



Modularization for mastery learning in CS1: a 4-year action research study

Claudio Alvarez^{1,2} · Maira Marques Samary³ · Alyssa Friend Wise⁴

Accepted: 24 February 2023

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2023

Abstract

Computer programming is a skill of increasing importance in scientific and technological fields. However, in introductory computer science (CS1) courses in higher education, approximately one in every three students fails. A common reason is that students are overwhelmed by an accelerated and inflexible pace of learning that jeopardizes success. Accordingly, in the computer science education literature it has been suggested that the pedagogical philosophy of ‘mastery learning,’ which supports students progressing at their own pace, can improve academic outcomes of CS1 courses. Nevertheless, few extended mastery learning implementations in CS1 have been documented in the literature, and there is a lack of guidance and best practices to foster its adoption. In this paper, we present a four-year action research study in which a modular mastery-based CS1 course was designed, evaluated and improved in successive iterations with cohorts of engineering freshmen in a Latin American research university ($N=959$). In the first year of the intervention, only 19.3% of students passed the course in their first semester attempting it. In successive iterations, the instructional design, teaching and learning activities, course content, and course management were iteratively improved such that by the fourth year of offering 77.1% of students passed the course in their first semester. Over this period, course attrition was reduced from 25.0% to 3.8% of the cohort, and students’ mean time spent in the course decreased from 23.2 weeks ($SD=7.38$) to 14.9 ($SD=3.64$). Results indicate that modularization for mastery learning is a viable approach for improving academic results in a CS1 course. Practical considerations towards successful implementation of this approach are presented and discussed.

Keywords CS1 · Mastery learning · Modular course · Action research

✉ Claudio Alvarez
calvarez@uandes.cl

Extended author information available on the last page of the article

Introduction

Introductory computer science and programming courses at the undergraduate level (referred to as CS1 courses) experience failure rates near 30% in multiple student populations worldwide (Bennedsen & Caspersen, 2019). This occurs in a global context in which there is an increasing demand from industry for software engineers, computer scientists, and STEM professionals with proficiency in computational thinking and programming (CompTIA, 2021; Eurostat, 2022). Experiences in different educational systems around the world show that students' exposure to computer science at the K-12 level influences their decision to pursue computing in higher education (Armoni & Gal-Ezer, 2014; Kurhila & Vihavainen, 2015; Webb et al., 2017). Yet despite the increasing importance of these skills in diverse professional and scientific fields, it is not compulsory in many education systems to teach computational thinking and programming at the K-12 level (Bocconi, et al., 2016). For example, a recent report documented that only 51% of high schools in the United States offer at least one foundational computer science course (Code.org, CSTA & ECEP Alliance, 2021). Students in the developing world find even more limited opportunities to learn the foundations of computer science at the K-12 level (Vegas et al., 2021). Thus, students who enroll in STEM curricula in higher education must typically learn computational thinking and programming at an accelerated pace in their first introductory course to the subject (Ahadi, et al., 2014). When this is done via traditional 'one size fits all' instructional approaches, students experiencing learning difficulties in introductory programming courses are rapidly overwhelmed by the pace of instruction, resulting in early attrition and high rates of course failure (Patitsas et al., 2019; Robins, 2010).

Mastery learning is a pedagogical philosophy and instructional approach that has been researched since the 1960's (Bloom, 1968; Keller, 1968), based on the assumption that every student can master a given skill if given enough time, instruction and support. In contrast with traditional instruction, which sets a standard time period (such as a quarter or a semester) in which students are expected to learn the materials, mastery learning sets standards for mastering material that students are expected to achieve at their own pace. In the computer science education literature, it has been suggested that mastery learning permitting students to learn at their own pace can help them make sustainable progress in a CS1 course (Robins, 2010). This stands in contrast to the common occurrence of students being unable to keep up with the pace of instruction and risking course failure (Petersen et al., 2016) due to a combination of cognitive (Robins et al., 2019), meta-cognitive (Liao et al., 2019) and motivational factors (Dorn & Tew, 2015).

Despite the potential benefits, adoption of mastery learning in computers science education has remained limited, and few studies describe its implementation and evaluation. A recent review by Garner et al., (2019) reports that there is a lack of documented attempts to implement mastery learning in CS1 courses and that general guidelines and best practices on how to do so have remained

unavailable. Thus, there is a pressing need to explore how to best implement mastery learning in CS1 courses, establish whether these efforts lead to improved learning, and if so, derive guidelines and recommendations that can be helpful for computer science educators interested in adopting mastery learning in their own contexts.

In this paper, we report on the development of an introductory computer programming course based on mastery learning at a research-based university in Latin America. The process was based on iterative cycles of an action research methodology over a four-year period, involving a total of 959 students. The mastery learning course was designed to fit a semester-based curriculum and academic calendar. To accomplish this, a modular instructional design approach was adopted; that is, one in which the course is broken up into multiple discrete, short duration segments of material and learning activities, known as modules (Jenkins & Walker, 2014; Dochy, et al., 1989; Goldshmid & Goldshmid, 1973). Considering mastery learning's philosophy, students complete a module and are promoted to the next only after demonstrating mastery of the respective learning goals, through both low and high-stakes assessments. Students can retake a module they failed without necessarily failing the entire course. Consequently, multiple different module progressions are possible in the course, reflecting students' varying prior preparation, aptitude and efforts to learn computer programming. In turn, the modular structure facilitates teachers in adjusting content and/or instruction according to students' needs.

The following section presents the theoretical background for this work, followed by the research questions. Subsequent sections describe the design, implementation and progressive improvement of pass rates and attrition over the four year-long iterations of the course. The discussion section presents a holistic view of the challenges and difficulties experienced in the development of the course, and practical recommendations drawn from lessons learned in the process. Finally, conclusions and avenues for future research are presented.

Theoretical background

Academic failure in CS1

Failure in CS1 courses has been a topic of attention in the academic literature for the past several decades (e.g. Bennedsen & Caspersen, 2007, 2019; Watson & Li, 2014) with studies consistently reporting failure rates close to 30% on average. Bennedsen and Caspersen (2019) report a 28% average failure rate in CS1, with variations between geographic regions and by type of institution, i.e., college or university. For example, the combined rate of students aborting, skipping or failing the course in universities is 29%, while this figure for colleges is 17%. In Asia, the average failure rate is 29%, in Europe it is 31% and in North America it is 24%. It can thus be affirmed that in much of the world and across different types of institutions, approximately 20 to 30% of students fail their first programming course. Improving the academic results of CS1 courses can reduce student attrition, increase graduation rates, and increase the supply of qualified workers in scientific and technological

fields to meet the growing demand. A wealth of research in computer science education has sought to explain why some students perform better than others in CS1 courses (Basnet et al., 2018; Guzdial, 2019; McCartney et al., 2017; Patitsas et al., 2019). One hypothesis for academic failure in CS1 courses relates to the cumulative nature of the materials and posits the existence of “stumbling points” in the learning path of programming: that is, there is a small number of identifiable skills and concepts that can have a major impact on a student’s progress (Ahadi & Lister, 2013). The implication of this is that students failing to master critical skills at key points in their learning will likely experience severe difficulties later in their learning path. A more general theorization of this idea is seen in Robins’ (2010) theory of learning edge momentum (LEM) as an explanation for students’ variability in learning in CS1. LEM proposes that with the tight integration of concepts and skills in the domain of computer programming, failure to learn concepts becomes self-reinforcing, thus creating momentum towards unsuccessful outcomes. Conversely, successful learning is positively reinforcing and creates momentum towards more successful outcomes.

