Unreined Students or Not: Modes of Freedom in a Project-Based Software Engineering Course

Øystein Nytrø
Norwegian University of Science and Technology
Trondheim, Norway
0000-0002-8163-2362

Anh Nguyen-Duc
University of South-Eastern Norway
Bø, Norway
0000-0002-7063-9200

Hallvard Trætteberg
Norwegian University of Science and Technology
Trondheim, Norway

Madeleine Lorås
Norwegian University of Science and Technology
Trondheim, Norway

Babak Amin Farschian
Norwegian University of Science and Technology
Trondheim, Norway

Abstract— **Software engineering courses include practical and theoretical elements that give many options for pedagogical combinations among them. In this paper, we report on two different pedagogical approaches for an undergraduate, introductory project-based software engineering course with more than 500 students working in collaborative scrum teams. We call one approach 'Every Student is an Innovator', and the other 'No Student Left Behind'. This SE course has been long-running, with stable learning objectives and content. However, from one year to another, we radically changed the pedagogical approach of the course along several dimensions, among them the technical framework, software tools, project topic, mentor roles, assessment form and frequency, feedback and degree of student innovativeness. We report on the perceived challenges, detailed changes, the anticipated effects on the course learning outcomes. The results showed that innovativeness and fun need freedom and flexibility with processes and technology. However, strict design requirements and systematic guidance ensure fulfillment of learning objectives. Analyzing student and staff feedback, we find that both approaches lead to students using more time than intended and worrying about unknown assessment criteria.**

Keywords— Software Engineering Education; Project-based Software Engineering; Software Innovation; Continuous Integration; Continuous Assessment; Technology Support

I. INTRODUCTION

Educating software engineering (SE) students is a challenge of ensuring practical competence while still providing future-proof research-based knowledge [1, 4, 7]. Balancing the theoretical foundation with abilities in current practice and technology is an acknowledged problem [2]. On one hand, students need knowledge and awareness of SE principles, not only proper coding but also software architecture, requirement and testing. They should also be able to understand and use various types of development tools. On the other hand, an academic program in SE should enable future professionals to reason about and even develop new methods, competencies and tools as needed [3]. As educators, we should give the students insight into recent trends in the software industry, such as continuous deployment, DevOps, lean startup, digital transformation, and we should continually update various aspects of the curriculums. Large, lab-based courses in SE put a high demand on infrastructure and organizational resources. Course qualities like assessment consistency, precise schedules, explicit expectations and 24/7 infrastructure with technical support are expected. Teaching assistants must be trained, theory and practice must form a coherent unit, and technical and scientific staff must be agile and swift. Lecturers often struggle to ensure both lucid research-based lectures and a smooth large-scale lab operation. SE courses risk ending up with outdated practices and technology no longer in use [5], so detailed training in tools and methods may age quickly and become a career obstacle rather than an asset.

The backbone of our SE-course is a project-based lab-course where students work in agile teams, interspersed with this, we lectures on theory, tools and technology [8, 9, 10, 21]. We have seen that a simulation of a real-world project with failures, iterations, and successes motivates students to collaborate, explore and take responsibility [6]. *Our task as educators is to feed this peer-driven educational experience with the right constraints, theory, tools, and infrastructure. However, we struggled with finding the best framing, and we experienced a wide range of group behaviours and results: With too much freedom, the students' learning processes degenerated when encountering decisions and choices beyond their competence. However, too strict process constraints (if requested or found*

necessary by TA's) would hinder innovativeness, stifle the competitive spirit among the groups and inadvertently change behavior from controlled learning by doing, and failing, to rule-following.

This paper presents two different attempts to handle the multi-dimensional choices of course design: tools and methods, lectures, curriculum, supervision, and assessment. The SE research group responsible for SE courses, regularly changes lecturers, methods, tools and theory, while learning objectives remain unchanged. Our motivation for regular updates comes from a perceived gap between competence required in the national software industry and what can be found in traditional, text-book-based SE course curriculum. In two consecutive years, the SE course was taught with two different approaches by two different teaching teams. The underlying learning objectives were unchanged: Students should learn industry-relevant, modern, yet fundamental principles of software development and project practice. The course has several note-worthy attributes:

- *Large class size*: The course has more than 500 students annually, and students have widely varying experience with practical development tools. With limited resources for teaching and supervision, it is essential to find a cost-effective approach that realizes the course objectives and employs peer-learning, collaboration, student-based assessment and supervision in a realistic project setting.
- *Progressive enhancement of curriculum:* The curriculum must be research-based, according to Department Research Priorities. Yet, fundamental SE topics must be covered in detail. Besides, the course should give a sampler of further SE research topics and courses in programming environments, languages, domain modeling, architecture, Requirements engineering, DevOps, CSCW, etc.
- *Project-based course:* The students will develop a working product as a project team that is continuously being challenged to use methods and tools new to them. The course must offer both aid to the development process as well as systematic assessment and feedback regarding the learning objectives.
- *Trained, but lower-grade TA staff*: TA's are carefully selected and trained, each working 8-12 hours per week in addition to normal academic progression.

