# Effective Teaching Strategies for Large Classes: A Case Study in Software Architecture Education

Laura M. Castro
Universidade da Coruña
A Coruña, Spain
lcastro@udc.es

Mauricio Hidalgo
Universidad Finis Terrae
Santiago, Chile
mhidalgob@uft.edu

Hernán Astudillo
Universidad Andrés Bello
Viña del Mar, Chile
hernan@acm.org

## Abstract

Right after the Covid-19 pandemics, faculty in charge of a Software Architecture course (3rd year subject of a Software Engineering program) faced an unprecedented increase in enrollment figures, doubling previous numbers to reach over a hundred.

To adapt their teaching methodologies to this situation, they deployed a strategy of *early involvement* using a gamified programming learning platform during the first weeks of the semester, together with *personal learning contracts* during the whole duration of the course.

The course had a record success rate of 92%, versus an average of 71% in previous years, but some aspects demand redesign for future versions. First, most students got a full score on their personal learning contracts, and got it *early* rather than waiting for the semester final weeks; hence, chronological ordering of assignments matters even more than before, as latter assignments may go unfulfilled. Second, many students did this while skipping assignments for some topics, specially visual modeling; hence, learning contracts should stipulate minimal achievements for all learning of special interest.

This work aims, through the detailed analysis of this teaching experience, to serve as an inspiration for other faculty who may face similar realities.

## CCS Concepts

• **Social and professional topics** → **Software engineering education**; • **Applied computing** → *Education*; • **General and reference** → Empirical studies.

## Keywords

learning contract, engagement, mentoring, continuous evaluation

**ACM Reference Format:**
Laura M. Castro, Mauricio Hidalgo, and Hernán Astudillo. 2024. Effective Teaching Strategies for Large Classes: A Case Study in Software Architecture Education. In *2024 the 16th International Conference on Education Technology and Computers (ICETC) (ICETC 2024), September 18–21, 2024, Porto, Portugal.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3702163.3702414


*ICETC 2024, September 18–21, 2024, Porto, Portugal*

ACM ISBN 979-8-4007-1781-9/24/09
https://doi.org/10.1145/3702163.3702414

## 1 Introduction

The Degree in Software Engineering at Universidade da Coruña includes in its curriculum five intensifications, corresponding to five Specific Technologies of Software Engineering, which have a precise parallel with the five disciplines outlined in the ACM/IEEE Computing Curricula [12].

Out of these five possible specializations within the official Degree in Software Engineering at Universidade da Coruña, two are clearly oriented towards software development: the intensification in Software Engineering (hereinafter, SE) and the intensification in Computer Science (hereinafter, CS). SE focuses more on creating robust software by satisfying corresponding application-level requirements, while CS concentrates on a broad spectrum of computing problems and applications, but their *proximity* allows students pursuing these mentions to share content (specifically, courses) beyond the common core of the degree. This is the case of the Software Architecture course, which is offered in SE as a mandatory subject in the third year, and in CS as an elective subject in the fourth year.

After the implementation of the Degree in Computer Engineering at Universidade da Coruña in the 2010/11 academic year, the subjects corresponding to the third year started to be taught in the 2012/13 academic year. Figure 1 shows the evolution of the number of enrolled students in the years since then.

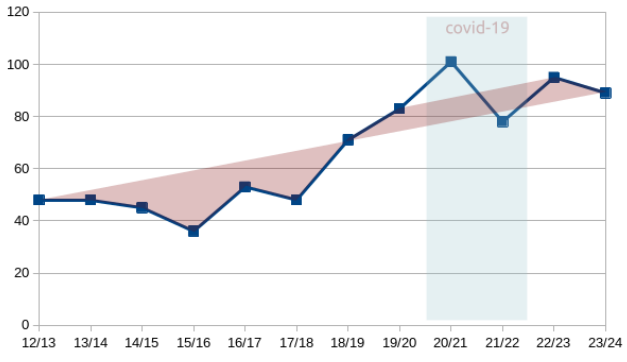


**Figure 1: Enrollment statistics for the Software Architecture course since its implementation**

As we can observe, although there was a to-be-expected gentle upward trend from the very beginning, the enrollment for the course kept consistently rising, reaching more than a hundred students in 2020/2021, the academic year following the COVID-19 pandemic. It is needless to reiterate all the exceptional circumstances surrounding that time, but the fact is that the data for the

subsequent academic years consolidated this course as a large-class one.