The implication of both stumbling points and LEM for traditional one-size-fits-all instruction in CS1 courses is that in every cohort, students who fail to master fundamental concepts and skills are substantially more likely to fail to learn more advanced and/or composite knowledge. The solution to this problem lies in allowing students to progress to more complex knowledge only after they have mastered the earlier knowledge that is required. This demands instructional design and pedagogy that departs from the overarching constraint of a single fixed timeline (i.e., the monolithic rate and path of progression in a traditional course; Robins, 2010), thus allowing each student to progress according to their own ability.

Mastery learning in CS1

Bloom (1968) developed the mastery-learning approach based on the observation that providing all students with the same amount of time to learn and the same instruction resulted in a distribution of student achievement that reproduced the distribution of students’ initial aptitudes, i.e., a normal curve. Carroll (1963) argued that if individual students were provided with the time they needed to learn the material, most students should be able to eventually master it. The shift from fixing learning time to fixing a standard level of achievement is the driving feature of mastery learning (Emery et al., 2018).

Mastery learning emphasizes the need for constant tutoring, feedback and proctoring, which is difficult to scale to large cohorts (Fox, 2004). Keller, (1968) recommended a maximum cohort size of a hundred students due to practical difficulties of supervising student proctors. Furthermore, since in mastery learning can decide when to take the assessments, the approach was found to cause students to procrastinate and prioritize engaging in other academic activities. In traditional university contexts, where students are under pressure to study several subjects simultaneously, the approach might be difficult for some, especially first year students new to the independence required in the university context. These students often lack the

self-regulation skills necessary to organize their time and the dedication needed to be successful in both mastery-based and traditional courses (Eyre, 2007).

Despite the relevance and potential value of mastery learning for computer science education, literature specific to this domain is scarce, and the limited efforts to adopt mastery learning in computer science have been carried out in isolation of each other. Recently, Garner and colleagues (Garner, et al., 2019) conducted a review of mastery learning in Computer Science, including literature published between 1992 and 2017. Considering twelve studies in this time frame, it was identified that motivators for implementing mastery learning include the possibility of improving teaching for diverse student cohorts, addressing the 'learning edge momentum' problem, guaranteeing skill mastery, and contributing to the mastery learning literature as well as other case specific reasons. Obstacles to the implementation of mastery learning include student procrastination, scaling delivery and developing assessments. Many experimental approaches have been observed in distance learning, automated assessments, personalized feedback, and each program has had to solve similar problems. Moreover, few studies focus on *introductory* computer science courses; only five studies were reviewed, with a duration no longer than two semesters. The review concludes that implementation of mastery learning in computer science has been based on the general education literature, and that there is still a lack of an authoritative body of scholarly knowledge to guide curricular innovation, implementation and research of mastery learning in computer science.

Course modularization for mastery learning in CS1

The notion of dividing a full-semester course into discrete modules has been discussed by many authors beginning in the 1960s (Goldshmid & Goldshmid, 1973). Modularization emerged as a methodology for curricular and instructional design in European higher education, emphasizing the possibility for students to have a high degree of customization in their curriculum (e.g., presenting students with catalogs of modules with which to configure their courses) and to develop autonomous learning skills (Dochy, et al., 1989; van Eijil, 1986; van Meel, 1993). Nonetheless, there are a variety of applications and ways of implementing modularization according to the characteristics of the target educational problem and context (Jenkins & Walker, 2014).

For the present effort, modularization of a computer programming course offers the possibility to implement mastery learning in a scalable way, within the constraints of the semester-based academic activities typical of higher education institutions. A modular design permits dividing a traditional course into modules. Each module has its own learning goals and students can work on the appropriate module according to their mastery level. From the operationalization standpoint, modules can be taught multiple times each semester, performance outcomes can be quickly tested and evaluated, and teaching can be continuously improved in terms of instructional strategies and assessment methods. With regard to the common mastery learning issue of procrastination (Fox, 2004), a modular design that prescribes dates

of both formative and summative assessments can drive students to focus their attention on the course and strive to meet the required mastery standard in a predictable time frame. With this rationale and expectations, the present authors persuaded faculty leadership at a research-based Latin American university to support transformation of an introductory programming course into a modular format for mastery learning.

Research framework and questions

Figure 1 depicts the methodological framework underpinning the present study, supporting the iterative process towards implementing and improving a modular CS1 course for mastery learning. The framework comprises a four-step action research methodology as described by Cohen et al., (2013) drawing on Zuber-Skerritt, (2003). The action research process focuses on improving relevant aspects of teaching and learning as identified by Biggs, (2011) in his ‘*constructive alignment*’ framework, which include the course’s intended learning objectives, its teaching and learning activities, and assessment tasks (see components A–C in Fig. 1). In addition, instructional design, course content and skills, and course management are three aspects identified by Fink, (2013) which we consider central in the methodology of the present research (see components D–F in Fig. 1). Given these components of interest, the steps of the action research cycle include (1) strategic *Planning*, wherein the A–F components are designed, planned, and revised according to relevant learning theories, and a research agenda aligned with the educational problem is crafted, (2) *Action*, i.e., plan enactment, (3) *Observation*, evaluation and self-evaluation of the course implementation with regard to components A–F, and (4) critical and self-critical *Reflection* on the results of points 1–3 deriving in decision making for the next action research cycle.

Based on the discussion presented in the previous sections, and the methodological framework that underpins the current research, the research questions addressed in this paper are the following:

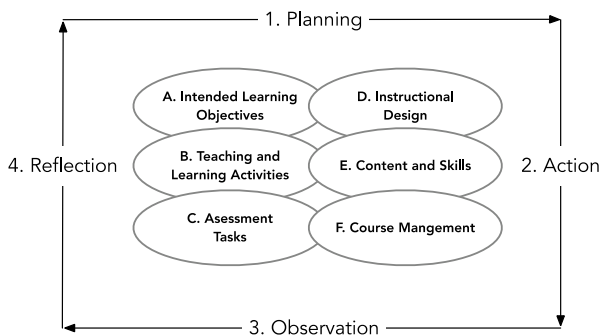


Fig. 1 Action research cycle towards implementing and improving the modular course for mastery learning

1. What challenges and difficulties arise in the process of implementing a CS1 modular course for mastery learning with regards to the instructional design, content and skills, teaching and learning, assessment and course management, and how are these overcome in successive action research cycles?
2. How do students perform in the CS1 modular course for mastery learning? How diverse are their module progressions and how do these evolve as a result of interventions conducted within the action research process?
3. What practical implications and recommendations emerge from the experience to support successful implementation of modular CS1 courses for mastery learning?

Educational context

The research was conducted in the engineering school of a Latin American research university. The engineering school offers undergraduate programs in five areas: Industrial, Civil, Electrical, Computer Science and Environmental Engineering. As of 2020, only 7.6% of the students enroll in the Computer Science program, however, the CS1 course is compulsory for all first year engineering students, regardless of program. The intended learning objectives of the course are that by its completion students should be able to:

- Explain key concepts involved in procedural computer programs, namely: input and output, data types, variables, operators, control flow, function invocation and return values, random access data structures (i.e., simple and multidimensional arrays), strings, dictionaries, file access, and recursion.
- Model algorithms based on a procedural model of computation, specifying the sequence of steps required by use of a graphical language.
- Model simple recursive algorithms comprising base cases and recursive steps.
- Write procedural programs in a high-level, text-based programming language, including the above-listed concepts.
- Write programs that involve numeric computation and data visualization capabilities based on 2D plots.

Previous version of the CS1 course

The version of the CS1 course that existed prior to the present study was based on a semester format comprising the following weekly activities: two lectures, a recitation session, and a lab assignment. Assessment was based on weekly low-stakes assessments during lecture hours, weekly graded lab assessments and homework. In addition, the course incorporated a final exam worth of 30% of the final grade. It was customary to offer students who did not pass the exam but still reached a certain grade cutoff the ability to take a recovery exam and still pass the course. Under this scheme, the failure rates in the previous version of the course fluctuated between 30 and 40% in the last three years of its implementation prior to the study.