This study reports the findings from comparing two teaching approaches with regard to the effects on both teaching and learning. The Research Questions were:

RQ1. What dimensions of students' choices are possible to accommodate in a large, project-based SE course?

RQ2. What are the strengths/weaknesses associated with such freedom?

The remainder of this paper is organized as follows: Section II presents related work. Section III outlines our research methodology. Section IV presents the two different approaches, which we call 'Every Student is an Innovator' (ESIAI) and 'No Student Left Behind' (NSLB) respectively. Section V describes results, while Section VI discusses findings, validity and future work. Section VII concludes the paper.

II. Related work

Software engineering (SE) is defined as "*the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software*" [11]. Several challenges of teaching SE has been explored in various contexts, such as (1) to identify and transfer the key knowledge to the students, and also (2) to encourage students to develop the skills required to apply such knowledge in real-world work scenarios [15,16]. Although challenges apply to traditional teaching methods [15,16], some challenges are best met with hands-on experience [14].

Project-based teaching of software engineering is not new [2, 5, 10,13]. Recently, SE courses with more control over processes and practices, and less control over programming and tools have been reported: For example, Mahnic et al. described the experience of giving a strict guideline for teaching Scrum in a Slovenian capstone project [10]. Students learned and was graded by how well they carried out Sprints, controlled product quality, and produced documentation. The study does not mention how students were trained with regard to programming and tools. Some other courses use tools as the key infrastructures for teaching and assessing students, hence having more control over this dimension. Bruegge described the experience of teaching a large Software Engineering class (size of 300 students) with real industrial clients [2]. The author used a collaborative management environment that automates recurring assessment tasks. Feliciano et al. reported the benefit of using GitHub and open workflow to support collaboration among students [6]. However, that study does not answer how such course infrastructure support teachers in managing and assessing student's work. Other reports on innovative teaching approaches are eg. Broman et al. who described a company approach to teaching software engineering project courses. Students were organized into simulated companies and assessed according to process and product features [13]. These studies, however, did not systematically compare alternative approaches to student freedom of choice. They also did not compare the teaching outcome after introducing changes.

III. Research methodlogy

Our study analyzed both qualitative and quantitative data from multiple viewpoints [19]. Towards the common goal of

improving Software Engineering courses, we have designed and conducted similar data collection processes every year with different course instances. The overview of our research methodology is shown in Figure 1. Every year, we collected pre- and mid/end-course feedback from students by having them filling in surveys. These surveys provided us the students' background and experience with programming, project work and their motivations. The surveys also include their evaluation of our courses, for instance, their opinion about teaching materials, lectures, exercises, lecturers and project settings. The surveys also asked for perceived strengths and weaknesses experienced in the course. At the end of the courses, we asked students to provide feedback about learning outcome. Students' and TA opinions were also gathered by representative groups, informal interviews and smaller surveys.

We performed a retrospective comparative analysis that uncovers history and experience. The unit of analysis is the course itself, so we had two instances, one in 2017 (ESIAI) and another in 2018 (NSLB). The two teaching teams and teaching philosophies were different between these years. The analysis includes:

- Lecturers' focus group: including four lecturers of the two instances of the course, discussion of own experience with the course, feedbacks from TA teams, and comments from other lecturers in the department and external guest lecturers. This background provided the perceived strengths and weaknesses of each approach, together with their dimensions of teaching freedom.
- Comparison of students' feedback between the two course instances: We compared the summary of students' rating from both course instances.

