of coding. Infrastrtucture, databases, message systems, end-to-end testing. A really vast set of skills and knowledge."

Those interviewees who currently work in the industry mentioned that taking up on such a high responsibility and a variety of tasks is usually not possible within the timeframe of one year. This is especially true when considering larger companies, teams or software, in which the responsibilities tend to be much more refined and focused. Some interviewees also explained that this kind of variety of tasks, gives the student a possibility to test various things in a working context and only then decide which direction they want to specialize in the future.

> "I could not perform in my current job as a software developer if I hadn't been to the SDA first. I would not survive my tasks. Entering working life right after the Software Engineering Project would have made me cry."

Along with wide responsibilities, a few interviewees specifically mentioned high autonomy regarding software development choices and practices as a positive thing. Being able, and also having the responsibility, to choose the way some feature is implemented gave many

interviewees the feeling of satisfaction and they regarded it as an important motivational aspect. However, it is worth mentioning that the high level of autonomy was not perceived entirely as a positive thing. Some interviewees noted that while having freedom in your work is a nice thing to have, autonomy combined with relatively inexperienced developers might lead to some poor choices or less optimal ways of implementing a feature, tool or configuration.

> "Even though there are some guidelines and rules on how people do things in the SDA, there are so so so much fewer of those things than in some companies. That might show in the way people sometimes make insane choices what it comes to development. For instance, someone might code something that has no use. Or focus on something funny. Which of course is nice and fun, but might not give so much to the product being developed."

5.1.3 Employee status and salary

Naturally, one of the major differences between project courses and the SDA is the salary. Even though the salary is not high, the interviewees report it being one of the main motivators to join the SDA in the first place. With salary, comes also the job contract, which ties the student members to the team on a whole different level than in traditional capstone projects.

> "It is a good aspect of the SDA that you work there and you get money. If the SDA was a course worth of 25 credits without any money, I think the outcome would be very different."

Some find already intrinsic value in the university employee status and reflect on how they tought it was interesting to be on the "inner circle" at the university. They got the official status saying they are working as a developer, got to network and had an office space available at the university. Most mention that they have found the extrinsic value of the employee status, when applying for employment after their time in the SDA. Year worth of industry-relevant work experience using modern technologies gave them a head-start in the job search.

5.1.4 Strong community and networking possibilities

Almost all of the interviewees brought up the community, which has formed at the SDA over the years, as a positive factor. Some explain how they have created real friendships with their collegaues and others how they have enjoyed having like-minded people in the work environment. This not only has made their time at the SDA more enjoyable, but also affected their future employment. In the interviews, 3 out of 15 mention finding employment directly due to the relationships they built at the SDA. Few others have been in job interviews arranged by people at the SDA, even though they did not end up in those positions. One interviewee mentioned that it is nice to know people who are working in the same industry, but not in the same company as they currently are. This gives them a possibility to gain perspective on how others do things in the industry.

> "Sense of community and team spirit is unusually strong at the SDA. It has shown also after I left the SDA. The SDA has a strong community and people care about each other and take care of each other."

5.1.5 Easy integration with studies

One recurring topic in the interviews was the convenience and flexibility of working at the SDA. The location of the office at the campus, right alongside auditoriums and classrooms where the CS courses are held, was perceived very handy. Some interviewees mentioned that it was possible to work a few a hours in between lectures. Not having to work strictly from nine to five was convenient for many. The general atmosphere also encouraged to study, and few found their Master's thesis subject within the SDA.

> "The SDA fits into the studies really well. Studies and the SDA are really easy to integrate into one another. While I was doing the SDA-things, I was also doing the school things. For instance project work could be of both. In that sense, it was really efficient use of my time. I also gained a subject for my thesis from the SDA and peer support for doing it. That is something that was really not available elsewhere."

However, despite the convenience, the SDA being part-time work naturally takes time out of one's weekly schedule. When asked about the progress of their studies during their time at the SDA, 5 out of the 15 said that the work at the SDA was so preoccupying that

their pace of progress slowed down. Two interviewees mention that it is hard to estimate the effect, since they had other factors slowing down their studies, such as the COVID-19 pandemic. Six felt that they progressed as planned. Some of these six were finishing their studies, and felt that the SDA did not have any effect in one way or another. Two mentioned that their time at the SDA boosted their pace in studies as the environment was encouraging.

The ones who progressed as planned, or better, mentioned flexibility with the working hours, the general attitude towards importance of studies in the team, and the close relation to the university community as things that helped them to maintain their pace of progress.

5.1.6 Selectiveness

When asked of negative aspects of the SDA, few raised up exclusivity as an issue. Some felt that it is an unfair advantage that few selected members of the student body are given, and not everyone gets to have such an experience during their studies. Others however raised exclusivity up but in a positive light. Since the students are specifically hired to the SDA, there is a good chance that they are motivated and fit into the environment well. This not only makes the team more productive but also the working experience more enjoyable. With capstone projects, many interviewees felt that since the course is a compulsory one, the groups are often random collections of students which has a negative effect on the overall performance of the team. With the selective recruitment process comes also the risk. As with any company hiring, recruitment to the SDA contains the risk of the wrong person getting hired. That is something some interviewees perceived as a challenge at the SDA.

5.1.7 One-year duration

Capstones and other project-based courses tend to last only one semester, which does not necessarily give students sufficiently deep experience into software engineering (see Section 4.1). This opinion was also shared by many interviewees regarding the Software Engineering Project at the University of Helsinki. The SDA however, lasts for a year, including full-time work in the summer. This provides a more ample ground for students to build their skills.

> "In the SDA you have a lot more time, and it gives you the possibility to dive deeper into the projects. Even though the Software Engineering Project is 200 hours, it still is a scratch on the surface what it comes to software development. After all, you are suppose to fit the whole software lifecycle in that 200 hours."

Some raised up the one year duration as a negative aspect, or a thing that they might want to change about the SDA. But in the sense, that it is too short of a time to experience everything you want.

> "I, of course, understand why students get to spend only one year in the SDA. But still, I feel that the first year in a new job is usually some sort of an orientation period. ... That is how long it takes to make yourself comfortable and acquinted in any team, no matter how good the team is and how well you yourself might fit into it. ... I still think that the second year is a lot more productive for most"

5.1.8 Working with external stakeholders

Especially out of those students, who had previously not done project-work with clients or end-users, many felt that their project management, client negotiation and communication skills grew at the SDA. Some of these interviewees also felt that they would not have gotten the same experience what it comes to clients as juniors in the software engineering industry. In such environments there might have been project managers or account managers handling the communication and planning.

> "I learned a lot about project management. I learned how to evaluate people and tasks and how to fit those two together. I learned to appreciate the expertise that other people have. I gained a lot of experience in working with people. I learned how communicate and conduct myself with clients."

5.1.9 Challenges and further improvement

When analyzing the interview data, it became clear that working at the SDA is regarded as a highly positive experience. Most of the interviewees struggled with finding any clear negative aspects of the SDA, apart from the relatively small salary compared to industry averages.