Student cohorts under study

The student cohorts that participated in the current study are described in Table 1, including their size and gender composition. Annually, student cohorts are comparable with an average of 80% male and 20% female students. The secondary education origin of students has also remained stable, with 78% to 83% coming from private secondary education institutions. Another 8–14% of the students come from Publicly-Funded Private Schools (PFPS). A minority of the students, less than 5%, come from public schools. In the national context, 37% of high school students attend a public school, 48% attend PFPS schools and 14% attend private schools. In addition, only 33% of students that attend public schools enroll in higher education, while 76% of students who attend private schools enroll in higher education (OECD, 2015).

Course design features

The course developed in this study is based on the following design features, which respond to the defined learning outcomes and a mastery learning approach:

1. **Modular course architecture for mastery learning:** The intent of the modular architecture is to permit students to follow different paces of module progression in the course according to their learning capabilities, while achieving mastery of each relevant course unit (module) before being promoted to the next module. For students to demonstrate attainment of mastery, each module includes a summative examination at the end. To make the course must be operational within the structure of a semester-based calendar in the host institution with sixteen weeks of classes per term, it was chosen to structure the course into four successive

Table 1 Student cohorts under study

Cohort	2017	2018	2019	2020	Total
Size	228	242	227	262	959
<i>Gender</i>					
Male	183 (80.3%)	190 (78.5%)	188 (82.8%)	200 (76.3%)	761
Female	45 (19.7%)	52 (21.5%)	39 (17.2%)	62 (23.7%)	198
<i>Secondary education institution type</i>					
Private	171 (75.0%)	186 (76.9%)	183 (80.6%)	219 (83.6%)	759
PFPS	31 (13.6%)	32 (13.2%)	22 (9.7%)	21 (8.0%)	106
Public	10 (4.4%)	5 (2.1%)	10 (4.4%)	11 (4.2%)	36
Other	16 (7.0%)	19 (7.9%)	12 (5.3%)	11 (4.2%)	58

modules with equal duration of three weeks. This makes possible teaching the modules in parallel, and the students can repeat a module in the first semester and still be able to complete the course in the term. Also, students do not need to wait to resit a module if they fail, as parallel modules can start and end in the same dates.

2. **Foster algorithm design skills before coding:** According to Koulouri et al. (2014), learners' analytic capability to logically decompose problems has a positive effect on learning programming, regardless of the programming language being used in the process. In the previous version of the course, problem analysis and algorithm design skills were taught as the initial content unit in the first week of the course. Arguably, these skills were not given sufficient attention as they were taught superficially, and only evaluated by low-stakes assessments. Thus, for the modular course, it was considered that time dedicated to problem analysis and algorithm design needed to be increased, as well as the need to adopt a more rigorous assessment of these skills.
3. **Imperative procedural programming:** For decades there has been debate about the order in which programming skills should be taught (Luxton-Reilly, et al., 2018); that is, if object-oriented analysis and design should be taught first, i.e., an "objects-first" approach, or if prior to that, students must master procedural programming. Faced with these alternatives, the present authors had developed the preceding course based on an imperative-procedural approach, emphasizing top-down problem analysis, and algorithm design based on the use of variables, logical and arithmetic operators, domain of flow control structures, use of functions, arrays and dictionaries. The modular course here presented maintains this tradition, with object-oriented analysis and design skills being taught in a later course in the curriculum.
4. **Performance-based assessment:** Assessment sends a strong message to students about what counts as knowledge, insofar students' perceptions of the requirements of the assessment influence their approach to learning (Ott et al., 2016; Weurlander & Soderberg, 2012). The current authors consider that performance-based assessment can therefore foster deep learning of course topics, as students are required to produce working programs in their solutions to given problems, and in this process, integrate various programming skills. Therefore, assessment problems in the modular course are based on 'multistructural-applying' and 'creating' categories of the taxonomy proposed by Meerbaum-Salant et al., (2013).
5. **Situated problems:** A student's intrinsic motivation for computer programming can be influenced by their perceived real-world applicability of knowledge and skills (Dorn & Tew, 2015; OGrady, 2012). This is salient in educational contexts in which students have diverse interests, as is the case in the current study's educational context. Furthermore, from a constructivist standpoint, problem understanding can be facilitated if students can anchor problems to their real-world experiences and prior knowledge (Ben-Ari, 2001). Thus, both practice and assessment tasks in the modular course involve problems in which students are encourage to 'think as engineers' in a particular scenario.
6. **Pursuit of Constructive Alignment:** The theory of constructive alignment (Biggs, 2011) is based on the idea that learners use their activity to construct knowl-

edge as interpreted through their own existing schema, and that assessment tasks should be aligned to what it is intended to be learned. The instructor's duty is to set up a learning environment that encourage students to perform learning activities that align with the intended outcomes, and to assess student performances against the latter. The relevance of constructive alignment, its implications and potential enhancement of teaching and learning have been emphasized by some authors in computer science education literature (Bayu Bati et al., 2014). The modular course here proposed aims at achieving constructive alignment among learning outcomes, teaching and learning activities and assessment tasks.

Course iterations

Iteration 1

Planning

A design of four successive Modules (M1-M4) of three weeks each was adopted in both semesters of the academic year (see Appendix A a). The teaching staff was led by a coordinator (first author of the present study), and included four lecturers. The coordinator planned the course schedule, as well as the evaluation and course material development plan. All teaching staff had to contribute to the development of problem sets for tutoring sessions, lab assignments, and exams during the semester, according to the material development plan. The course coordinator monitored these duties and oversaw the distribution of the materials in addition to their work preparing the lecture material (which included slides sets and code samples). Grading was conducted by 20 Teaching Assistants (TAs), directed by a chief TA who reported to the course coordinator. The number of lecturers and TAs was maintained from the traditional course.

An example of how the modular course can evolve throughout an academic year is shown in Fig. 2. The horizontal axis shows the division of the academic year (and semesters) into Time Blocks (TBs). A TB within a semester spans for three weeks, so five TBs are commonly allotted within sixteen weeks of classes in a semester. The vertical axis denotes the modules that are active (given) in the TBs. Initially,

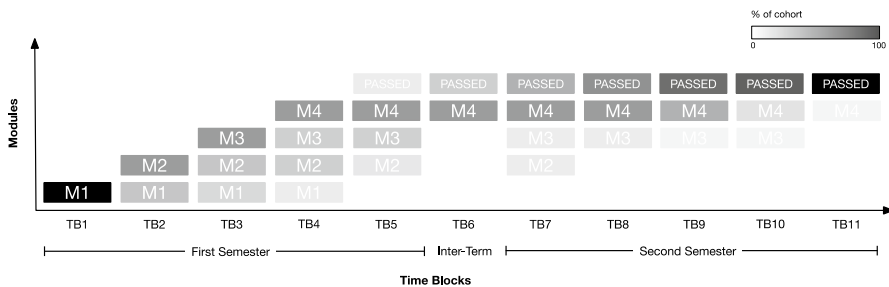


Fig. 2 Modular course evolution example

all students start at M1 in TB 1. As students pass and fail modules through the TBs, different modules need to be taught in parallel. By the end of TB 4, a part of the cohort may have completed all four modules, thus being the first students to pass the course, accomplishing so in 12 weeks. The rest of the students continue to complete the course in successive TBs, thus taking a varying amount of TBs to pass. From one TB to the next, lecturers can be reassigned by the course coordinator to teach a different module, as the number of students requiring attending each module varies from one TB to the next. For planning purposes, and considering classroom capacity restrictions, the number of lecturers allocated to teaching each module had to be decided before the start of every TB, as module pass rates were unknown in the first iteration of the modular course.