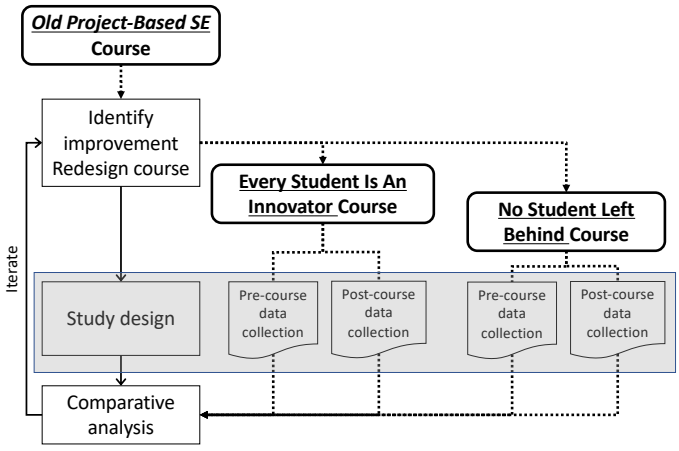

Figure 1: Research methodology

IV. THE TWO TEACHING APPROACHES

A. Common settings for the courses

The course is given to second-year computer science students, primarily admitted to a full 5-year master program. The course nominally requires students to work 12 hours per week for 13 weeks. The students are predominantly very high achievers, with top grades from secondary schools, but not necessarily technology savvy. Before taking the course, students must have completed courses in OO programming, algorithms, lab- and project courses, mathematics, statistics, and cross-faculty courses. The learning objectives of the SE course encompass theory and practice of development processes, agile methods, project management and –planning, modeling, UML, architecture, testing, evolution, CM, reuse, safety, security, quality and process improvement. Assessment is done mainly on group effort, without a final individual exam. We employed teams of nine to eleven TAs, with a total effort ranging from 2200 hours with additional graduate student support (ESIAI), to 2500 hours (NSLB). Both instances provided approximately 20 minutes/student/week, or more than 2 hours per group per week. TA effort is divided between giving aid and doing product/deliverable assessment. Lecture halls for plenary sessions with all students are available 6 hours per week, and course-assigned smaller lab and seminar rooms for presentations, group work and TA-aid are available 24/7. TA's are recruited among higher-year students, with grades in the same course among the 20% best. TA's receive pedagogical training and are instructed to be good mentors. Different TA functions (in particular assessment) are circulated, but direct group aid and feedback are often given by the same 1-2 persons throughout the course. The schedules of the courses are described in Table 1. Lectures, sprints, deliverables and assessment weighting are shown.

B. Approach 'Every Student Is An Innovator' (ESIAI)

ESIAI: Motivation for the Approach

Besides teaching fundamental Software Engineering concepts and phenomena in a hands-on and understandable way, the Spring 2017 instance of the course emphasized students' innovative capacities. Inspired by a large number of successful startups founded by university students, we would like to encourage the self-learning process that is driven by joy, curiosity, and discovery. We aimed at not only teaching students about fundamental software development activities but also attempting to introduce novel software development approaches, i.e. Lean Startup, Mobile D, and SEMAT

Table 1: Course schedules for approaches ESIAI and NSLB

	Week no.	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17
ESIAI	Lectures	Process-SEMAT, MobileD	Process - Metric	Architecture	Requirement	Software quality	Testing	Project management	SEMAT (2)	Secure SE						
	Process				Sprint 0	Sprint 1		Sprint 2		Sprint 3		Sprint 4			Sprint 5	
	Deliverables			Project plan Backlog Poster			Sprint 1 demo		Sprint 2 demo		Sprint 3 Demo		Team practice card	Final delivery Video Report		
	Assessment			100%												
NSLB	Lectures	Intro, Tech	Scrum, CI, Req, GitLab	VC, Testing, Framew.	Agile meth., Testing	Safety, Quality, CI	Cardiology, SO, servers	Enterprise arch., UML	Archi., Gamific., Security							
	Process			Sprint 1				Sprint 2				Sprint 3				
	Deliverables			GitLab config	Project plan	MC Test					Demo 1				Demo 2	MC Test
	Assessment			10%	10%	10%	10%	10%			10%	10%			20%	10%

ESSENCE. Students also participated in various innovative SE activities, such as interviewing professional developers, making presentation slides, posters and videos. The dimensions of ESIAI are summarized in Table 2.

Table 2: Dimensions of Teaching ESIAI

Dimensions	Fixed	Flexible
Project setting	Project theme Report template Overall architecture	Project idea Product backlogs Detailed architecture
Process	Number of Sprints Duration of Sprints Delivery of each Sprint Weekly supervision meeting	Process metrics Adopted practices Team communication and meeting Quality assurance
Technology	Version Control System	Programming language (frontend, backend) Servers, Database
Supervision	Weekly meeting Delivery	N/A
Assessment	Video-presentation and report eval. at end of course	N/A

ESIAI: The project and process

Due to a large number of student teams (around 110 teams), we did not assign specific projects to students. Rather than that, a general theme was introduced at the beginning of the course. In Spring 2017, the theme was "*Revolutionizing the learning-experience in university education with roBOT technology*". Students gathered requirements about problems of university education solvable by a software system, conceptualized and presented a solution. After that, they were guided through a systematic workflow to implement demonstrable solutions. In other words, students had considerable freedom in the journey of revolutionizing educational work while learning SE. The project did not involve real customers. A team of supervisors was employed to assist students in refining their requirements and implementing the solutions. Students were required to formulate user stories, product backlogs, non-functional requirements, to reason and document their architectural decisions via architectural diagrams.