Naturally, this situation requires a proactive attitude on the professors' side, as there is a risk that a large proportion of the students, in a course which teaching strategies originally designed and well-dimensioned for a smaller group (half the size), may end up dropping out or failing to pass.

In the remainder of this paper, we will present in detail the innovations introduced, which constitute a successful experience in managing a large group of students. Our contribution materializes in a combination of two strategies:

- *Early Engagement Strategy*: This strategy aims to involve the student from the first days of the course as a method to prevent dropout.
- *Adapted Learning Contract Strategy*: This strategy aims to encourage reinforcement of the content covered, adapting it to the particular characteristics of each student, as a method to increase performance and, consequently, the chances of passing the course.

The rest of this document is structured as follows: Section 2 presents the descriptive elements of the Software Architecture course, specifically detailing the learning objectives, teaching methodologies used to achieve them, and evaluation methods employed to determine their achievement. Subsequently, Section 3 describes the innovations introduced, and Section 4 analyzes the results of their implementation. We discuss the experience in Section 5 and finally present our conclusions in Section 6.

## 2 Software Architecture

With a teaching load of 6 ECTS credits, equivalent to 150 hours of work for each student, the Software Architecture course at Universidade da Coruña aims to expose students to the current alternatives in Software Engineering regarding the design of applications and software systems at the architectural level. This includes studying the most typical architecture styles and their characteristics; examining the non-functional requirements of systems and their relationship to architecture; and developing and/or analyzing real-world applications. In other words, the course seeks to prepare future graduates for selecting, adapting, and maintaining architecture as part of the software development process, including identifying components and dependencies, and selecting structures, data, and communication channels.

### 2.1 Learning objectives

To achieve the competencies assigned to the Software Architecture course, the learning objectives specified and summarized in Table 1 are defined.

The learning objectives motivate both the choice of content for the course, which we present below, and the selection of teaching strategies, described in Section 2.2. The content is structured into four main blocks:

**Software Architecture Concept**
Exposing students to the concept of software architecture implements the first learning objective (**O1**), as both the theoretical concepts related to software architecture and the practical professional profile of a software architect are inherently part of Software Engineering. The key methodology to achieve this is lectures (cf. Sect. 2.2).

**Models, Patterns, and Reference Architectures**
Exposing students to various well-known architectural models, including reference architectures in both distributed and non-distributed environments, primarily implements the second objective (**O2**). The use of modeling artifacts to describe these architectures (specifically, the C4 model [15]) also contributes to the fourth objective (**O4**). Analyzing bibliographic sources and engaging in guided discussions (cf. Sect. 2.2) is particularly relevant in developing critical thinking skills.

**Component Design and Integration**
Exposing students to the design of architectural proposals based on components, either following reference models or proposing mixed or hybrid approaches, implements the third and fifth objectives (**O3** and **O5**). The use of modeling artifacts to describe these designs also contributes to the fourth objective (**O4**). In this context, laboratory practices and supervised projects are the most relevant methodologies for achieving the learning objectives associated with this content block.

**Traceability and Integration Testing**
Explicitly exposing students to requirements traceability and integration testing, which are specific to the architectural level (i.e., system testing), implements the sixth and seventh objectives (**O6** and **O7**). Similar to the previous point, laboratory practices and supervised projects are the most prominent methodologies for the learning objectives addressed in this content block.

### 2.2 Teaching methodologies

We now review the different teaching strategies used to convey course content and materialize learning objectives. These methodologies encompass three fundamental axes: conveying knowledge, developing skills, and fostering attitudes.

Each of the following methodologies is applied depending on the type of session. At the Software Architecture course at the Faculty of Computer Science of the Universidade da Coruña, each of the 15 weeks of each teaching semester features a 90-minute class in a lecture room with the entire group, typically consisting of a master class, and a 90-minute session in a working lab with a maximum of 20 students, typically involving hands-on activities:

**Lecture** In lectures, various resources are used for presenting notions and theoretical concepts: blackboard, projection of materials in electronic format, notes in electronic format, and other resources provided through a virtual platform (Moodle). All materials are made available to students before each session.

**Lab Assigments** Hands-on tasks designed to allow students to put their new knowledge into practice as they acquire it, reinforcing their learning.