In Iteration 1, similarly to the example shown in Fig. 2, an inter-term period (TB 6) was planned to offer only the final (fourth) module of the course, and thus allow students with the last pending module to complete the course before the following semester. Enrollment in the inter-term period was made optional. It was expected that by the end of the first semester, most of the students would have reached or passed M4. In the second semester the modular course was continued for an additional five three-week TBs (i.e., TBs 7–11). Students that were not able to pass the course in the first semester were allowed to resume the course starting with the module following the highest one they had passed.

In each assessment, as well as for calculation of final module grades, the grading system followed local conventions using a continuous scale from 1.0 to 7.0, with a passing level of 4.0 (50%). Grades are absolute and not curved.

Action

The structure of all course modules across iterations, including teaching and learning activities, and assessments, is depicted in Appendix A. The first two weeks of classes each included a lecture, a tutoring session and a lab assessment. Between the lecture and the tutoring session, students were expected to dedicate self-directed study time to solving a homework assignment and solving practice problems. In the third week of the module, the students sat the summative examination of the module instead of a lab assessment. Table 2 shows assessments and their weights per each module, in the first three iterations of the course.

The contents of the course modules are shown in Appendix B, along with their evolution throughout successive course iterations. In the first iteration, M1 encompassed the basics of computational thinking, including problem analysis and algorithm specification. The students were taught the latter skills through drawn flow charts. Then, computer programming based on Python 2.7 was introduced, including basic input–output, variables, operators, pseudo-random numbers and the selection statement. In M2, more advanced flow control (including loops and flow control nesting) and functions were taught. M3 introduced lists, nested lists, and file access. Finally, in M4, other data structures were taught, together with Python’s numeric computation library (NumPy), and 2D plots.

Following common recommendations in the mastery learning literature (Fox, 2004), lectures were intended as motivational with the aim to provide students with

Table 2 Course assessments and weights in the first three course iterations

Assessment type (per module)	Iteration 1–2017		Iteration 2–2018		Iterations 3–2019	
	Amount	Weight	Amount	Weight	Amount	Weight
Lab Assessments	2	15% each	2	7.5% each	2	7.5% each
Final Exam	1	70%	1	70%	1	70%
Homework	1 (M2, M3, M4)	Optional *	1 (M1-M4)	7.5%	1 (M1-M4)	7.5%
Attendance	–	–	1	7.5%	1	7.5%

*Homework grades were only considered as a bonus, if these raised the students' final average after completing all four course modules

key concepts and demonstrations to get started learning on their own. Because of this, lecture time was reduced to 50% of the time that was allotted in the former course. After the lectures, students were provided with readings, videos, and problem sets to support their personal study. Then, in tutoring session, students were presented with a set of increasingly complex problems, which they could solve supported by a TA and peer collaboration. These problems were not graded. The following day, the students had to attend a graded laboratory assignment (weeks one and two) or the final module exam (week three). In the first iteration of the course, in M2 to M4, the students had a homework assignment lasting seven to ten days. These assignments were optional, and counted in the final course grade with 16.6% weight if their average surpassed the weighted average of other course assessments.

With the intent to provide students with instant feedback on their performance, it was decided to implement automatic grading in most of the lab assignments. For this, contest management system (CMS; Maggiolo & Mascellani, 2012), a platform utilized in competitive programming environments, such as the International Olympiad in Informatics, was tailored to the course's needs. Following the intent to give assessments of a situated nature, problem statements each described a particular scenario. The problem itself was divided into subgoals of increasing difficulty, each involving a number of test cases. Examples were provided for each subgoal, showing the corresponding data inputs and outputs. Upon submitting a response, each successful test case awarded points to the student.

The final summative exam in each module required students to demonstrate mastery of the knowledge and skills seen in the course up to that time. Problem statements described a scenario and functional requirements of the solution to be implemented. Students were always asked to write a working program that solved the given problem. Grading of examinations was always conducted by the TAs following a common rubric.

Observation

Students' performance in the first iteration of the course was well below expectations at the outset of the intervention. By the end of the first semester, in TB5,

Table 3 Proportion of students in each course state by the end of the first (TB5) and second (TB13) semesters in iteration 1

	M1	M2	M3	M4	Passed	Dropped out
End of first semester (TB5)	.07	.13	.37	.21	.19	.03
End of second semester (TB13)	.00	.00	.00	.04	.75	.21

Table 4 Module pass rates for students' first, second and third attempts in their first three instances in iteration 1

Instance	M1	M2	M3	M4
First	.38 (228)	.34 (88)	.50 (30)	1.00 (15)
Second	.56 (140)	.79 (137)	.37 (123)	.63 (46)
Third	.39 (61)	.53 (62)	.27 (110)	.72 (32*)

*Corresponds to the inter-term period, wherein only 32 students out of possible 47 opted in

there were students still in all four course modules (see Table 3), and only a small fraction of students had passed the complete course.

In the first semester, pass rates were alarmingly low in M1 and M2, especially the first time students attempted each module (see Table 4). The first time M1 was given (i.e., in its first *instance*), only 37.6% of the cohort passed, thus the majority of the students had to repeat it (i.e. *attempt* passing the module a second time). Given the low pass rates in modules 1 and 2, only a small percent of the cohort of students were able to reach modules 3 and 4 the first time they were offered. This smaller number of (high-achieving) students can help explain the higher student success in these modules in their first offering.

Students' average time to pass all four course modules in the first iteration was 23.2 weeks (SD=7.38), close to eight TBs, considering both semesters and the inter-term period. The course pass rate in the first semester was only 19.3%, but including students who passed M4 in the inter-term period, this figure increased to 28.5%. Thus before the start of the second semester, slightly below a third of the original cohort had finished the course.

At the start of the second term 46.9% of the original cohort was in M3 and M4, and 10.1% of the students had dropped the course. The number of students that passed the course increased steadily in the second semester, reaching 66.7% by the end of TB 11. As in the previous semester, M4 was offered in an extra time block (TB12) in weeks 33–35. This increased the overall course pass percentage to 75% (see Table 4). The overall percentage of students who dropped the course doubled in the second semester, reaching 21%. Nearly half of this increase (i.e., 5.7%) was due to students failing to pass M3 in the last TB it was given (TB11). Lastly, a remaining 4.4% of the cohort in M4 did not drop but was unable to pass the course by the end of the year.

Reflection

The reflection step in the first iteration involved meetings with course TAs and the teaching staff, and conducting an interview process with students. Purposeful sampling was used, selecting ten students with different levels of achievement in the course, including students who had passed the course in the first semester, students who had failed the course in the first semester, and students who dropped out of the course. Based on these information sources, different issues were noted, which provided explanation for the poor results obtained in the first iteration of the modular course. The issues are summarized in Table 5.

One of the main issues found was that the methodology used in M1 to introduce computational thinking and algorithm design did not meet the expected results. Table 4 shows meager M1 pass rates in its first three instances. In the light of academic results, and students' testimonies in interviews, it was considered that both pedagogy and assessment required major changes (e.g., see issues AS1, CSK1 in Table 5).

Iteration 2

Planning

In Iteration 2 the schedule of the academic year remained similar to the previous iteration, however, due to limitations of the academic calendar, it was not possible to accommodate six TBs in the second semester but only five. The teaching team remained stable except for a lecturer who was replaced by a doctoral student with prior teaching experience in a traditional CS1 course. With regard to course content, it was decided that rework of M1 was necessary, considering the need to adopt more effective ways to teach the initial skills of the course. In planning the second iteration of the course, most of the issues described in Table 5 were addressed through a decision making process in which the course coordinator consulted the teaching staff, including both professors and teaching assistants, for ways to overcome each issue (Table 6).