Regarding methodologies, students were required to follow Mobile-D approach [17]. The method is based on Extreme Programming (XP), Crystal methodologies, and Rational Unified Process, optimizing for developing mobile, lightweight software products. The project consists of five iterations, namely set-up, core, core 2, stabilize, and wrap-up. Each project phase consists of three different types of development days: Planning Day, Working Day, and Release Day. As the course instructors, we set up checkpoints at the end of each iteration, where students reported their progress. During each iteration, the students had large freedom in deciding their team, roles, and working style in the projects. The students were introduced to SEMAT ESSENCE language [18] to build their work

processes. Students needed to identify language elements, such as Opportunity, Stakeholders, Requirements, Software System, Work, Team and Way of Working and monitor the evolution of the elements' states.

ESIAI: The development platform and technology stack

In general, there were no fixed requirements for technology and programming language in the course. The students would propose a technology stack. They were recommended to use GitHub as the version control system. Almost every team were guided to use Trello for project management. Some teams chose alternative tools, such as taiga.io. Students had experience from other courses, and would use HTML, CSS, and Javascript in the front-end, Python, and Java in the backend. Most of the teams used external libraries or frameworks, such as jQuery, Jasmine, NodeJS, Django, JavaFX, api.ai, etc. The extent of using databases varied among students, from traditional SQL databases to No-SQL databases.

ESIAI: Supervision and assessment

The teaching team included lecturers who delivered lectures, prepared assignments, did administrative work and managed the TA team. The TA team consisted of seven people (Ph.D students, researchers and master students), who closely interacted with student teams, gave feedback to their assignments, supported them with the methods (i.e., SEMAT, Mobile D), and technical infrastructure (Git, Trello, etc.). During the process observation, a team would meet their supervisor every 1-2 weeks for process and technology assistance. Each TA supervised 15 groups, spending an average 20-30 minutes for each group weekly. TAs used Google sheets to note the progress of their teams. The intermediate deliveries from students during the projects were co-assessed by course responsible and TA's.

The final assessment of the performance of students were based on the demonstrated quality and implementation of their products, deployment packages and final reports. The course instructors went through all deliverables from students, took into account the notes and pre-assessment from TAs to determine the final grade. There was no written exam in this course instance. Student received no grading or formal feedback during the course.

C. Approach 'No Student Left Behind' (NSLB)

NSLB: Motivation for the Approach

The main goal of the Spring 2018 instance of the course was to ensure that a majority of students reached the learning objectives, and to prevent that entire groups or individuals in a group would fail to reach basic SE learning objectives because of unfortunate decisions, group dynamics, unvetted requirements or plain bad luck. Secondary to the idea of not leaving any student unwillingly behind was an ambition to ensure a higher average student competence and experience in software engineering. The dimensions of NSLB are summarized in Table 3.

Table 3: Dimensions of teaching NSLB

Dimensions	Fixed	Flexible
Project setting	Overall architecture and server functionality	Domain, application, user functions
Process	Scrum with sprints	Roles, organization
Technology	GitLab support, setup and working code templates, example service stack setup.	Students could decide on prog. language, version control, service stack etc. Most would follow provided examples and templates.
Supervision	Given according to defined deliverables, process and content requirement. TA's trained in templates and examples.	Ample resources and agile and eager staff. Full flexibility in use of staff with time, location and medium.
Assessment	Continuous, structured, assessment of various types of deliverables. Individual multiple choice tests.	Senior staff would receive complaints about unfair deliverable evaluation, and could intervene if valid complaint.

NSLB: The project and process

To reach these goals, the staff developed a semi-realistic, agile project, with common objectives/requirements and deliverables for all student groups. This, in turn, enabled TA's to give comprehensive and competent assistance within one problem domain, and the staff to make predictable/fair criteria for student-based assessment.

The student project groups were randomly composed with a team size of 6 to 8 members. To make the project challenging, the student groups needed freedom in forming their products and deciding on their functionalities. In order not to overstretch TA competence, we specified the general architecture and technology stack in some detail. Hence, elements of realism, like having a specific product owner or selecting technology, were sacrificed.

A typical group project would be about data collection, storage and data analysis. Data from some kinds of activities performed by 'data providers', eg. patients or athletes would be gathered by wearable sensors, communicated and stored on a server. A 'service provider' or data analyst, eg. a health monitoring service or coach, would give value back to the data provider by analyzing the data and visualizing the result. The mentioned scenarios were suggested and supported with guest lectures, eg a cardiologist lecturing about the importance of remote monitoring of patients at risk for sudden cardiac death. Based on student choices, and ideas, the groups were required to conceive a system consisting of a server, user interface and data collection interface. The requirements were detailed as user stories, and three sprints were scheduled for the development process.