**Supervised Projects** Individual reinforcement activities for the course content involve the analysis of bibliographic sources, the development of reflective and critical texts, the search for or creation of new resources, etc. These activities

**Table 1: Learning objectives of the Software Architecture course**

| | |
|---|---|
| **O1** | Acquire knowledge of concepts and techniques specific to Software Engineering. |
| **O2** | Interpret typical problems in defining software architectures and the situations in which they occur. |
| **O3** | Define and document specifications and models to promote maintainability and extensibility. |
| **O4** | Apply modeling languages with agility. |
| **O5** | Handle tools for defining and constructing applications. |
| **O6** | Validate the architecture of a system against its requirements. |
| **O7** | Synthesize success stories. |

are proposed throughout the 15 weeks of the semester (although not every week) with a limited timeframe (typically, one week) and are evaluated asynchronously. Students can request further feedback on these projects during personalized tutorial sessions.

Table 2 relates the methodologies we just described to the learning objectives (cfg. Section 2.1).

Given that a semester typically lasts 15 weeks, a balanced distribution of the 150-hour workload corresponding to the 6 ECTS credits implies about 10 hours per week (including both classroom hours and personal student work). Since, as mentioned, a 90-minute lecture and a 90-minute laboratory practice are planned each week, this results in 7 hours of personal work per week (a little less than 1.5 hours per day).

## 2.3 Evaluation methods

We now describe the components of the academic assessment of the students enrolled in the course:

**Continuous assessment (60%)** The weight of the continued work of the students enrolled in the subject is the element that most contributes to their final grade. This part of their grade is calculated as the sum of two components:

**Lab assigments (40%)** Assessment of the lab work carried out throughout the quarter, both individually and as in groups. The different tasks are valued differently, with 10% corresponding to an initial individual task and 30% to a subsequent group project. This weighting aims to reflect the evident differences in the difficulty and complexity of the two types of assignments.

**Supervised projects (20%)** Assessment of the individual work carried out throughout the semester. Typically, each task proposed as part of this methodology is valued between 0.5 points and 1 point, with between 2–4 tasks proposed each semester.

**Objective test (40%)** A written test, the official final exam scheduled by the university for each course, which assesses the overall knowledge acquired by the students.

Although it may be mathematically possible to pass the course based on the grade obtained in the continuous assessment, it is required to take the final exam to actually do so.

# 3 Adaptation to Large Groups

Given the sustained growth in enrollment numbers, which by the academic year 2022/23 had doubled the expected size class (cf. Fig. 4), the faculty decided to face the risk that the large size of the group would negatively affect the effectiveness of the lecture sessions, particularly during directed discussions where student participation would not maintain the same dynamics (in terms of individual engagement). In light of this risk, a contingency plan was formulated with the aim of increasing engagement through other methodologies: lab assignments and supervised projects. For lab assignments, an early engagement strategy was introduced; for supervised projects, an adapted learning contract.

## 3.1 Early Engagement Strategy

The chosen strategy for *early engagement* in the context of lab sessions was the use of a gamified platform for individual practice (cf. Sect. 2.2): Exercism [7]. From a teaching perspective, the Elixir programming language track on Exercism is a very valuable tool, as it provides:

- Specific-purpose programming exercises of increasing difficulty. The Elixir track currently features 161 exercises at the time of writing this article.
- A syllabus or concept tree of the language. Exercises are organized in a tree-like structure according to the concepts they illustrate, which helps guide the student's progress at all times. At the time of writing this, the syllabus has 57 concepts ranging from the basic types of the language (booleans, integers, floats, …) or its typical control structures (functions, *pattern-matching*, recursion, …) to the most advanced elements, such as complex data structures (*streams*) or processes, agents, and tasks. Each concept has a different number of exercises associated with it depending on its complexity, allowing each person to keep on practicing those they find more difficult (or fun!) with different proposals.
- Automated tests for each exercise, so that the student can validate their solution independently.
- Simple automated feedback for each exercise, where aspects related to good and idiomatic programming practices are suggested to the student for improving their solution once the functional tests pass.
- Public mentoring by the community, or individually and privately upon request.
- Rewards and gamification incentives for students in the form of *badges*, reputation, progress percentages, etc.