Action

A major part of course improvement efforts in Iteration 2 were directed at addressing the pedagogical issues in M1. To deal with these, it was decided to adopt the approach of teaching visual programming with interactive tools instead of traditional hand-drawn flowcharts, before introducing text-based programming. Two block-based visual languages were considered, including code.org (Kalelioğlu, 2015) and MIT Scratch (Meerbaum-Salant et al., 2013). Scratch was considered more convenient as it focuses strictly on visual programming, while

Table 5 Issues found in iteration 1

Component	Issue found	Description
Teaching and learning	(TL1) Same lectures for both repeaters and newcomers	Repeating students along with those taking the module for the first time attended the same lectures, thus, repeaters did not receive feedback or teacher's attention targeting their weaknesses, and were taught in the same manner as before
	(TL2) Lack of study material	Tutoring problems and exercise problem sets did not change from one Time Block (TB) to the next, and the summative exams from the previous TBs were the only new study material repeaters had as preparation material
	(TL3) Ineffective tutoring sessions	Students from different modules attended the same tutoring sessions, thus the Tas had to constantly switch between problem sets when guiding the students, providing clarification and feedback
Assessment	(AS1) Problem statements too verbose	Problem statements in both lab assessments and exams were too verbose. Furthermore, statements lacked a consistent structure, and sometimes explanations appeared vague or confusing. For instance, the first M1 exam (see Fig. 6 in Appendix C), consisting of two problems, was 1266 words long. With two hours for the exam, students required an excessive amount of time to read the problem statements and interpret them correctly
	(AS2) Unfair lab grading	Lab assignments based on automated grading often over-penalized students (i.e., awarded zero points) when the output of a student's submission did not exactly match the expected output, even though the program may have been algorithmically correct. This affected students' motivation and often required assignments to be manually regraded Additionally, students tended to solve all parts of the lab problem before submitting their solution to the grading system, commonly at the last minutes of lab session. Minor mistakes in their solutions could lead them to obtain zero points in these situations
	(AS3) Inadequate homework incentives	Incentives to complete homework were inadequate, given that homework average was offered as a bonus to improve the final course grade only after the student had completed all four modules. Given that homework assignments were long and difficult, a considerable number of students did not submit their homework. The percentage of students not submitting homework was 20.4% in M2, 35.1% in M3, and 11.9% in M4
Instructional design	(ID1) Inappropriate timing of activities	Timing of module activities was inconvenient, as assessments were conducted the day after tutoring exercises. Students did not have enough time to practice and prepare for assessments properly

Table 5 (continued)

Component	Issue found	Description
Content and skills	(CSK1) Inappropriate tools	Students reported in interviews that hand drawn flowcharts were difficult to work with, as they had to frequently modify and iterate their solutions. They often realized they had to make major corrections to the chosen approach of solving the problem and start over with a blank piece of paper, which they felt was a waste of time. On the other hand, Tas noted that students had trouble identifying relevant variables, setting their initial values, updating their state, and working with complex flow control constructs, such as loops
	(CSK2) Content difficulty	As for the most difficult content, many of the interviewed students declared that M1 had been the most difficult, as the course content and skills were completely new to them and at the same time, difficult to learn. Regarding programming skills, functions (M2) and use of nested lists (M3) were reported by the students as the most challenging
Course management	(CM1) Lack of experience in assessment construction	The traditional programming course had only one summative examination at the end, whereas the modular course required multiple summative examinations focusing on the specifics of each module. The teaching staff did not have the experience creating summative assessments that could be highly readable so as to reduce students' extraneous load, while allowing them to demonstrate mastery of a specific (reduced) set of programming skills, especially in the case of the earlier course modules
	(CM2) Inappropriate scheduling of assessment development	Assessment development activities were not scheduled in ways that allowed sufficient time for revision and improvement of problem statements, before they were administered

Table 6 Actions performed in Iteration 2 to address issues found in Iteration 1

Component	Issue found	Actions performed
Teaching and learning	(TL1) Same lectures for both repeaters and newcomers	Repeaters and newcomers were separated into different groups both in lectures and in tutoring sessions. In addition, to encourage student participation in these activities, an attendance grade was added, with a weight of 7.5% on the grade of the module (see Table 2)
	(TL2) Lack of study material	Assessments from the previous year were delivered to students as study material, except in M1 where methodological and technological changes were enacted. In M1 the homework assignment was carefully aligned with the exam, to give students targeted preparation practice
	(TL3) Ineffective tutoring sessions	Students were assigned to tutorial sessions specific to their module. A third tutorial session was added to each Time Block (TB) just before the exam, primarily focusing on revising previous exams
Assessment	(AS1) Problem statements too verbose	Design of assessments was simplified, regarding language and structure. Description of the initial problem context was reduced to a minimum. Exams were designed to comprise a single problem context with sub-problems and/or subgoals of increasing complexity (see the M1 exam from Iteration 2, TB 1 in Fig. 7, Appendix C). Furthermore, illustrations were added to problem statements as a means to explain expectations for the problem solution
	(AS2) Unfair lab grading	Problem statements for lab assessments were improved with explicit recommendations for when students should submit to the CMS system, such as after achieving a specific milestone
	(AS3) Inadequate homework incentives	Homework assignment became mandatory, with a grade weight of 7.5% in each module (see Table 2). In addition, homework was made to align better with exam problems to ease students' study and preparation
Instructional design	(ID1) Inappropriate timing of activities	Time between tutorial activities and lab assessments was increased from one to three days
Content and skills	(CSK1) Inappropriate tools	It was decided to adopt the teaching of visual (i.e., block-based) programming before text-based in M1. This also required updating class material, creating short instructional videos and tutorial exercises for the first two weeks of M1
	(CSK2) Content difficulty	

Table 6 (continued)

Component	Issue found	Actions performed
Course management	(CM1) Lack of experience in assessment construction	As the teaching staff remained unchanged in Iteration 1, experience and reflection gained in the first course iteration was beneficial for the process of assessment construction. Teachers became aware that careful attention had to be paid to the format, clarity and conciseness of assessments (this conception, however, was further improved in the next iteration)
	(CM2) Inappropriate scheduling of assessment development	The course coordinator planned assessment creation deadlines for the teaching staff allowing at least four working days in order to revise and rework assessments if necessary

Table 7 Proportion of students in each course state by the end of the first (TB 5) and second (TB 11) semesters in Iteration 2

	M1	M2	M3	M4	Passed	Dropped out
End of first semester (TB5)	.00	.03	.37	.27	.31 (↑.12)	.02 (↓.01)
End of second semester (TB11)	.00	.00	.00	.02	.89 (↑.14)	.09 (↓.11)

(↑) Increase in desired direction, (↓) Decrease in desired direction

Table 8 Module pass rates for students' first, second and third attempts in their first three instances in Iteration 2

Instance	M1	M2	M3	M4
First	.78 (↑.39) (242)	.56 (↑.22) (189)	.36 (↓.14) (105)	.89 (↓.11) (38)
Second	.87 (↑.31) (53)	.35 (↓.44) (130)	.41 (↑.04) (117)	.77 (↑.14) (52)
Third	.43 (↑.4) (7)	.77 (↑.27) (83)	.42 (↑.15) (135)	.61(↓.11) (*67)

(↑) Increase in desired direction, (↓) Decrease in undesired direction

*Corresponds to the inter-term period, wherein 67 students out of possible 68 opted in

code.org draws parallels among block-based and text-based programming; the latter in ECMAScript language, which differs from Python.