An agile software development process is close to a full-time effort. The division of labor, roles, and progress relies on high turn-around, agility and frequent exposure to a problem owner and other driving factors. For each sprint, user stories with high priorities were turned into development tasks and combined with management tasks (like set up, configuration and later refactoring) into a backlog for the sprint.

We required the student groups to use our in-house GitLab instance for process support, both because it's a realistic and relevant tool and, as such, a part of the course's learning goal, and to ensure we could follow and assess the process without assessment-specific deliverables. All team tasks were (supposed to be) recorded as GitLab issues and related to sprints through milestones. Issue boards were used to manage the overall process, while details of each issue, e.g., team member assignments, discussions and decisions, were part of the issue. GitLab supports relating code commits to issues, so ideally, the issue page should capture every relevant aspect of development work, making the process completely transparent to team members, TA's and senior staff. This proved to be very useful and relevant for (student group) team members, it also allowed assessing the process without requiring the students to produce documentation not part of the development process.

NSLB: The development platform and technology stack

As a part of making the project realistic, we wanted to ensure the products had a certain architectural complexity and were suitable for practicing both testing and continuous integration, both of which are important in agile projects. We also needed to build on the expected student competency from previous courses, which included procedural programming with Python, object-oriented programming with Java, simple desktop application architecture, but not writing own tests or using build tools. Most students were taking UX and DB courses in parallel with the SE course.

We decided on using the Java platform, with a JavaFX-based desktop client, a SQL-based database, and a servlet-based server as requirements for the product. We did not require any specific Servlet-framework but lectured and provided examples of both HttpServlet and JAX-RS (with Jersey) programming. We used Maven as a build tool and provided complete but minimal project code stubs to get teams started. In addition, the course staff developed a 'product' with the same architecture, complexity, and size as what we expected from them, using the same tools, including GitLab with proper issue tracking, that they could use as learning material. This included tests for domain classes, database and Servlet code, with proper maven configuration (pom) files.

NSLB: Supervision and assessment

The team consisted of two responsible lecturers who planned, prepared, set up GitLab and templates and gave lectures on general SE topics. Guest lectures covered Scrum approach, security, architecture, UI, safety, mobile monitoring, health monitoring. Our TA's were lower-grade students. More competent supervisors (graduate or higher-grade students of SE) were not available on short notice. An engineer maintained the local GitLab installation. A group of three leading TA's managed and collected assessment instructions and templates, and configured the learning software. A student reference group provided continuous input on student progress, general problems, and areas needing improvement. Each group had two specific TA's allocated throughout the course, both taking roles as mentors and assessors. The supervision was continuous and organized as meetings, message interchanges, or GitLab-based communication. TA's and lecturers continuously monitored progress and products. Students and TA's were free to organize meetings and evaluation sessions (except demos which required suitable rooms).

Piazza was used as a common Q&A channel and logged 380 posts, 1906 contributions, 464 instructor responses, and 124 student responses. Piazza usage was very useful, and almost all questions were of general interest and not answered by TA's. Correspondingly, the two responsible lecturers provided 22% and 12% respectively of the total contributions, with TA's very much less. At times, this was challenging, requiring a responsive and hands-on approach. Lecture topics were partially decided from Piazza discussions.

Students' grades were mainly based on group deliverables and two individual multiple-choice tests. The other deliverables were three sprints, two demos, and two technical products. Each deliverable accounted for 10% of final grades towards the total individual score, except for the last demo, which accounted for 20% of the final grades. The distinction between individual students was in other words only based on individual multiple-choice tests making up 20% of the total evaluation.

Demos required all group members to be prepared and present, and the assessors would draw random presenters and demonstrators among the group. Demos were taped for record and control and held publicly. In practice, the availability of suitable seminar rooms limited attendance, and the other groups were often preoccupied with their own demos. Multiple-choice exams were introduced to ensure and test individual theoretical knowledge, tools competence, and process awareness.
The responsible lecturers made final individual grading based on the accumulated individual point scores, TA reports, control of deliverables, detailed inspections and in some cases, investigation into perceived inconsistencies and possible unfair practice.

V. Results

This section presents the results of the evaluation of two teaching approaches.