This strategy was first applied during academic year 21/22, and the perceived effect was that the use of this gamified platform encouraged active participation from the very beginning of each academic term, with an extremely high follow-up rate (cf. Sect. 4). Each student solved more than 50 programming exercises in the first month, receiving private, personalized feedback after each exercise

**Table 2: Relationship between methodologies and learning objectives in the Software Architecture course**

| Methodology | Learning Objectives |
|---|---|
| Lecture | O2, O3, O7 |
| (guided discussion) | O1, O2, O3, O4, O7 |
| Lab Assignments | O1, O2, O3, O4, O6, O7 |
| Supervised Projects | O1, O2, O3, O4, O5, O7 |
| (documentary source analysis) | O2, O3, O7 |

from the teachers for improving their style and performance using Exercism's individual mentoring. The involvement and dedication of the teaching staff in this personalized attention favored a very high ratio of exercise completion week by week. Since this activity was the individual lab assignment (10% of the student's final mark), that meant that almost the entire group obtained most of that grade, an excellent ROI for their effort in starting up the subject. In other words, given that within a few weeks most students got 10% of the course points, the number of early dropouts diminished significantly.

## 3.2 Adapted Learning Contract Strategy

The chosen approach to introduce the *adapted learning contract* strategy was to expand the 2-4 tasks that were being used in previous academic years to a range of 8-12 possible tasks, with the same quantitative weight as the previous ones. In other words, in this learning contract, the teaching offer was significantly increased without increasing the learning demand. The details of this strategy were as follows:

- Each week, several alternative tasks were offered, which typology could be repeated in different weeks (unlike the previous approach, in which each task was of a different type, and occurred only once in the semester). The (re)appearance of similar tasks provides the opportunity for students who had little success in a previous submission to retry that kind of task, potentially putting into practice teachers' feedback comments to improve their performance; alternatively, students could opt to focus their efforts on the types of tasks that suited their abilities, confidence, and/or learning style.
- The greater distribution of tasks throughout the semester (from 2-4 opportunities in total to several opportunities each week) allowed for unprecedented flexibility and adaptation to each student's preferences and needs based on their particular workload (due to demands from other courses and/or personal or professional circumstances). Moreover, this diversity of preferences and needs translated into a greater temporal distribution of submissions (since a smaller number of students attempted each specific task), which enabled the faculty to review and provide feedback in a timely, more thorough manner.

This strategy was applied from academic year 22/23 onwards. The perception was that, thanks to this adapted learning contract, students were capable of adjusting their effort according to their needs, interests, and personal progress, contributing to a greater involvement and satisfaction with this evaluation methods both at individual level and for the entire group.

# 4 Results

Figure 2 shows the percentage of students not involved in the individual lab assignment during the first weeks of the course, or that exhibited poor performance in doing so. Academic years 19/20 and 20/21, which took place in the context of the Covid-19 pandemic, showed a somewhat specific trend, with fewer students failing the individual lab assignment but a slightly greater percentage of students not even engaging on it. Reading the data on, the effect of the first strategy for adapting the course to the large group, namely introducing the gamified environment, seems significant: failing and not engaging percentages drop by half, and maintain a descendant trend ever since, possibly by the accumulative effect of the application of the second strategy.

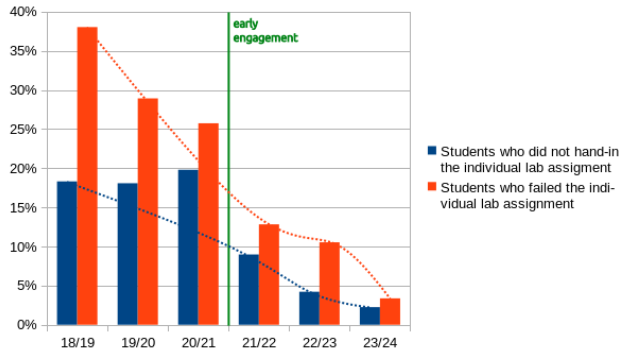


**Figure 2: Impact of the introduction of the gamified platform on students' early engagement**

Figure 3 shows the percentage of students that achieved half, 75%, or all the marks that could be accumulated by completing individual tasks, before and after introducing the adapted learning contract strategy. Once again, academic years 19/20 and 20/21 are considered outliers, since students were confined to their homes due to the Covid-19 pandemic, but the rest of the data reflects a consistent and significant increase in student's performance at all levels.