With Scratch, in less than three weeks' time, students could learn basic problem analysis and how to generate programs with variables, input and output, operators, logical and arithmetic expressions, pseudo-random number generation, conditional and iterative flow control, and use of lists. In addition, the teaching of Python in M1 was simplified, moving basic flow control (i.e., the selection statement) to M2 (see Appendix B).

Observation

In the second iteration, students' progress through the course was notably faster than in iteration one, with an average time to finish the course of 20.5 weeks. This is a reduction of 2.7 weeks (SD=6.58), i.e., almost a complete TB, compared to the first iteration (M=23.2, SD=7.38). Correspondingly, module repetition was reduced.

By TB5 there were no students in M1, and 3% of the cohort were in M2 (see Table 7). This contrasts with Iteration 1, as in TB5 7% of the cohort were in M1 and 13% in M2. By the end of the inter-term period (TB6), 47.4% of the cohort had passed the course in iteration two, compared to only 28.5% in the first iteration. In addition, 89% of the cohort completed the course by the end of the academic year (see Table 7), compared to only 75% in the previous iteration (see Table 3).

Improved results in Iteration 2 with regard to student progress, were the result of increased pass rates in M1 and M2 (see Table 8). On the other hand, no major changes in teaching and format of summative exams were introduced in M3 and M4,

thus, results in these modules did not improve noticeably compared to the first iteration. However, like in the first iteration, students who passed M3 performed well in M4.

Table 9 shows homework and examination grades considering the first instance of each module. Greater student dedication to homework assignments was observed in Iteration 2. The percentage of students handing in their homework across the different modules ranged between 84.0% in M3 and 93.7% in M2. However, homework grades were consistently higher than examination grades in M1-M3, and the difference was verified to be statistically significant in these cases, with large effect sizes. In addition, correlations among homework and examination grades were poor, ranging between 0.14 and 0.28. This may relate to different conditions under which students develop homework assignments compared to exams; in terms of time, and the possibility to seek help from others. Notably, in M4 students did better in the exam than in the homework assignment, while the mean of homework was the lowest of all modules.

Reflection

After reflections involving the teaching team, teaching assistants and data analysis of students results in each module, we were able to see that the changes to the course in Iteration 2 succeeded at improving module pass rates at the beginning of the course, which accelerated students' overall progress in the course. The course was improved by taking into consideration the many issues found in the past iteration. However, pedagogical and course management aspects could be improved further. Table 10 shows salient issues found in iteration two, which related to teaching and learning and course management.

Iteration 3

Planning

Planning of the course remained similar with regards to TBs in each semester. Like in the previous iteration, only one member of the teaching staff having two years' experience in the modular course was replaced by a young lecturer with a few semesters experience teaching traditional CS1 courses. In order to cope with issue CM3 raised in Iteration 2 (see Table 10), planning of assessment construction had to be carefully negotiated by the course coordinator with each member of the teaching staff.

Action

Table 11 summarizes the actions performed in Iteration 2 to overcome issues TL4, TL5 and CM3. In addition, the Python language was upgraded to version 3.6, as version 2.7 was scheduled to sunset in January 2020. Syntactic and semantic changes among language versions are minor, especially as the present programming course

Table 9 Homework (HW) grades vs exam grades in the first instance of M1-M4 in iteration 2

TB & Module	N*	% of HW turned in	HW Grades		Exam Grades		Cohen's <i>d</i> HW vs. Exam Grades	<i>t</i> test M_{HW} vs. M_{Exam}	df	<i>p</i> value	Pearson's Correlation HW vs. Exam Grades	<i>p</i> value
			<i>M</i>	<i>SD</i>	<i>M</i>	<i>SD</i>						
1	207	87.7	5.85	1.20	4.91	1.38	-.73	8.01	206	<0.001	.14	<0.05
2	159	93.7	6.30	.81	4.63	1.77	-1.22	11.5	158	<0.0001	.16	<0.05
3	77	84.0	5.97	1.40	3.10	1.33	-2.12	15.4	76	<0.0001	.28	<0.05
4	33	89.7	5.35	1.54	6.39	.96	.83	-3.11	32	<0.001	-.15	<i>n.s</i>

* Only considers students who turned in homework and attended the first exam in the TB (i.e., not a recovery exam)

Table 10 Issues found in iteration 2

Component	Issue found	Description
Teaching and learning	(TL4) Spiral teaching in M1 and M2	Among faculty it became apparent that the progression of skills comprising M1 and M2 could be improved by having students learn block-based programming and later text-based programming through encountering similar examples and problems, and a similar ordering of skills. That is, opt for teaching in a spiral fashion in the first half of the course. This was expected to stimulate students' development of problem-solving schemata and metacognitive-strategies
	(TL5) Foster schemata building and meta-cognitive strategies in M3	The number of skills required of students to solve problems increases considerably in M3, while students encounter problems in assessments that have clear structural and/or skill commonality. Thus, it was considered that class material should be enriched with detailed explanations on how to deal with such problems
Course management	(CM3) III assignment of assessment authorship responsibilities to the teaching staff	Authorship of homework and examinations was commonly done by different members of the teaching staff, who were often unable to coordinate their creative process. Having a designated member of the teaching staff create both kinds of assessments for each module and TB, and striving to align them in terms of problem structure and required skills could improve constructive alignment in the course

Table 11 Actions performed in iteration 3 to address issues found in iteration 2

Component	Issue found	Actions performed
Teaching and learning	(TL4) Spiral teaching in M1 and M2 (TL5) Foster schemata building and meta-cognitive strategies in M3	<p>Contents in M1 and M2 were reordered to allow a spiral teaching strategy in the first half of the course. Class material and code examples were updated as required</p> <p>The third lecture in M3 was aimed at teaching students common programming patterns to solve recurring classes of problems found in assessments. The class material and code examples were updated accordingly</p>
Course management	(CM3) III assignment of assessment authorship responsibilities to the teaching staff	<p>Assessment authorship duties were distributed to the teaching staff so that a member had to create both the homework and the exam in each module/TB in liaison with the course coordinator</p>

only focuses on the imperative procedural paradigm. However, this required a comprehensive update to the courses' textbook, class material and code examples.

Observation

Students' progress was further improved in the third iteration, with a mean time to complete the course of 17.7 weeks ($SD=5.60$). This is an improvement of an additional 2.8 weeks over Iteration 2, due mainly to an increase in module pass rates. As shown in Table 12, by the end of TB5, 54% of the cohort had passed the course. After the inter-term period, this figure increased to 67%. By the end of the year, 94% of the cohort passed the course, and only 6% dropped out.

Quicker student progress in the course was due to pass rates in M1 continuing to be above 0.7 in the first two instances the module was offered, and later improvements to pass rates in M2 and M3. Lastly, M4 maintained relatively high pass rates comparable to the second iteration (see Tables 12 and 13).