> "I can't come up with anything really negative."

> "I really can not think of any bad aspects"

> "This is again hard when I should come up with something negative. Let's see. I can not think of anything negative compared to project courses when I think about it."

And moreover, some aspect which one interviewee would change, might have been the best thing about the SDA to another. High autonomy was the aspect that most clearly divided opinions.

> On Q13: *How would you change the SDA?*
> "Of course, if I were given the chance to dictatorially decide how we are gonna do things at the SDA, I would start using TDD and pair programming. ... Everything would go through pull requests and code reviews as well. I mean, *if I got the choice*, I would enforce these quite strict quality control measures."

> On Q5: *What positive aspects do you think the SDA has?*
> "It is really free. You get to fly solo. You have some guidelines within you are free to go solo. You get to choose which technology or library to use for the feature and how to implement it. Not a lot of strict practices."

Even though no common and strictly negative aspects can be found based on the interviewees, there are few challenges and areas at the SDA that could be further improved. The most commonly mentioned challenge relates to the experience levels in the group. SDA is comprised of students with generally little work experience in software development. On one hand, this was regarded as a highly positive thing, it meant finding friendships within the work community was easier, when the other workers felt more like peers. Having co-workers of the same age group with same areas of interest also enhanced the feeling of relatedness and created a distinct culture to the workplace. But on the other hand, similar backgrounds causes a certain lack of clear seniority in the group, especially compared to other work environments. In a more typical work environment, there would be senior staff who might have already been in the industry for several years, who then could mentor the junior developers what it comes to work practices, tool selection and even coding patterns.

> "The good thing about the SDA is that you have a lot of responsibility and you need to do actual things. The thing you don't get that much, which you might get in other jobs, is that there is someone truly experienced and skilled senior developer guiding you. That experience can be really educational. So even though you have those "senior" coders in the SDA-model, it is not the same as you have someone who has worked in the industry for 15 years. So you are missing that kind of an experience in the SDA."

According to the interviewees, this lack of mentorship in some cases meant that one could carry on doing a bad practice for a long time, without anyone necessarily noticing. Not having assigned mentors and processes in place for mentoring, also leaves more responsibility of the progress to the hands of the worker themselves. The interviewees mention that one could always easily ask for help from the more senior members of the team. But especially when working remotely, the threshold for asking help is always higher than threshold for accepting mentoring, when a skilled senior offers it.

Some suggestions were made on how to tackle this issue, first one being pro-longing the contracts to last more than one year. Another possible solution to this would be having more processes or structure within the ways of working, for instance, starting code reviews. Some interviewees mention that restricting the autonomy and freedom of a single student, might increase the way students learn from each other in the team.

Other negative aspects mentioned by the interviewees are more clearly related to the global COVID-19 pandemic. Many negative aspects can quite clearly seen to be caused or at least amplified due to being forced to work fully remotely. One interviewee explained how their time before the pandemic was great, but after moving to full remote work, they felt that the peer support and joy of working in a team decreased. Similarly, another one mentions that the communication and brain-storming within the team was lacking, but a big part of it can be pinned on "the times we are living in".

Those, whose SDA-year was in its entirety done at the office, mention teamwork being as one of the key things they learned from the experience. They also see it is a thing that worked well at the SDA. But the ones who worked only remotely, did not see teamwork in such a positive light. Or if not negative, it at least was missing from the list of "on top of your head" skills that they learned. Despite the fact, that these students felt that the community, friendships and networks they have gained through working in the SDA were extremely valuable and positive, they still were left with the feeling that something was missing when working only remotely.

> "The fact that you get to work whenever you want is easier and more flexible, but on the other hand you don't get the benefits of real teamwork. In my current job, I talk and work with people on a daily basis. But since so many at the SDA has the 50% job contract, the work gets pretty async at times. Someone does something, and then you get a message from them, and then you reply when you work and so on. And then we have some weekly meeting. So that is different compared to my current job."

5.2 Could courses replace the SDA?

This section discusses whether the model could be replaceable by an university-level course or courses. The evaluation is done based on both, the student interviews and manager interview.

5.2.1 Experience compared to capstone courses

Most of the interviewees had completed the Software Engineering Project (SEP) capstone course during their studies in the CS Bachelor Program (see Table 3.2). From the students' perspective, the similarity of the capstone and SDA experiences vary a lot. The ones who have had an external client from another department in the university perceived the work as being fairly similar. In their capstone projects they too got work for a real client from the university and develop software for the university. Some even worked with the same technologies and in the same office space during their capstone as they did at the SDA. For these students the transition from project-work to the SDA was fairly small.

Many however consider the difference between the capstone project and the SDA as night and day. The considerably longer time they spent working at the SDA allowed them to fully delve into the project topics and work on implementing fairly large systems. They also felt a lot more responsible for their work in the SDA, due to the products being in actual production use by end-users. They refer capstone projects rarely making it beyond the proof-of-concept phase and rarely getting integrated into the clients' systems. Many of these interviewees make a clear distinction between the other one being a school project and the other one being real work, and add that there is a certain level of professionalism at the SDA that they did not experience while completing the capstone project course.

> "Projects are just projects. I never regarded the SDA as a (school) project, since the SDA is a place of work, and those two are hard to compare. Projects give you experience, but I wouln't put any of my school projects into my CV. I can, however, put the SDA into the CV, since at the SDA people develop software with a much wider scope and they do it for actual users and clients."

> "The SDA is the only place in the university where you get to work with such projects, that match working in the industry"

Out of the interviewees, a group of their own are the six who have been credited the capstone course due to their participation in the SDA. Interestingly, those students generally place a much higher value on the project course, than those who had completed it. Those who had not completed it themselves, speculated that it might have been a comparable experience to the SDA in terms of realism, client experience and skills it teaches. Those who had completed both the capstone course and the SDA, regarded the SDA a more valuable experience, that could not be replaced in its entirety by the capstone course.

In the faculty's eyes, the two are also entirely different experiences. The manager states that the SEP almost "runs on its own", now that the upfront investments in time and effort have been made. The manager has coded tools which help to automate the process of team formulation in the course. The course has skilled senior students as mentors for the projects, who have the responsibility then again to propose skilled students to be the mentors for the next year's projects. The role of a responsible teacher is mainly managerial at this point. According to the manager, managing the SDA is much more work-intensive. The manager is much more hands-on in everything that happens at the SDA from coding to pitching new software products in university meetings. The recruitment also takes up a considerably bigger chunk of time, as the manager is very invested in every recruitment decision.

> Q10. *What differences do you think leading the SDA and the software engineering project course has?*
> "It is really different. There really are no true similarities."