A detailed analysis of the data related to the implementation of learning contracts (i.e., the tasks effectively carried out by students, their distribution over time, and their performance) led us to observe two interesting facts:

**Observation #1** Students achieve a high percentage of the maximum score corresponding to the tasks in the adapted learning contract, and they do so early rather than later: somehow breaking certain stereotype, they tend not to wait

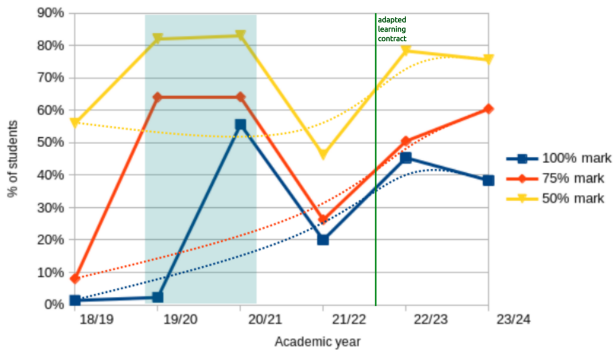


**Figure 3: Impact of the introduction of the adapted learning contract on students' performance at individual tasks**

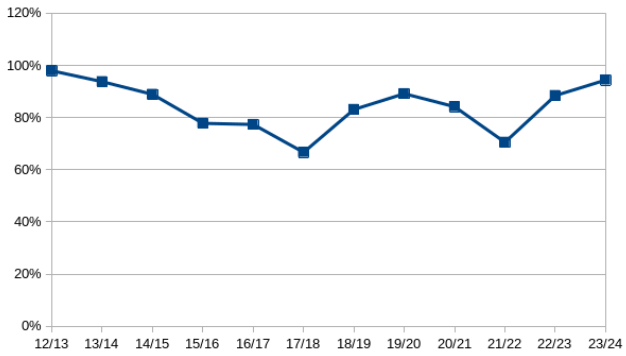


**Figure 4: Software Architecture success rate evolution**

until the last weeks of the semester, but rather engage with the first tasks in greater numbers. Specifically, only between 5% and 12% of students complete the score of their learning contract in the second half of the semester. An analysis derived from this observation is that the chronological order of task types reveals as rather important, as concentrating certain types of tasks at the beginning or end of the semester will result in a different number of students addressing them. Thus, we can say that the adapted learning contract fosters early engagement, and that the order in which tasks are offered is very relevant.

**Observation #2** A percentage of students achieve the maximum score by avoiding certain types of tasks (notably, in our case, diagramming tasks). Specifically, between 23% and 30% of students who achieve the maximum score in the adapted learning contract do so while skipping any task related to C4 usage [15]. We must reflect then on the fact that to prevent students from not completely avoiding tasks of a specific type (especially if they exercise a desirable competence, such as diagramming skills), the learning contract may have to include a certain level of obligation regarding the variety of task types that students must complete. In other words, if there are skills that only one type of task in the learning contract exercises, the contract should establish a minimum number of tasks of that type to be completed by the students.

Last but not least, Table 3 shows the number of students enrolled in the subject each year up to 2022/23, as well as the number of people who took the final exam, and the number of grades awarded in each range. The ranges are the usual ones in the Spanish system: failed [0-5), passed [5-6], notable (6-8], and outstanding [9-10)[1].

As can be more easily appreciated in Fig. 4, overall success rates[2] were very high in the first years of the course, which is somewhat expected given that these were typically highly performant students completing all the courses of each academic year for the first time. The performance seems to later on stabilize in the range of (60%-80%), a result that is not only reasonably positive but also consistent with what is established as a desirable goal by the institutional quality-assurance system.

Aside from the outlier represented by the academic year 2019/20 (corresponding to the COVID-19 pandemic outbreak), as well as the immediately following year, the results obtained in the 2022/23 academic year stand out, precisely the year in which the teaching methodologies exposed in Sect. 2.2 were adapted as explained in Sect. 3. We can therefore say that the success rate achieved thanks to the introduced innovations reaches a value that can only be compared to ten years earlier. In making this comparison, we must take into account that the number of students enrolled in the subject in the 2012/13 academic year was 48, compared to 103 in the 2022/23 academic year, while the number of teachers (and specific docent persons) in the course maintained constant.