A. Approach 'Every Student Is An Innovator'

Students provided both qualitative and quantitative feedback on how they perceived the course. Overall, students felt the course to be interesting and inspiring, close to a professional working environment (in so far as they would know what that would be like). The freedom given to students in the problem space had motivated them to acquire competence to implement their projects:

> *The assignment given to us has been open and inspiring, motivating us to learn new things on our own and make ourselves able to complete the goals we had set at the start of the project.*"

However, the flexible options for methodological practices make many teams confused about what they should use for their projects. The amount of freedom in both problem and solution domains had burdened the TA's, making it difficult to assist each team (with their own problems and own approach). Specific choices of programming language and database solution were recommended, in order to be able to provide students with technical support:

> *We* realize *that having available and competent student assistants is a challenge, especially considering the degree of freedom every student team has in deciding how to develop their software. TA's would often be of little help wrt. technical problems.*

The course assessment motivated students to focus on conducting a proper team-based process and product. It was reported from students that they would not prefer a final written exam, given the amount of efforts they have put in the project. A retrospective meeting and focus groups among teachers also provided learned lessons for future course design. Firstly, with the current project setup, we could reduce the amount of focus on development methodologies and spend more time on technology stack and infrastructure.

Table 4: Analysis of Teaching ESIAI

	Strength	**Weakness**
Project setting	Fun and motivating for students	Ambitious Change-prone
Process	Learning all elements of software development	Lack of guidance. Overwhelming Not involving all team members
Technology	Practical and flexible	Time consuming Lack of prior knowledge
Supervision	Quick overview of student teams via a team of seven supervisors	Little time and focus for each student team
Assessment	Manageable workload	Not repeatable and TA-specific

It seemed a good idea to require, or recommend the same technology for all teams, so they could get sufficient technical support when needed. Secondly, the students should be taught a key set of practices and how to adopt them in a real case, before selecting among approaches without required experience. The structured support and guidance to the practices would complement the given workflow. The summary of the strengths and weaknesses of the approach is shown in Table 4.

B. Approach 'No Student Left Behind'

The defined project process and provided templates gave students a well-defined workflow with a set of team assignments. The problem space was fixed and the solution space was guided. The process was not realistic but afforded TA's and teachers good control and competent and timely feedback. The most successful part of the course was a fulfillment that *no one was left behind*. Tests showed consistently high average score on all SE topics. Teams were monitored for excluded or parasitic team members, but this was hardly a problem (2 cases had to be resolved because of conflicting student schedules). Randomly composed teams worked seemingly very well wrt. shared workload.

Some students wanted to use a different technology stack, e.g., write a web-client in JavaScript and server using node.js or Python, based on what they were comfortable with and what they wanted to learn. That was allowed, at the risk of getting less TA support. Some felt the expected product complexity was too high, particularly the hierarchical maven project configuration. The argument for making templates and comprehensive code examples was to ensure all learning goals could be met and assessed. The supervision and assessment proved to be robust, repeatable, but required hands-on competence and training among staff. Some statements from the TA's are quite interesting:

> *…Technology and tool focus quenched innovativeness … but increased control of progress and effort.*

> *…Very difficult to separate between groups, all performed at a very high level and got high scores.*

Anonymous student responses from a reference group, surveys and discussion fora are also illustrative about the strengths and weaknesses. Some important observations from students are:

- *Agile methods require fast feedback from the simulated problem owner!*
- *A question from one student group would lead to public broadcast of clarification. Spiraling effect of added detail and irrelevant information were seen in stressful periods.*
- *Assessment and mentoring (feedback) role of TA not clear enough. Roles should be clearly separated.*
- *Agile course produced too much information. Public channels lead to uncertainty and disturbances.*
- *Lots of work! Good experience! Different kind of course!*
- *Very transparent and detailed scoring of deliverables made even small variation among TA practice a source of discontent. Group competition was not intended or wanted.*

Two individual multiple-choice tests (each with 20 questions drawn from separate pools of approximately 60 questions) were designed to test theoretical knowledge as well as process- and technology awareness. The tests were reasonably difficult, with the mean score at 80% and 72% respectively. The test had very high participation, low deviation and very few (less than 1%) below a critically low competence level (40%). We tried informally to measure stress and insecurity by reading and searching for relevant keywords in student comments in discussion fora. Some findings:

- Assessment discontent quickly fell after the first two deliverables, the students got to understand the process and we instructed TA to clarify the distinction between feedback (mentoring) and requirement/assessment.
- After a while, project- and team-specific problems were not published, unless relevant and posted by TA or lecturers. Initial overload of technicalities scared the less savvy students. Reduction of information reduced student insecurity.
- Multiple-choice tests were not familiar to many students. Before, during (!) and after these, complaints about the relevance of theory to the project surfaced. However, this was intended. The second test was thus perceived as much less stressful.
- The significant majority of students performed very well. Some islands of discontent about other teams getting too good scoring on products persisted for a while, but this was handled and explained using student representatives. Hopefully as a learning experience for all involved.

Strengths and weaknesses of the approach are summarized in Table 5.