Reflection

Iteration 3 showed that positive effects of changes made in Iteration 2 could be replicated, and that further improvement was achieved in the light of greater pass rates observed in M2 and M3. As pass rates in M2 and M3 were consistently improved in several of their instances, it can be affirmed that changes to pedagogy linked to addressing TL4 and TL5 (see Table 11) had a positive influence on students' learning and performance. However, no evidence could be elicited regarding achievement of a better alignment among laboratory assignments, homework and examinations, as a result of addressing issue CM3. A path analysis procedure was conducted as a

Table 12 Proportion of students in each course state by the end of the first (TB 5) and second (TB 11) semesters in iteration 3

	M1	M2	M3	M4	Passed	Dropped out
First semester	.00	.01	.24	.16	.54 (↑.23)	.05 (↑.03)
Second semester	.00	.00	.00	.00	.94 (↑.05)	.06 (↓.03)

(↑) Increase in desired direction, (↓) Increase in undesired direction, (↓) Decrease in desired direction

Table 13 Module pass rates for students' first, second and third attempts in their first three instances in iteration 3

Instance/module	M1	M2	M3	M4
First	.71 (↓.07) (227)	.51 (↓.05) (162)	.73 (↑.37) (78)	.75 (↓.14) (56)
Second	.83 (-) (65)	.60 (↑.25) (138)	.65 (↑.24) (107)	.95 (↑.18) (84)
Third	–	.80 (↑.03) (55)	.54 (↑.12) (81)	.78 (↑.17) (37*)

(↑) Increase in desired direction, (↓) Decrease in undesired direction

*Corresponds to the inter-term period, wherein 37 students out of possible 48 opted in

means for establishing whether results in an assessment could predict results in later assessments. This was based on the notion that an assessment could predict performance in later assessments if students manage to build and apply schemata and transfer knowledge related to the intended learning outcomes throughout successive, aligned learning and assessment events (Robins et al., 2019). The procedure was performed with R v4, considering assessments in the first instances of modules M1 to M3. In our case, with all modules it was found that only the first lab assignment could predict performance in the second lab assignment and in the summative exam (i.e., path coefficients in the range 0.44 to 0.58, $p < 0.01$), although the latter to a lesser extent (i.e., path coefficients in the range 0.02 to 0.3, $p < 0.05$). The second lab assignment could not predict performance in homework nor in the summative exam. In turn, homework did not predict lab assignment nor examination performance (see Appendix C). This provides indication that further improvement with regard to constructive alignment can be pursued by increasing opportunities for knowledge transfer, schemata activation, and display of meta-cognitive strategies across homework, lab assessments and the summative exam.

Table 14 summarizes the issues found in Iteration 3, which relate to teaching and learning, assessment and course management. Overcoming these issues was considered essential in order to improve the course both from operational and academic standpoints.

Iteration 4

Planning

The fourth iteration of the course began in the midst of the worldwide COVID-19 crisis. In the first week of the course, the university complied with the regulations imposed by the local health authorities and the decision to teach online throughout the year was made official. The course had to be quickly adapted to this format.

The annual calendar of the course did not undergo changes compared to the previous year, keeping the number of weeks, TBs and the inter-term period unchanged. There was a 15% increase in freshmen enrollment compared to the previous year, so a sixth lecturer was added to the teaching staff. In the second semester the teaching staff was reduced to two lecturers, as in all course iterations.

Action

With the change to online education, synchronous activities such as tutorials and lectures were streamed on platforms such as YouTube and Twitch, or conducted by using videoconference systems, such as Google Meet. The lectures were reduced from 100 to 80 min, and their focus continued to be about demonstration of skills and step-by-step problem solving by the lecturers. In order to improve students' preparation for the lectures, the teaching team produced two to four short videos (i.e., three to ten minutes long) per week, 48 in total, to cover the fundamental contents covered in each lecture. In previous iterations of the course, there was a limited

Table 14 Issues found in iteration 3

Component	Issue found	Description
Teaching and learning	(TL6) Student procrastination in the second semester	Up to Iteration 3 it had been observed that regardless of the many changes to the course, students' progress stagnated in the second semester. This was mostly due to students' procrastination, as most who failed the course in the first semester needed to resume the course in M3 in the second semester, with five TBs ahead to complete it. This excess of TBs in relation to the number of modules pending passing did not compel students to complete the course timely, but rather incentivized them to dedicate more time to other activities
Assessment	(AS4) Improve constructive alignment in assessments	There is a need to improve the process of assessment construction, in ways that foster knowledge transfer from low stakes assessments, including lab assignments and homework, to the final summative examination
Course management	(CM4) Surge of regrading requests by TB end	In all course iterations so far, students were given the opportunity to ask for regrading of laboratory assessments, homework and the final examination by the end of each TB. While many regrading requests were well justified in relation to the assessment rubric and problem solutions published, poorly-justified requests also occurred. The large number of overall regrading requests in each TB (30 to 60% of the cohort submitted at least one regrading request) placed considerable burden on teaching assistants. In addition, a number of students complained in the end of semester survey every semester about having been awarded grades arbitrarily or unfairly

number of video capsules, used mainly in M1, but these were completely renovated. In a flipped classroom fashion, students were required to watch the video capsules before each class, and read sections of the textbook.

The course began with all six lecturers teaching M1, with groups of students assigned to each as had been done for the face-to-face format. Given that the online format allows assigning an unlimited number of students to each teacher, starting from TB2, the teaching staff that gave lectures was restricted to three or four lecturers per TB. Thus, about half of the teaching staff was dedicated to lecturing, and the other lecturers had greater dedication to developing assessments and short explanatory videos. Dedication to lecturing and content creation duties were planned by the course coordinator before the start of each TB.

With regard to assessment, changes were made in Iteration 4 (see Table 15), with the aim to reduce the number of students' requests for regrading at the end of each module, and thus lessen the burden this placed on teaching assistants at the end of every TB (i.e., issue CM4). In M1 and M2 grades were based only on class attendance and the summative exam. Homework assignments in these two modules were eliminated. Lab assessments were administered in M1 to M3, but were not compulsory. Rather, students could score bonus points to boost their final grade in the module (but were not allowed to request regrading). In M3, homework was kept in a similar format to the previous iteration and worth 12.5% of the final grade, and regrading was allowed. In M4, an integrative homework assignment, prompting students to comprehensively apply course knowledge and skills, was introduced, accounting for 22.5% weight in the final module grade. Given that this homework assignment was expected to demand a greater effort, lab assignments were omitted in M4 altogether.

The modules were taught with the same knowledge and skills ordering as in the last iteration. This allowed all the assessment material from the previous year to be completely reused in the form of worked examples so that the students could improve their preparation for the assessments in tutorials and in their personal study.

Changes in Iteration 4 were mostly focused on implementing the course in online format, however, it was feasible to address the issues that emerged in the past iteration to some extent (see Table 16). In the second semester, an attempt was made to improve student engagement with the support of the academic counseling model traditionally implemented in the institution. This consists of each first year student

Table 15 Course assessments and weights in Iteration 4

	Module 1	Module 2	Module 3	Module 4
Assessment per module	Weight			
Low stakes lab assessment	Two lab assessments which award bonus points (up to 0.6 points on the module grade, considering 1–7 grade scale, with 4 as the cutoff grade)			–
Final exam	92.5%	92.5%	80%	70%
Homework	–	–	12.5%	22.5%
Attendance	7.5%	7.5%	7.5%	7.5%

Table 16 Actions performed in Iteration 4 to address issues found in Iteration 3

Component	Issue found	Actions performed
Teaching and learning	(TL6) Student procrastination in the second semester	Support from the Faculty's secretary for student affairs was requested, in order to have students in the second semester being contacted by their appointed academic counselor to increase their course engagement
Assessment	(AS4) Improve constructive alignment in assessments	The assessment construction methodology was kept similar to Iteration 3. However, the amount of study material available for students in Iteration 4 was increased substantially, as most of the assessments conducted in the previous iteration were delivered to the students as worked examples, and used as class material for programming demonstrations
Course Management	(CM4) Surge of regrading requests by TB end	Lab assessments were only administered in M1 to M3, but these were not compulsory. Rather, students could score bonus points to boost their final grade in the module, and were not allowed to request regrading

Table 17 Proportion of students in each course state by the end of TB 5 in Iteration 4

	M1	M2	M3	M4	Passed	Dropped out
First semester	.00	.01	.05	.16	.77 (.23↑)	.00 (↓)
Second semester	.00	.01	.01	.02	.96 (.02↑)	.04 (↓)

(↑) Increase in desired direction, (↓) Decrease in desired direction

Table 18 Module pass rates for students’ first, second and third attempts in their first three instances in iteration 4

Instance	M1	M2	M3	M4
First	.96 (↑.25) (262)	.67 (↑.16) (251)	.78 (↑.05) (169)	.73 (↓.02) (131)
Second	.27 (↓.56) (11)	.88 (↑.28) (90)	.26 (↓.39) (117)	.92 (↓.03) (118)
Third	(-)	.73 (↓.07) (11)	.70 (↑.13) (47)	.84 (↑.0.06) (43)

(↑) Increase in desired direction, (↓) Decrease in undesired direction

having an assigned academic counselor, generally a full-time academic. The secretary for student affairs of the Faculty was asked to coordinate the appointment of video calls so that the students could meet their counselors, and be encouraged by them to finish the course as soon as possible. Regarding the aligned development of the assessments to facilitate knowledge and skill transfer, the criterion of the previous iteration was maintained, consisting of the same teacher having to develop homework and exams in M3 and M4 (Table 17).