5.2.2 Replaceability with other studies

To Question 11: *What kind of a course could provide what the SDA provides?* 9 out of 15 interviewees stated that they cannot see a course could offer the same experience that

the SDA does. The long duration and the skills and dedication it takes to work at the SDA, were seen as some of the most severe hindrances for replacing the SDA with a course. Students' commitment has a key role in the success of the SDA and the commitment would be hard to replicate in a course environment. The interviewees mentioned that the course would have to last at least a year, including summer, which is hard to achieve without paying the students. Four interviewees hesitated, but ended up saying that there might be a chance to build something similar, if not quite like it. Those four mainly focused on the skills that the SDA provides, and on how they could be taught via project-based courses.

When asked about how many credits do they perceive that their year at the SDA is worth, the interviewees' answers ranged from 15-120, averaging around 53 credits. For reference, in the European Credit Transfer System, 1 ECTS credit amounts to approximately 27.5 hours of studying, and a full year is 60 ECTS credits. And even still, many interviewees felt, that the salary from the SDA was so important, that it cannot be replaced with course credits.

Another major difference, which is not easily replicable, is the employee status and real work experience that the students gain when starting to work at the SDA. The student interviewees also noted that finding skilled and motivated managers who can guide the students, amidst all their own teaching and researching responsibilities, would became an obstacle for creating such a course.

> "It requires quite a lot of skills from the responsible teacher. The one who coaches and guides the developer team. They ought to be really good."

> "Compulsory internship where you go to a real firm. It should be a real work environment to get the same that the SDA provides. Even though the software engineering project course tries to mimic it in some ways, people just aren't as invested in those projects like you should be in work life. People come and go as they please and it is really different in my experience."

The manager strongly aligned with these views and according to them, there is no course that could offer the same as the SDA. The manager states that the selective recruitment process, and the fact that students do real work in exchange for a salary creates a community and an experience that are not possible to recreate in a compulsory course.

5.3 Summary and comparison of the results

This section provides a summary of the results and compares the mechanisms used in capstone projects to the ones found in an university-lead internal software startup. The summary presented in Table 5.2 does not attempt to find an exhaustive mapping between the two, but rather seeks to find the most relevant mechanisms that were discussed and then present what the literature review and case study revealed about the matter.

Table 5.2: Main aspects of capstone projects and a university-lead internal software startup (SDA)

Feature	Traditional capstone	Internal startup (SDA)
Duration	Few months to one year	1-2 years
Intensity	Few hours to 40 hours	8 months half-time, 4 months full-time
Salary	No	Yes
Employee status	No	Yes
Clients	Little over half of the courses have external clients from the industry or from other units in the university	External clients from other units in the university
Project sourcing	Industry-proposed, teacher-generated or student-proposed	Manager finds suitable projects that are based on a real need
Software phases	Mostly greenfield coding: from idea to a robust proof-of-concept	Existing projects in production use
Team size	Varies from 1–15 students per team, generally 4–6 students	4–10 students and one manager
Team composition	One team of students working on one project, generally no interaction with other teams	Several, partially overlapping subteams for different projects
Mentoring available	Available mostly from faculty, sometimes from industry or more experienced students.	Available from faculty and more experienced students
Student selection	Generally students with certain prerequisite courses completed or of certain class in their studies	Selective recruitment process
Technologies used	Varies a lot, some have common stack, some don't	Common stack across projects

6 Discussion

In this thesis, the main objective was to understand how universities and other tertiary education institutions prepare their students for the life in software engineering. Section 6.1 will briefly describe the findings for each of the three research questions while discussing on the answers found. Section 6.2 discusses the threats to validity in this research and how they have been mitigated. Related work is presented in Section 6.3, with examples of software development teams with similar organization or results as the presented case study of an internal software startup in a university-context.

6.1 Answering the research questions

Three research questions were formulated that guided this research. We will next go through these questions and discuss the answers found for each one.

RQ1: What mechanisms traditional capstone courses use to prepare students for the software engineering industry?

The first research question was approached by going through conference proceedings and journal articles regarding software engineering capstone courses. Based on the literature review, seven different aspects were identified and all the courses were categorized along these aspects. These aspects represent the mechanisms that educators use in their capstone courses to simulate work-life in the software engineering industry.

In more than half of the capstone courses, students produce software systems for external clients from local businesses, non-profit organizations or from other units in the university. In these courses, the project ideas are also produced by the client themselves and further refined in collaboration with the teachers. Some educators choose to conduct the courses without external clients. In those cases students work either from project ideas crafted by the course staff or students themselves generate the project proposals. Even though students might be highly motivated to work on their own ideas, or having a common project outlined might be convenient and consistent for the course staff, using external clients was generally perceived as the best practice. Completing a software project based on the real needs of someone else gives the students a chance to practice their project

management and communication skills with external stakeholders. Students will likely face ambiguous problems of non-technical people in their worklife, which makes these capstone experiences more closely resemble real life.

In software engineering related programs capstone courses should span the whole academic year (ACM/IEEE, 2014) or otherwise be substantially larger than what the students have experienced during their earlier studies (ACM/IEEE, 2013). However, we discovered that the average duration for capstone courses is one semester. Real-world constraints of cramped curriculas, finite teaching resources and students' commitments to work and other studies were found to be the major reasons for this. Those institutions who had implemented a two-semester course found it highly beneficial for the students. It enabled students to dive much deeper into the projects and new technologies and learn about the long-term nature of software development.

Mentoring and coaching students especially related to work practices and project management related issues are generally applied to these projects. In most cases, the course staff are the ones doing the mentoring, while some utilize industry mentors or more experienced students. Student team sizes vary from single person endeavours to the whole course of up to 15 students participating in a common project. Most educators had found that the sweet spot lies between 4–6 students. Such team size enables students to resolve communication issues as they would in real life teams but the risk for free-riding on others' work diminishes.

As capstones are simulations by nature, students rarely get to work with large software in production-use, which again is a likely situation in the software engineering industry. The projects typically progress from an idea to a robust proof-of-concept. Assigning students to work with large and complex Free Open Source Software (FOSS) was proposed as a potential remedy for this discrepancy. However, only one source within the literature review was found using this methodology (Holmes et al., 2014), which would indicate that using FOSS projects is not common in the software engineering education.

RQ2: What mechanisms does a university-lead internal software startup use to prepare students for the software engineering industry?

Along with the second research question, this research proposed a novel framework of a university-lead internal software startup. The framework was presented through a case study, Software Development Academy (SDA). SDA is a team of students, lead by faculty that develops and maintains administrative educational software for the use of the entire University of Helsinki. We also gathered the experiences of SDA alumni currently working

in the software engineering industry to answer the presented research question. We quickly saw that the students found the experience highly relevant and it prepared them for the life in software engineering well. Students got to spend an entire year developing production-quality software, which was regarded as a truly educational experience. Having wide responsibilities ranging from user interface fine-tuning all the way to leading a small software project team gave them a much wider skillset for worklife that they could have gotten from elsewhere in the university. The strong community and networking possibilites provided by the SDA had enabled some of them use it as a stepping stone for finding their current jobs. While no clear negative aspects of the SDA were identified, the lack of clear developer seniority and sometimes asynchronous communication resulting from remote work, had created some challenges for the work at the SDA.