Table 5: Analysis of teaching NSLB

	Strength	**Weakness**
Project setting	Bootstrapping course, readymade examples, very direct and early start, no time wasted on fictive users.	Lack of realistic stakeholders, user stories. No RE exposure. Not so fun. Not a soft start.
Process	Realistic. Controllable. Transparent. Competent TA's.	Scrum not suited for part-time simulation. Prone to escalate resource usage.
Technology	Solid, relevant, robust and useful	Less fear and exploration. New to TA's**!**
Supervision	Available, agile, personal, continuous, and at times worn thin.	Less improvising, grit and innovativeness. High demand on staff resources.
Assessment	Fair, repeatable and predictable. Each deliverable had detailed criteria.	High score attainable. Intra- and extra-team peer pressure and competition.

VI. DISCUSSIONS

To answers both RQ's, we performed a comparative analysis of the two teaching approaches. The discussions on both RQ's are given below.

A. *Answering RQ1* - What dimensions of students' choices are possible to accommodate in a large, project-based SE course?

To answer RQ1, we find the common flexible elements across dimensions between the two approaches that are positively perceived by both lecturers and students. The result is that all of the dimensions (1) project setting, (2) process, (3) technology, (4) supervision, and (5) assessment allow and need a certain level of flexibility to cope with the variety of projects, students' experience and TA's experience. We clearly experienced that freedom of choice is both inspiring and challenging but must be balanced with precise control in order to reach learning objectives and maintain fairness. From the findings, we suggest several dimensions of choices that might impact the course settings:

- Freedom of technology and method choices reduce the value and validity of TA aid and assessment.
- Freedom of problem selection increases involvement, and time spent.
- Freedom of team arrangement increases team competitiveness and potential student lockout or team failure.
- Freedom (lack) of precise deliverable content and form makes assessment non-transparent and subjective.

B. *Answering RQ2 - What are the strengths/ weaknesses associated with such freedom?*

For answering RQ2, Table 6 summarizes the strengths and weaknesses of the two approaches. Increased freedom of choices would probably increase the fun and creativity among some students, however it increases the risks in teaching and managing the course as well. Also, it comes at a cost of decreasing freedom and learning for some individuals and groups. Increased freedom would probably also increase the learning effect and efficiency. For some. Few, not the majority. Having the freedom to make active decisions with regards to the problem domain, process, technology and deliverable content is valuable. The question remains when, and if, the students are competent enough to thrive and learn from this freedom. Furthermore, at what stage are the student mature enough to accept that failure is an important part of innovation? We do not yet know if an early failure, or success, as a software engineer in an introductory course, will weed out the conscientious data analysts, and leave us with the ever-innovative and sanguine entrepreneur. The industry needs both, and both need a proper software engineering background.

Table 6: Comparing NSLB and ESIAI

	Strength	Weakness
No Student Left Behind	Thorough and predictable coverage. High, average level of competence. Effective TA involvement.	Too many details that may be relevant to everybody. Hard work and less fun, negative appreciation of innovation.
Every Student Is An Innovator	Fun and motivating. Lifelike and realistic learning. Exposure to innovative thinking in teams.	Uncertain individual learning outcomes. Hard to control resource use. Little cross-team communication. Ineffective TA's. Overwhelming.

VII. CONCLUSION

This study compares two teaching approaches, one focusing on systematic guidance and education, and another focusing on innovativeness and inspiration. Feedback from students and relevant lecturers were collected to evaluate each of the approaches. The comparison between the two approaches provides direct implications for teaching the course in the future and also for similar project-based SE courses.

Both of the teaching approaches have conflicting strengths and weaknesses that must be balanced according to objectives, available resources and student background. Innovativeness and fun need freedom. Predictable outcomes and attainment of basic learning objectives require control. Mixing these is possible, but put a high demand on a responsive, resourceful and competent teaching staff. A possible combination of these dimensions would probably be a "innovative problem – guided solution" approach, where students are inspired and motivated to define their projects, with requirements and product backlogs. The requirements and early design phases can be validated by staff to ensure feasible technical solutions for them. In the solution space, students should be systematically guided with technological infrastructure and necessary

programming competence. They need to carry out the solution by themselves, according to a set of given processes and practices. Future work in this area would include a more thorough quantitative and qualitative analysis of individual student experience and outcome. This would open up for precise adjustment of freedom along the dimension that we have outlined, according to available resources and our ambitions to teach future innovative and competent software engineers.

VIII. Acknowledgments

We are grateful to all the students, technical staff, student representatives, reference groups and TA's who challenged, asked, helped and answered us. Thank you!