We are thus confident in concluding that the teaching innovations introduced as measures to mitigate the effect of group size increase in the Software Architecture subject at the Universidade da Coruña in the 2022/23 academic year not only achieved this objective, but also had a significant impact on improving the subject's success rate. In other words, beyond alleviating the possible negative effect of group size increase on student performance, either directly or through the impact of group size on attendance and participation, as we will discuss in the discussion (cf. section 5), the early implication and formative contract seem to have resulted in a sizeable improvement in student performance.

## 5 Discussion

Determining the extent to which the number of students in a class affects their performance is not a trivial matter. Although there are significant results that support the intuition that large groups have a clear negative effect on the results obtained by their members [13], others suggest that this effect is not really relevant [11], or that it is not uniform when comparisons are made taking into account variables as diverse as gender or even the very subject of study [1]. In contrast, there is indeed consensus around the positive correlation between class attendance and performance [10] (i.e., those who attend class more frequently tend to obtain better results), as well as the fact that a reduced group size tends to favor higher class attendance [10].

[1]Since the regulations for assigning outstanding honors changed over the span of the analyzed years, we have opted not to include this aspect in this analysis.
[2]Universidade da Coruña defines *success rate* as the ratio of students who pass the subject over the total number of enrolled students.

**Table 3: Software Architecture historical grading statistics**

| | 12/13 | 13/14 | 14/15 | 15/16 | 16/17 | 17/18 | 18/19 | 19/20 | 20/21 | 21/22 | 22/23 | 23/24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Students | 48 | 48 | 45 | 36 | 53 | 48 | 71 | 83 | 101 | 78 | 95 | 89 |
| Not pass[a] | 1 | 3 | 5 | 7 | 11 | 16 | 11 | 9 | 13 | 20 | 11 | 6 |
| Pass | 5 | 8 | 15 | 14 | 22 | 5 | 19 | 36 | 28 | 29 | 24 | 24 |
| Notable | 38 | 33 | 24 | 13 | 15 | 24 | 36 | 32 | 52 | 23 | 54 | 53 |
| Outstanding | 4 | 4 | 1 | 1 | 4 | 3 | 4 | 6 | 5 | 3 | 6 | 5 |

[a] Includes students not attending the compulsory final exam, as well as failing students.

On the other hand, we know that early participation or involvement in a subject has a significant positive impact on student performance [14], both due to the improvement of the learning experience [6] and emotional investment [8]. Moreover, it is precisely the students with fewer skills who benefit the most from increased early participation [4]. Regarding learning contracts, their effectiveness as an educational tool has been known for a long time [2]. Notably, they empower students by allowing personalized adaptation according to their needs [9].

Both early engagement and learning contracts have been used on multiple occasions in higher education, including in Computer Science studies. In the state of the art we find, for example, positive results using learning contracts in Software Development projects [3], or engagement strategies in subjects such as Software Engineering or Software Testing [5]. While all these experiences aim to improve academic performance, we have not found previous work that uses these methodologies as a positive action designed by the teaching staff in response to an increase in group size. In other words, we are not aware of any precedent efforts that illustrate the effectiveness of these teaching methodologies as a tool to mitigate the effects of a significant increase in the number of students enrolled in a course.

## 6 Conclusions

This article presents an experience of two very successful teaching strategies introduced in a course on Software Architecture (3rd year, degree in Software Engineering), following a consistent upward trend that had doubled the number of enrolled students to twice the size for which learning and evaluating methodologies had been previously designed for. Specifically, we have detailed the two teaching strategies which introduction resulted in achieving a 92% success rate:

- **Early Engagement**. Using a gamified platform, active participation was encouraged from the very first weeks of the course. Each student solved more than 50 programming exercises in the first month, receiving personalized feedback after each exercise to improve their style and performance. The involvement and dedication of the teaching staff in this initial mentorship favored high participation and weekly completion rate of exercises.
- **Adapted Learning Contract**. The choice of individual tasks for concept reinforcement was significantly increased from a few tasks (2-4) to a handful (10-12) each semester, maintaining their potential contribution towards the student's final qualification. Each student could select, from the wide choice of possible tasks, those that best suited their abilities, preferences and/or workload. The distribution of task deadlines throughout the semester allowed the teaching staff to review and provide feedback to students in a timely manner, and students to adjust their effort according to their progress, contributing to an increased individual and group performance. Somewhat unexpectedly, the adapted learning contract further fostered early engagement, which makes the order in which tasks are offered relevant (since early-on tasks had significantly more takers than late-in-the-semester ones). Also, if there are skills that only one type of task in the learning contract exercises, the contract should establish a minimum number of tasks of that type to be completed by the students.