RQ3: How do traditional capstone courses and a university-lead internal software startup differ?

With the third research question we aimed to see, if there are any relevant differences between the more traditional capstone courses and the novel framework of an internal software startup. It was clear that both approaches benefit the students and have their place in the university context. Capstone courses are often mandatory experiences, and all students in a certain phase of their studies complete such projects. Therefore they serve the whole student body of the program. The SDA has a selective requirement process which aims to ensure that the incoming students are motivated and their technical competencies fit the work environment well. Due to its restrictive nature, only some students get to experience working at the SDA. This is were some of the biggest differences between the two approaches lie. At the SDA, a handful of students develop production-quality software with an employee status in exchange for a salary, whereas in capstones, the entire class goes through a simulation of what developing software in a professional team is. Due to these differences, both students and faculty at the SDA concurred that no course can, or should, replace the SDA, and that the SDA is a valuable experience on its own. The aspects that separate the SDA of the project-based courses most, ironically, are also the hardest to replicate in a course environment.

6.2 Validity

The fact that the interviewers were working at the SDA during the time of the interviews, and had also worked as colleagues for some of the interviewees, may have had an impact

on how the interviewees responded to the interview questions. For instance, when asking about the negative aspects of the SDA, they may have replied less harshly than if the interviewer was an unknown third party. This bias has been somewhat diminished by the fact, that third parties were involved in generating the interview questions for the student interview. In addition, negative aspects of the SDA were specifically asked on several occassions to get a full view of the matter. The relatively large collection of 15 interviews also decreases the risk of answers being one-sided. Thus it is reasonable to believe that the negative aspects of the SDA were also at least somewhat captured.

Faculty perspective was gained by interviewing the manager, who at the time of writing the research was also the author's immediate superior. As there are no other managers within the Software Development Academy, the faculty member is also more identifiable and true anonymity of the answers is not possible achieve. The manager has a known vested interest in the SDA. Similarly as with the student members, these conditions might lead to overly positive answers. To somewhat diminish the risk of biased results, the manager was asked fairly neutral questions relating to the past of the SDA and its management. There were less questions reflecting on the experiences and opinions of the manager.

Due to her employment at the Software Development Academy, the author also has an acknowledged personal bias towards thinking that participating in the SDA gives a relevant experience to any student member and that the SDA in general is a good place to work at. This is to certain extent mitigated by the fact, that 16 other people were interviewed for the purposes of this research. And as stated above, negative aspects were asked on multiple occasions. Additionally, studies where educators themselves report their experiences and findings on the education they provide, are common practice in the field of computer science. This is indicated for instance in the 85 case studies the author found in the literature review part of the research. In this research the objectivity is somewhat increased by the fact that it is not the founder of the SDA, neither the author, themselves reporting their experiences. Working at the SDA also gives the author great perspective for assessing the interview data and describing the framework.

The sources selected for the semi-systematic literature review do not provide an exhaustive outlook on the industry-relevant education in computer science. They only describe the ways capstone courses are provided in different tertiary education institutions and only represent the capstone courses that have some aspects or outcomes worthy of a publication. The inclusion and exclusion criteria for sources were generated to give a wide enough perspective but also keep the workload manageable. The capstone courses are presented

to discuss the ways that the Software Development Academy differs from the general way of preparing students for the sofware engineering industry.

6.3 Related work

The author could find very little evidence that student-based internal software startups are a common occurence in the university context. Never-the-less, there are cases with some organisatorial aspects or project results comparable to those of the Software Development Academy. This section presents those cases to provide context on how similar work has been done elsewhere.

Ding et al. (2017) present a software engineering capstone project, where students worked on a large university-owned project, a Class Attendance Tracking System (CATS). As the name suggess, the software was designed to help track the class attendance of university students. The project resembles those of the SDA in the sense, that it is developed and maintained over a long period of time, instead of being a single-shot proof-of-concept by one capstone team. The product owner of CATS is similarly another organization in the same institution: the staff from academic enrichment department. Ding et al. state that the product owners were eager to participate in the process of developing the software as it was designed specifically for their needs. Similarly to the projects of the Software Development Academy's projects, CATS is a successful example of a software with product owners coming from the university and which ended up being released to more than ten thousand students. The authors also mention that a real and big project such as CATS is very attractive to students. Students in their team found it helpful to be able to put such skills and work in their resumes, and ended up finding similar job positions in the industry. Ding et al. thus were also able to witness how using industry-preferred technology in a real university-lead project had highly positive effects on the employability of the students.

Buffardi et al. (2017) have also implemented tech startups into their programme with the aim of providing realistic experience in software development and to cultivate development of real software products. They combined the efforts of Computer Science and Entrepreneurship programs to give the students more realistic setting for software development. In their model, the entrepreneuship students acted as business clients and the computer science students were responsible for the agile development of the software product. The authors collected the experiences of the computer science students working in this manner (Industry) and compared them with the experiences of students who

completed the course without having business students as external clients (Software Engineering -only). Their initial findings are somewhat contradictory to the general perception of how external clients affect student motivation, in that in the Software Engineering -only projects the students worked more hours per week on the project than those on Industry projects. However, they anecdotally observed that students without external clients for their projects, demonstrated a weaker sense of direction and accountability. Moreover, those students did not benefit from the opportunities to professionally network and develop their soft skills through interacting with clients.

Williams et al. (2021) report having a very similar software development team as the SDA, at Berea College, KY, US. The idea for founding Student Software Development Team (SSDT) rose from the wish to have software matching their needs, without running into the prohibitive costs that acquiring and maintaining custom-made software from external software houses would have. They also realized that their version of the SDA would provide valuable skill-building experience for students, making them more employable by the software industry. By the time of their report, SSDT had run for six years and created nine software systems for the purposes of their institution, which marks up to similar kind of progress done by the SDA. In SSDT the work is organized similarly as is done at the SDA, in that during summers students work full-time and during the academic terms only part-time. However, in their model, summers are reserved for larger changes and features in the software and the academic terms are used solely for maintenance, bug fixes and fulfilling small feature requests. In the SDA, the nature of work stays the same throughout the year, only the velocity of development goes down during the academic term. Similarly to experience gained from the SDA, Williams et al. also report the maintenance phase being extremely beneficial for students. They emphasize that maintenance of production-quality software after deployment is a skill rarely taught in software engineering courses. In their experiences capstone projects and similar are “completed“ and delivered to customers, with no students involvement in the maintenance. Their report has been written very recently, in 2021. They also mention finding no other institutions hiring students to develop custom software solutions, including maintaining the software throughout its lifetime. This enabled them to claim their framework to be a novel one.

7 Conclusions

This chapter concludes the research and summarizes the findings done on empirical interview data as well as experiences of other academic institutions. This research aimed to understand how tertiary education prepares computer science students for the work-life in the software engineering industry. Firstly, we looked into how traditional capstone project courses do this based on 85 primary sources. The main findings indicated that the duration of capstones ranges from one period to one year, average duration being one semester. Projects for the courses can be based on students' own ideas, specifications set by the course staff or come from external clients represented by businesses and other organizations. Teams can be formed in various ways but typically include 4–6 students of the same year level while course staff act as mentors for the teams. Projects generally progress from an idea to a proof-of-concept or a prototype.