References

[1] T. B. Hilburn, and W. S. Humphrey, W.S.The Impending Changes in Software Education. IEEE Softw. 19, 5 pp. 22–24, 2002

[2] B. Bruegge, S. Krusche, and L. Alperowitz. Software Engineering Project Courses with Industrial Clients. Trans. Comput. Educ. 15, 4 pp. 1–31, 2015

[3] H. Jaakkola, J. Henno, I.J. Rudas, IT Curriculum as a complex emerging process, in: IEEE International Conference on 2006 ICCC Computational Cybernetics, IEEE, IEEE Society, pp. 1–5, 2006

[4] J. Chen, H. Lu, L. An, Y. Zhou, Exploring teaching methods in software engineering education, Computer Science & Education 2009. ICCSE'09. 4th International Conference on, pp. 1733-1738, 2009.

[5] M. Luukkainen, A. Vihavainen and T. Vikberg. Three Years of Design-based Research to Reform a Software Engineering Curriculum. Proceedings of the 13th Annual Conference on Information Technology Education, New York, USA, pp. 209–214, 2012

[6] J. Feliciano, M. Storey, and A. Zagalsky. Student Experiences Using GitHub in Software Engineering Courses: A Case Study. 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pp. 422–431, 2016

[7] R. Lingard and S. Barkataki, "Teaching teamwork in engineering and computer science," in *2011 Frontiers in Education Conference (FIE)*, 2011, pp. F1C-1-F1C-5, doi: 10.1109/FIE.2011.6143000.

[8] J. Vanhanen, T. O. A. Lehtinen, and C. Lassenius, "Software engineering problems and their relationship to perceived learning and customer satisfaction on a software capstone project," *Journal of Systems and Software*, vol. 137, pp. 50–66, Mar. 2018, doi: 10.1016/j.jss.2017.11.021.

[9] J. Campbell, S. Kurkovsky, C. W. Liew, and A. Tafliovich, "Scrum and Agile Methods in Software Engineering Courses," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, Memphis, Tennessee, USA, 2016, pp. 319–320, doi: 10.1145/2839509.2844664.

[10] V. Mahnic, "A Capstone Course on Agile Software Development Using Scrum," *IEEE Transactions on Education*, vol. 55, no. 1, pp. 99–106, Feb. 2012, doi: 10.1109/TE.2011.2142311.

[11] I. ISO, Systems and software engineering – vocabulary, ISO/IEC/IEEE 24765:2010(E), 2010, pp. 1–418, doi:10.1109/IEEESTD.2010.5733835

[12] L. Brodie, H. Zhou, and A. Gibbons, "Steps in developing an advanced software engineering course using problem based learning," *Engineering Education*, vol. 3, no. 1, pp. 2–12, Jun. 2008, doi: 10.11120/ened.2008.03010002.

[13] D. Broman, K. Sandahl, and M. Abu Baker, "The Company Approach to Software Engineering Project Courses," *IEEE Transactions on Education*, vol. 55, no. 4, pp. 445–452, Nov. 2012, doi: 10.1109/TE.2012.2187208.

[14] E.O. Navarro, A. Baker, and A. van der Hoek. "Teaching Software Engineering Using Simulation Games". Proceedings of the International Conference on Simulation in Education. IEEE Press, 2004.

[15] R. B. Vaughn and J. Carver, "Position Paper: The Importance of Experience with Industry in Software Engineering Education," in *19th Conference on Software Engineering Education and Training Workshops (CSEETW'06)*, 2006, pp. 19–19, doi: 10.1109/CSEETW.2006.14.

[16] D. L. Parnas, "Software engineering programs are not computer science programs," *IEEE Software*, vol. 16, no. 6, pp. 19–30, Nov. 1999, doi: 10.1109/52.805469.

[17] P. Abrahamsson, A. Hanhineva, H. Hulkko, T. Ihme, J. Jäälinoja, M. Korkala, J. Koskela, P. Kyllönen, and O. Salo. Mobile-D: An Agile Approach for Mobile Application Development. Companion to the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications (New York, NY, USA, 2004), pp. 174–175.

[18] http://www.semat.org

[19] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," Empir Software Eng, vol. 14, no. 2, p. 131, Dec. 2008, doi: 10.1007/s10664-008-9102-8.

[20] K.-K. Kemell, A. Nguyen-Duc, X. Wang, J. Risku, and P. Abrahamsson, "The Essence Theory of Software Engineering – Large-Scale Classroom Experiences from 450+ Software Engineering BSc Students," in *Product-Focused Software Process Improvement*, Cham, 2018, pp. 123–138, doi: 10.1007/978-3-030-03673-7_9.

[21] A. Nguyen-Duc, S. Khodambashi, J. A. Gulla, J. Krogstie, and P. Abrahamsson, "Female Leadership in Software Projects—A Preliminary Result on Leadership Style and Project Context Factors," in *Towards a Synergistic Combination of Research and Practice in Software Engineering*, P. Kosiuczenko and L. Madeyski, Eds. Cham: Springer International Publishing, 2018, pp. 149–163.