Given the interesting insights we have gained from analyzing the impact of the adapted learning contract in particular, we plan to further research student's motivations in order to better adjust the learning contract specification. Early engagement with the adapted learning contract may be related to student's anticipation of an increase in task difficulty as the semester progresses, or to a lighter workload at the beginning of the semester. Surveying the current state of the art on the effectivity of different pedagogical adaptations to learning contracts in the case of compulsory skills is also among our next steps.

## References


[1] Ethan Ake-Little, Nathaniel von der Embse, and Dana Dawson. 2020. Does Class Size Matter in the University Setting? *Educational Researcher* 49, 8 (2020), 595–605. https://doi.org/10.3102/0013189X20933836

[2] Geoff Anderson and David Boud. 1996. Introducing Learning Contracts: A Flexible Way to Learn. *Innovations in Education and Training International* 33, 4 (1996), 221–227. https://doi.org/10.1080/1355800960330409

[3] Malcolm Birtle. 1998. Negotiated learning contracts in team projects. *Annals of Software Engineering* 6 (1998), 323––341. https://doi.org/10.1023/A:1018961516018

[4] Robert M Carini, George D Kuh, and Stephen P Klein. 2006. Student Engagement and Student Learning: Testing the Linkages. *Research in Higher Education* 47 (2006), 1—-32. https://doi.org/10.1007/s11162-005-8150-9

[5] Peter J Clarke, Mandayam Thirunarayanan, Sai Chaithra Allala, Juan Pablo Sotomayor, and Monique S Ross. 2020. Experiences of Integrating Learning and Engagement Strategies (LESs) into Software Engineering Courses. In *ASEE Virtual Annual Conference Content Access*. https://doi.org/10.18260/1-2--34630

[6] Charlene Gerber, Nadia Mans-Kemp, and Anton F Schlechter. 2013. Investigating the moderating effect of student engagement on academic performance. *Acta Academica: Critical views on society, culture and politics* 45, 4 (2013).

[7] Independent, community funded, not-for-profit organisation. 2012. Exercism: Get really good at programming. Disponible en https://exercism.org.

[8] Jung-Sook Lee. 2014. The Relationship Between Student Engagement and Academic Performance: Is It a Myth or Reality? *The Journal of Educational Research* 107, 3 (2014), 177–185. https://doi.org/10.1080/00220671.2013.807491

[9] Catherine M. Lemieux. 2001. Learning contracts: Tools for empowerment and accountability. *Social Work Education* 2 (2001), 263–276. https://doi.org/10.1080/10.1080/02615470120044347

[10] Anna Lukkarinen, Paula Koivukangas, and Tomi Seppälä. 2016. Relationship between Class Attendance and Student Performance. *Procedia - Social and Behavioral Sciences* 228 (2016), 341–347. https://doi.org/10.1016/j.sbspro.2016.07.051

[11] Pedro S Martins and Ian Walker. 2006. Student Achievement and University Classes: Effects of Attendance, Size, Peers, and Teachers. *IZA Discussion Paper* 2490 (2006). https://doi.org/10.2139/ssrn.955298

[12] The IEEE & ACM Joint Task Force on Computing Curricula. 2004. Computer Engineering. Curriculum Guidelines (...). Disponible en http://www.acm.org/education/education/curric_vols/CE-Final-Report.pdf.

[13] Pedro J. Cuestas Pedro. Fenollar, Sergio. Román. 2010. University students' academic performance: An integrative conceptual framework and empirical analysis. *British Journal of Educational Psychology* 77 (2010), 873–891. Issue 4. https://doi.org/10.1348/000709907X189118

[14] Ronald L. Skidmore and Lola Aagaard. 2004. The Relationship between Testing Condition and Student Test Scores. *Journal of Instructional Psychology* 31 (2004), 304–313.

[15] Andrea Vázquez-Ingelmo, Alicia García-Holgado, and Francisco J García-Peñalvo. 2020. C4 model in a Software Engineering subject to ease the comprehension of UML and the software. In *IEEE Global Eng. Education Conference*. 919–924. https://doi.org/10.1109/EDUCON45650.2020.9125335