Secondly we looked into a more novel way of preparing students for the software engineering industry: a university-lead internal software startup. This approach was presented via a case study, Software Development Academy (SDA), run in the University of Helsinki. SDA is a software development team comprised of students and lead by faculty that develops and maintains educational administrative applications used in the University of Helsinki. A framework for this approach was presented along with the experiences of students and faculty involved in it. The startup approach was generally perceived as highly beneficial by the 15 alumni interviewed. They perceived working with production-quality software, with actual users and clients and having wide responsibilities in the team integral in giving them a wide skill set for their future worklife. There were no clear negative sides identified in the SDA but few challenges remained.

Finally, the findings of the approaches were compared. The main factors separating the two were the longer duration of the SDA experience, the selectiveness regarding team members, the quality of the software produced and employee status gained in the SDA. The interviews revealed that students generally regarded the SDA being much closer to the work-life in the industry than the capstone course. Students generally could not see how the experience could be replaced by a university course. Also the faculty perspective was gained on the matter by interviewing the SDA manager and teacher of the capstone course. They felt that both approaches are important, but shared the idea that no course

could or should replace the SDA.

What is missing from this report, is the client's perspective. Any future research could report on the client's perspective, especially on how they perceive ordering software from a student-based startup and what potential benefits and pitfalls exist in the process. From their own experiences, the author also knows that the SDA has had a notable impact on the course selection in CS programs at the University of Helsinki. This was not included in the faculty perspective in this research, as we focused more on the differences between traditional capstone courses and the SDA. Any further reseach could therefore deepen the faculty view on the matter and also further explain the role of the SDA in the curriculum development process.

The performance of the graduates who have gone through the SDA could be measured. It would be also interesting to see, how this framework could be either replicated or scaled. For instance how it could cover a larger portion of each class of the Computer Science and Software Engineering programs or possibly even extending it to other disciplines. In its current form, only a handful of students each year get to enjoy the SDA experience and it clearly could benefit many more.

Bibliography

ACM/IEEE (2013). *ACM/IEEE Joint Task Force on Computing Curricula: Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. URL: https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf%20(visited%20on%204/20/2022).

– (2014). *ACM/IEEE Joint Task Force on Computing Curricula: Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. URL: https://ieeecs-media.computer.org/assets/pdf/se2014.pdf (visited on 04/20/2022).

Adams, R. and Kleiner, C. (2016). "Collaboration Support in an International Computer Science Capstone Course". In: *International Conference on Social Computing and Social Media*. Springer, pp. 313–323.

Alhojailan, M. I. (2012). "Thematic analysis: A critical review of its process and evaluation". In: *West east journal of social sciences* 1.1, pp. 39–47.

Begel, A. and Simon, B. (2008). "Novice software developers, all over again". In: *Proceedings of the fourth international workshop on computing education research*, pp. 3–14.

Bowring, J. and Burke, Q. (2016). "Shaping software engineering curricula using open source communities". In: *Journal of Interactive Learning Research* 27.1, pp. 5–26.

Buffardi, K., Robb, C., and Rahn, D. (2017). "Tech startups: realistic software engineering projects with interdisciplinary collaboration". In: *Journal of Computing Sciences in Colleges* 32.4, pp. 93–98.

Delgado, D., Velasco, A., Aponte, J., and Marcus, A. (2017). "Evolving a project-based software engineering course: A case study". In: *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, pp. 77–86.

Dupuis, R., Champagne, R., April, A., and Séguin, N. (2010). "Experiments with adding to the experience that can be acquired from software courses". In: *2010 Seventh International Conference on the Quality of Information and Communications Technology*. IEEE, pp. 1–6.

Fornaro, R. J., Heil, M. R., and Tharp, A. L. (2007). "Reflections on 10 years of sponsored senior design projects: Students win–clients win!" In: *Journal of Systems and Software* 80.8, pp. 1209–1216.

Garousi, V., Giray, G., Tuzun, E., Catal, C., and Felderer, M. (2019). "Closing the gap between software engineering education and industrial needs". In: *IEEE Software* 37.2, pp. 68–77.

Haddad, H. M. (2013). "One-Semester CS Capstone: A 40-60 Teaching Approach". In: *2013 10th International Conference on Information Technology: New Generations*. IEEE, pp. 97–102.

Holmes, R., Craig, M., Reid, K., and Stroulia, E. (2014). "Lessons learned managing distributed software engineering courses". In: *Companion Proceedings of the 36th International Conference on Software Engineering*, pp. 321–324.

Iacob, C. and Faily, S. (2019). "Exploring the gap between the student expectations and the reality of teamwork in undergraduate software engineering group projects". In: *Journal of systems and software* 157, p. 110393.

Ikonen, M. and Kurhila, J. (2009). "Discovering high-impact success factors in capstone software projects". In: *Proceedings of the 10th ACM conference on SIG-information technology education*, pp. 235–244.

Isomöttönen, V. and Kärkkäinen, T. (2008). "The value of a real customer in a capstone project". In: *2008 21st Conference on Software Engineering Education and Training*. IEEE, pp. 85–92.

Johns-Boast, L. and Flint, S. (2013). "Simulating industry: An innovative software engineering capstone design course". In: *2013 IEEE Frontiers in Education Conference (FIE)*. IEEE, pp. 1782–1788.

Keogh, K., Sterling, L., and Venables, A. T. (2007). "A scalable and portable structure or conducting successful year-long undergraduate software team projects". In: *Journal of Information Technology Education: Research* 6.1, pp. 515–540.

Mahnic, V. (2011). "A capstone course on agile software development using scrum". In: *IEEE Transactions on Education* 55.1, pp. 99–106.

Majanoja, A.-M. and Vasankari, T. (2018). "Reflections on Teaching Software Engineering Capstone Course." In: *CSEDU (2)*, pp. 68–77.

Marques, M., Ochoa, S. F., Bastarrica, M. C., and Gutierrez, F. J. (2017). "Enhancing the student learning experience in software engineering project courses". In: *IEEE Transactions on Education* 61.1, pp. 63–73.

Paasivaara, M., Vanhanen, J., and Lassenius, C. (2019). "Collaborating with industrial customers in a capstone project course: the customers' perspective". In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, pp. 12–22.

Panicker, R. C., Sasidhar, S., Jien, S. Y., and Tan, C. K.-Y. (2020). "Exposing Students to a State-of-the-art Problem Through a Capstone Project". In: *2020 IEEE Frontiers in Education Conference (FIE)*. IEEE, pp. 1–8.

Radermacher, A. and Walia, G. (2013). "Gaps between industry expectations and the abilities of graduates". In: *Proceeding of the 44th ACM technical symposium on Computer science education*, pp. 525–530.

Radermacher, A., Walia, G., and Knudson, D. (2014). "Investigating the skill gap between graduating students and industry expectations". In: *Companion Proceedings of the 36th international conference on software engineering*, pp. 291–300.

Schneider, J.-G., Eklund, P. W., Lee, K., Chen, F., Cain, A., and Abdelrazek, M. (2020). "Adopting industry agile practices in large-scale capstone education". In: *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, pp. 119–129.

Snyder, H. (2019). "Literature review as a research methodology: An overview and guidelines". In: *Journal of business research* 104, pp. 333–339.

Spichkova, M. (2019). "Industry-oriented project-based learning of software engineering". In: *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, pp. 51–60.

Stevens, M. and Norman, R. (2016). "Industry expectations of soft skills in IT graduates: a regional survey". In: *Proceedings of the Australasian Computer Science Week Multiconference*, pp. 1–9.

Venson, E., Figueiredo, R., Silva, W., and Ribeiro, L. C. (2016). "Academy-industry collaboration and the effects of the involvement of undergraduate students in real world activities". In: *2016 IEEE Frontiers in Education Conference (FIE)*. IEEE, pp. 1–8.

Watkins, K. Z. and Barnes, T. (2010). "Competitive and agile software engineering education". In: *Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon)*. IEEE, pp. 111–114.

Ziv, H. and Patil, S. (2010). "Capstone Project: From Software Engineering to "Informatics"". In: *2010 23rd IEEE Conference on Software Engineering Education and Training*. IEEE, pp. 185–188.

Appendix A Interview questions - student perspective

1. How did you end up working in the SDA?
2. How did your studies proceed before your time in the SDA?
3. How did your studies proceed during your time in the SDA?
4. How many years of software engineering experience did you have before starting at the SDA?
5. What positive aspects do you think the SDA has?

   5.1. What about negative aspects?
6. What kinds of tasks did you have while working in the SDA?
7. How many credits do you think your time at the SDA was worth? ie. What would be an approriate amount of credits to cover the SDA experience?

   Let's compare the SDA to project-based courses or those courses that attempt to simulate working life, for instance the Software Engineering Project course.
8. How was your experience of the SDA in comparison to your experience of project-based courses?
9. What positive and negative aspects do you think the SDA has, compared to project-based courses?
10. How much responsibility do you think working in the SDA has, compared to working in project-based courses?
11. What kind of a project course could offer the same as the SDA?
12. What other differences do you feel that the SDA has, compared to project-based courses?
13. How would you change the SDA?
14. What useful skills did you learn while working at the SDA?

15. What skills did you learn during project courses, which have been proven to be useful in your work-life?

16. What skills did you learn during project courses, which have been proven to be useful in your work-life?

    16.1 Which skills were such, that you could not have gotten them from elsewhere in the university?

17. How did the SDA affect your employment?

18. How does your current work-life differ from your work-life in the SDA?

19. How did the traditional project-based courses affect your employment?

    19.1 What other useful courses can you think of, which have taught you skills beneficial in your work-life? what skills did you learn through them?

20. How did the nature of work change during your year in the SDA?

21. Who could answer questions about you in the work-life?

22. Can we connect your answers to the data of your course completions?

Appendix B Interview questions - faculty perspective

1. How have the technologies used in the SDA been selected?
2. Which organizations have officially been clients of the SDA?
3. Which aspects affect the recruitment decision of someone?
4. How long have you been the responsible teacher of the Software Engineering Project course?
5. How many assistants does the Software Engineering Project course have per year?
6. What are the main learning objectives in the Software Engineering Project course?
7. What kind of a percentage out of the projects in the course goes into production and is used by end-users?
8. What kind of percentage out of the projects in the course is developed for several academic terms and by several teams?
9. Has the Software Engineering Project course undergone any changes since 2017? *(ie. during the time the SDA has existed)*
10. What differences do you think leading the SDA and the Software Engineering Project course has?
11. How do you perceive the work load caused by the Software Engineering Project course compared to the workload caused by the SDA?
12. What have you learned due to your work at the SDA?
13. What things affect the decision of what courses you teach and their content?
14. Has the SDA affected the course selection? Or the content of any courses? If yes, how?
15. What kind of a project course could offer the same as the SDA?
16. Do you feel that the SDA could be scaled up so that it could involve more students? If yes, how?
17. How do you see the future of the SDA?

Appendix C Sources in semi-systematic literature review

Table C.1: Included sources for data extraction

SID	Publication details
S1	Gotel, Olly, et al. "A global and competition-based model for fostering technical and soft skills in software engineering education." 2009 22nd Conference on Software Engineering Education and Training. IEEE, 2009.
S2	Scott, Andrew, et al. "A Holistic Capstone Experience: Beyond Technical Ability." Proceedings of the 18th Annual Conference on Information Technology Education. 2017.
S3	Koolmanojwong, Supannika, and Barry Boehm. "A look at software engineering risks in a team project course." 2013 26th International Conference on Software Engineering Education and Training (CSEE&T). IEEE, 2013.
S4	Keogh, Kathleen, Leon Sterling, and Anne Therese Venables. "A scalable and portable structure or conducting successful year-long undergraduate software team projects." Journal of Information Technology Education: Research 6.1 (2007): 515-540.
S5	Thompson, J. Barrie, and Helen M. Edwards. "A scalable approach to graduate student projects: Hundreds with industry every year." 2009 22nd Conference on Software Engineering Education and Training. IEEE, 2009.
S6	Mertz, Joseph, and Jeria Quesenberry. "A scalable model of community-based experiential learning through courses and international projects." 2018 World Engineering Education Forum-Global Engineering Deans Council (WEEF-GEDC). IEEE, 2018.
S7	Brazier, Pearl, Alejandro Garcia, and Abel Vaca. "A software engineering senior design project inherited from a partially implemented software engineering class project." 2007 37th Annual Frontiers In Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports. IEEE, 2007.
S8	Rusu, Adrian, et al. "Academia-academia-industry collaborations on software engineering projects using local-remote teams." Proceedings of the 40th ACM technical symposium on Computer science education. 2009.
S9	Stettina, Christoph Johann, et al. "Academic education of software engineering practices: towards planning and improving capstone courses based upon intensive coaching and team routines." 2013 26th International Conference on Software Engineering Education and Training (CSEE&T). IEEE, 2013.
S10	Venson, Elaine, et al. "Academy-industry collaboration and the effects of the involvement of undergraduate students in real world activities." 2016 IEEE Frontiers in Education Conference (FIE). IEEE, 2016.
S11	Eloe, Nathan, and Charles Hoot. "Accommodating Shortened Term Lengths in a Capstone Course using Minimally Viable Prototypes." 2020 IEEE Frontiers in Education Conference (FIE). IEEE, 2020.
S12	Schneider, Jean-Guy, et al. "Adopting industry agile practices in large-scale capstone education." 2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET). IEEE, 2020.
S13	Rover, Diane, et al. "Advantages of agile methodologies for software and product development in a capstone design project." 2014 IEEE Frontiers in Education Conference (FIE) Proceedings. IEEE, 2014.
S14	Ye, Huilin. "An Academia-Industry Collaborative Teaching and Learning Model for Software Engineering Education." SEKE. 2009.

SID	Publication details
S15	Mondragon-Torres, Antonio F. "An agile embedded systems capstone course." 2013 IEEE Frontiers in Education Conference (FIE). IEEE, 2013.
S16	Bareiss, Ray, and Edward Katz. "An exploration of knowledge and skills transfer from a formal software engineering curriculum to a capstone practicum project." 2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE &T). IEEE, 2011.
S17	Rusu, Amalia, and Mike Swenson. "An industry-academia team-teaching case study for software engineering capstone courses." 2008 38th Annual Frontiers in Education Conference. IEEE, 2008.
S18	Bell, John T., and Anushri Prabhu. "An innovative approach to Software Engineering term projects, coordinating student efforts between multiple teams over multiple semesters." 2014 IEEE Frontiers in Education Conference (FIE) Proceedings. IEEE, 2014.
S19	Von Konsky, Brian R., and Jim Ivins. "Assessing the capability and maturity of capstone software engineering projects." Proceedings of the tenth conference on Australasian computing education-Volume 78. 2008.
S20	Ziv, Hadar, and Sameer Patil. "Capstone Project: From Software Engineering to "Informatics"." 2010 23rd IEEE Conference on Software Engineering Education and Training. IEEE, 2010.
S21	Cornejo-Aparicio, Victor, et al. "Capstone courses under the PBL methodology approach, for engineering." 2019 IEEE World Conference on Engineering Education (EDUNINE). IEEE, 2019.
S22	Paasivaara, Maria, Jari Vanhanen, and Casper Lassenius. "Collaborating with industrial customers in a capstone project course: the customers' perspective." 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET). IEEE, 2019.
S23	Adams, Robert, and Carsten Kleiner. "Collaboration Support in an International Computer Science Capstone Course." International Conference on Social Computing and Social Media. Springer, Cham, 2016.
S24	Tøndel, Inger Anne, et al. "Collaborative security risk estimation in agile software development." Information & Computer Security (2019).
S25	Watkins, Kera Z., and Tiffany Barnes. "Competitive and agile software engineering education." Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon). IEEE, 2010.
S26	Nguyen, Duc Man, Tien Vu Truong, and Nguyen Bao Le. "Deployment of capstone projects in software engineering education at duy tan university as part of a university-wide project-based learning effort." 2013 Learning and Teaching in Computing and Engineering. IEEE, 2013.
S27	Lago, Patricia, Joost Schalken, and Hans van Vliet. "Designing a multi-disciplinary software engineering project." 2009 22nd Conference on Software Engineering Education and Training. IEEE, 2009.
S28	Ikonen, Marko, and Jaakko Kurhila. "Discovering high-impact success factors in capstone software projects." Proceedings of the 10th ACM conference on SIG-information technology education. 2009.
S29	de Souza, Rafael Tome, Sergio D. Zorzo, and Daniele Aparecida da Silva. "Evaluating capstone project through flexible and collaborative use of Scrum framework." 2015 IEEE Frontiers in Education Conference (FIE). IEEE, 2015.
S30	Vu, John Huan, et al. "Evaluating test-driven development in an industry-sponsored capstone project." 2009 Sixth International Conference on Information Technology: New Generations. IEEE, 2009.
S31	Delgado, David, et al. "Evolving a project-based software engineering course: A case study." 2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T). IEEE, 2017.
S32	Longstreet, C. Shaun, and Kendra Cooper. "Experience report: A sustainable serious educational game capstone project." Proceedings of CGAMES'2013 USA. IEEE, 2013.
S33	Dupuis, Robert, et al. "Experiments with adding to the experience that can be acquired from software courses." 2010 Seventh International Conference on the Quality of Information and Communications Technology. IEEE, 2010.
S34	Burge, Janet. "Exploiting multiplicity to teach reliability and maintainability in a capstone project." 20th Conference on Software Engineering Education & Training (CSEET'07). IEEE, 2007.
S35	Iacob, Claudia, and Shamal Faily. "Exploring the gap between the student expectations and the reality of teamwork in undergraduate software engineering group projects." Journal of systems and software 157 (2019): 110393.
S36	Panicker, Rajesh C., et al. "Exposing Students to a State-of-the-art Problem Through a Capstone Project." 2020 IEEE Frontiers in Education Conference (FIE). IEEE, 2020.

SID	Publication details
S37	Burden, Håkan, Jan-Philipp Steghöfer, and Oskar Hagvall Svensson. "Facilitating entrepreneurial experiences through a software engineering project course." 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET). IEEE, 2019.
S38	Sievi-Korte, Outi, Kari Systä, and Rune Hjelsvold. "Global vs. local—Experiences from a distributed software project course using agile methodologies." 2015 IEEE Frontiers in Education Conference (FIE). IEEE, 2015.
S39	Paasivaara, Maria, et al. "How does participating in a capstone project with industrial customers affect student attitudes?." 2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET). IEEE, 2018.
S40	Włodarski, Rafal, Aneta Poniszewska-Marańda, and Jean-Remy Falleri. "Impact of software development processes on the outcomes of student computing projects: A tale of two universities." Information and Software Technology 144 (2022): 106787.
S41	Neyem, Andres, Jose I. Benedetto, and Andres F. Chacon. "Improving software engineering education through an empirical approach: lessons learned from capstone teaching experiences." Proceedings of the 45th ACM technical symposium on Computer science education. 2014.
S42	Flowers, John G. "Improving the capstone project experience: A case study in software engineering." Proceedings of the 46th Annual Southeast Regional Conference on XX. 2008.
S43	Schneider, Jean-Guy, et al. "Industry Agile practices in large-scale capstone projects." Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings. 2020.
S44	Zilora, Stephen J. "Industry-Emulated Projects in the Classroom." Proceedings of the 16th Annual Conference on Information Technology Education. 2015.
S45	Spichkova, Maria. "Industry-oriented project-based learning of software engineering." 2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS). IEEE, 2019.
S46	Palacin-Silva, Maria, et al. "Infusing design thinking into a software engineering capstone course." 2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T). IEEE, 2017.
S47	Jiménez, Manuel, et al. "Integrating fundamental and advanced concepts in a rounded capstone design experience in computer engineering." 2007 37th Annual Frontiers In Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports. IEEE, 2007.
S48	Zeid, Amir. "Integrating international students' contests with computer sciecnce capstone: Lessons learned and best practices." 2012 Frontiers in Education Conference Proceedings. IEEE, 2012.
S49	Pereira, Claudia, et al. "Introducing Good Design Principles in an early System Engineering Course." 2009 Information Systems Education Conference, ISECON 26, 2009.
S50	Krusche, Stephan, and Lukas Alperowitz. "Introduction of continuous delivery in multi-customer project courses." Companion Proceedings of the 36th International Conference on Software Engineering. 2014.
S51	Santoso, Harry Budi, et al. "Investigating Students' Metacognitive Skills while Working on Information Systems Development Projects." 2017 7th World Engineering Education Forum (WEEF). IEEE, 2017.
S52	Holmes, Reid, et al. "Lessons learned managing distributed software engineering courses." Companion Proceedings of the 36th International Conference on Software Engineering. 2014.
S53	Haddad, Hisham M. "One-Semester CS Capstone: A 40-60 Teaching Approach." 2013 10th International Conference on Information Technology: New Generations. IEEE, 2013.
S54	Rusu, Adrian, et al. "Overcoming limited resources: An academia-government partnership on Software Engineering and Capstone Projects." 2007 37th Annual Frontiers In Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports. IEEE, 2007.
S55	Vasankari, Timo, and Anne-Maarit Majanoja. "Practical Software Engineering Capstone Course–Framework for Large, Open-Ended Projects to Graduate Student Teams." International Conference on Computer Supported Education. Springer, Cham, 2018.
S56	Karunasekera, Shanika, and Kunal Bedse. "Preparing software engineering graduates for an industry career." 20th Conference on Software Engineering Education & Training (CSEET'07). IEEE, 2007.
S57	Feldgen, Maria, and Osvaldo Clua. "Reflections of nine years of interdisciplinary capstone courses." 2010 IEEE Frontiers in Education Conference (FIE). IEEE, 2010.
S58	Fornaro, Robert J., Margaret R. Heil, and Alan L. Tharp. "Reflections on 10 years of sponsored senior design projects: Students win–clients win!." Journal of Systems and Software 80.8 (2007): 1209-1216.

SID	Publication details
S59	Majanoja, Anne-Maarit, and Timo Vasankari. "Reflections on Teaching Software Engineering Capstone Course." CSEDU (2). 2018.
S60	Ribaud, Vincent, and Vincent Leilde. "Relating Student, Teacher and Third-Party Assessments in a Bachelor Capstone Project." International Conference on Software Process Improvement and Capability Determination. Springer, Cham, 2017.
S61	Roach, Steve. "Retrospectives in a software engineering project course: Getting students to get the most from a project experience." 2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T). IEEE, 2011.
S62	Yuen, Timothy T. "Scrumming with educators: Cross-departmental collaboration for a summer software engineering capstone." 2015 International Conference on Learning and Teaching in Computing and Engineering. IEEE, 2015.
S63	Johns-Boast, Lynette, and Shayne Flint. "Simulating industry: An innovative software engineering capstone design course." 2013 IEEE Frontiers in Education Conference (FIE). IEEE, 2013.
S64	Boti, Evangeli, Vyron Damasiotis, and Panos Fitsilis. "Skills Development Through Agile Capstone Projects." International Conference on Frontiers in Software Engineering. Springer, Cham, 2021.
S65	Bruegge, Bernd, Helmut Naughton, and Michaela Gluchow. "SLPC++: Teaching software engineering project courses in industrial application landscapes—A tutorial." 2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T). IEEE, 2011.
S66	Paiva, Sofia Costa, and Dárlinton Barbosa Feres Carvalho. "Software CREATION WORKSHOP: A capstone course for business-oriented software engineering teaching." Proceedings of the XXXII Brazilian Symposium on Software Engineering. 2018.
S67	Chen, Chung-Yang, and P. Pete Chong. "Software engineering education: A study on conducting collaborative senior project development." Journal of systems and Software 84.3 (2011): 479-491.
S68	Katz, Edward P. "Software engineering practicum course experience." 2010 23rd IEEE Conference on Software Engineering Education and Training. IEEE, 2010.
S69	Vanhanen, Jari, Timo OA Lehtinen, and Casper Lassenius. "Software engineering problems and their relationship to perceived learning and customer satisfaction on a software capstone project." Journal of Systems and Software 137 (2018): 50-66.
S70	Bruegge, Bernd, Stephan Krusche, and Lukas Alperowitz. "Software engineering project courses with industrial clients." ACM Transactions on Computing Education (TOCE) 15.4 (2015): 1-31.
S71	Smith, Tucker, Kendra ML Cooper, and C. Shaun Longstreet. "Software engineering senior design course: experiences with agile game development in a capstone project." Proceedings of the 1st International Workshop on Games and Software Engineering. 2011.
S72	Decker, Adrienne, Christopher A. Egert, and Andew Phelps. "Splat! er, shmup? A postmortem on a capstone production experience." 2016 IEEE Frontiers in Education Conference (FIE). IEEE, 2016.
S73	Mahnic, Viljan, and Igor Rozanc. "Students' perceptions of Scrum practices." 2012 Proceedings of the 35th International Convention MIPRO. IEEE, 2012.
S74	Jarzabek, Stan. "Teaching advanced software design in team-based project course." 2013 26th International Conference on Software Engineering Education and Training (CSEE&T). IEEE, 2013.
S75	Krusche, Stephan, Mjellma Berisha, and Bernd Bruegge. "Teaching code review management using branch based workflows." Proceedings of the 38th International Conference on Software Engineering Companion. 2016.
S76	Paasivaara, Maria, et al. "Teaching students global software engineering skills using distributed scrum." 2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013.
S77	Khmelevsky, Youry. "Ten Years of Capstone Projects at Okanagan College: A Retrospective Analysis." Proceedings of the 21st Western Canadian Conference on Computing Education. 2016.
S78	Khakurel, Jayden, and Jari Porras. "The effect of real-world capstone project in an acquisition of soft skills among software engineering students." 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T). IEEE, 2020.
S79	Iacob, Claudia, and Shamal Faily. "The impact of undergraduate mentorship on student satisfaction and engagement, teamwork performance, and team dysfunction in a software engineering group project." Proceedings of the 51st ACM Technical Symposium on Computer Science Education. 2020.

SID	Publication details
S80	Hoar, Ricardo. "The Real World Web: How Institutional IT Affects The Delivery Of A Capstone Web Development Course." Proceedings of the Western Canadian Conference on Computing Education. 2014.
S81	Isomöttönen, Ville, and Tommi Kärkkäinen. "The value of a real customer in a capstone project." 2008 21st Conference on Software Engineering Education and Training. IEEE, 2008.
S82	Ahtee, Tero, and Mikko Tiusanen. "Towards an ideal software engineering project course." Proceedings of the 15th Koli Calling Conference on Computing Education Research. 2015.
S83	Rico, David F., and Hasan H. Sayani. "Use of agile methods in software engineering education." 2009 Agile conference. IEEE, 2009.
S84	Maxim, Bruce R., Stein Brunvand, and Adrienne Decker. "Use of role-play and gamification in a software project course." 2017 IEEE frontiers in education conference (FIE). IEEE, 2017.
S85	Bastarrica, María Cecilia, Daniel Perovich, and Maira Marques Samary. "What can students get from a software engineering capstone course?." 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET). IEEE, 2017.