Observation

In Iteration 4 the improvement trend with regard to students’ mean time to finish the course was maintained. This was figure was reduced to 14.9 weeks (SD=3.64), that is, an improvement of 2.84 weeks (almost a complete TB) compared to Iteration 3 (M=17.7, SD=5.6). In addition, 96.2% of the cohort completed the course by the end of the academic year, compared to 94.3% of students achieving this result in the previous iteration (see Tables 12 and 17).

Pass rates on M1 and M2 continued to improve (see Table 18). In Iteration 3 the same pass rate was found for M3 as the previous year in the first instance, but it worsened considerably in the second instance. In M4 there was also a worsening in the pass rate in the first instance. Nonetheless, at the end of TB5, 77% of the cohort passed the course, i.e., an increase of 23% compared to Iteration 3, due to a greater proportion of students in M4 in TBs 4 and 5, explained by better pass rates in M1 and M2.

Reflection

The best academic results in Iteration 4 were obtained in a teaching context different from that of the previous modules. Therefore, the improvement in academic results

can be attributed to both the deliberate improvements pursued, as well as other factors beyond control of the teaching staff and research agenda.

Undeniably, students in the online mode had advantages over those who previously took the classroom-based course at the time of taking assessments. During the assessments, they could access study material, assessments from previous semesters, and other resources. On the other hand, students in the face-to-face format were only allowed to consult the course's textbook during the assessments. In spite of this, it could not be established that there was systematic cheating and plagiarism on the part of the students when taking the assessments in the online course. At the beginning of the course, it was announced to the students that all assessments would be reviewed with Measure of Software Similarity (Aiken, 2020). In spite of this, in TB2, 23 students were found (8.8% of the cohort) to be suspected of cheating on their assessments due to code similarity. After investigation, it was determined that 11 students cheated in lab assignments or examinations. After these students were informed of the sanctions, which consisted of failing the course in most cases, plagiarism in the cohort appeared to decrease substantially (no new suspicions of plagiarism were detected in the following TBs).

The lifestyle of the students was also considerably affected by the pandemic and this could have positively influenced their performance in the Programming course. Under normal conditions, most students spend considerable time commuting to campus from their homes. On the other hand, the University does not have laboratories open to students permanently to facilitate their study of programming, therefore, students must have their own laptop to study on campus. During the pandemic, students spent time at home in front of their computers for most of the day, so this could have facilitated their dedication to the study of programming.

Students' module progressions and performance

A longitudinal analysis of students' performance and module progressions in the course was performed by consolidating academic results in a relational database, and constructing reports through a development environment based on the R programming language. A student's module progression is defined as the sequence of modules followed by the student throughout the course's TBs, from start to end, and is specified by the corresponding sequence of module numbers, from left to right. For example, module progression '112345' indicates that the student took M1 twice (i.e., failed M1 in the first TB), and continued studying and passing the following modules until passing M4 and thus achieving course completion by the end of the fifth TB. The number 5 in TB six of this progression indicates that the student had passed the course already.

Figure 3 shows the relative frequency of students' module progressions in each iteration of the course, only considering the first five TBs (i.e., the first semester). A total of 31 different module progressions were found across the four years. For the sake of clarity in Fig. 3, module progression labels are only provided for progressions including at least 3% of the students in the yearly cohort. A decreasing trend can be observed through the years in module progressions ending in 1, 2, and 3 (i.e.

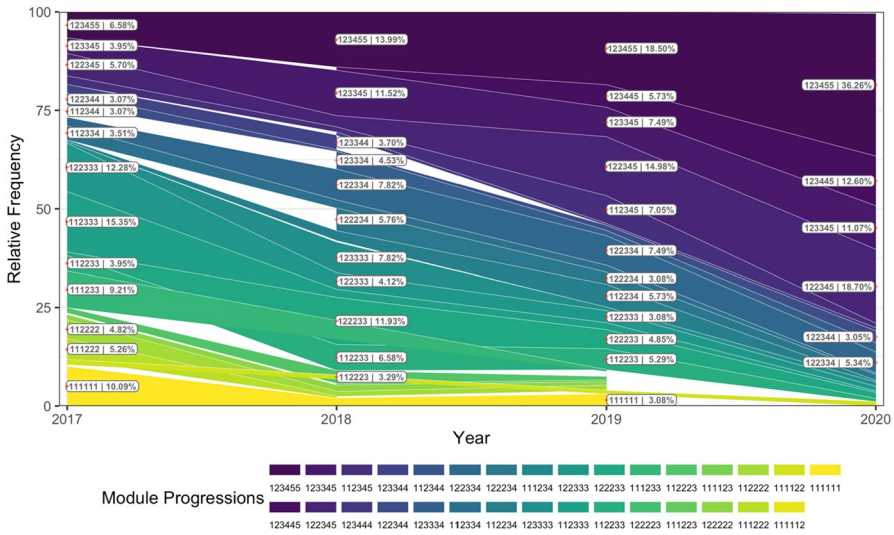


Fig. 3 Students' progressions in the modular course

students who did not pass the course in the first semester). Conversely, an increase is observed in progressions ending in 5 (students who did pass the course).

Table 19 shows trends in the different progressions (module sequences) at the end of the first semester (TB5) in each year (iteration). Trends were tested for statistical significance through a chi-squared test of proportions. Looking at progressions ending in M1 (i.e. 111111), in the first year, 10% of the cohort were still in M1 by the end of TB5. In the following two years, the proportion of students in this state was reduced to a minimal fraction of the cohorts (2 to 3%), and in the last year, there were no longer students who did not pass M1 by TB5. A similar trend is observed for progressions ending in M2 (e.g. 111222, 112222), as 13% of the students in the first year were still in M2 by the end of TB5. In contrast, in the last year only 1% of the cohort had not passed M2 by the end of TB5.

A considerable portion of the cohort in the first two years followed progressions that were still in M3 at the end of TB5 (e.g. 112333, 122333). These results were improved in the third year, as 70% of the cohort managed to pass M3 by the end of TB5. In the last year, only 5% of the cohort had failed to pass M3 by the end of TB5. Lastly, for progressions ending in M4 (e.g. 122344, 123344), the same proportion of students in the first year of the intervention as in the last. However, the proportion of students who managed to pass the full course at the end of TB5 increased steadily from year to year, reaching 80% in 2020.

Figure 4 presents the proportion of students who passed the course every year, starting at the conclusion of TB4. While this proportion increased in each iteration, a consistent trend of slow progress occurred in every second semester. Students' progress stagnates in TBs 7–9 in the second semester, and improves in the last two TBs. Consistently with other mastery learning implementations (Fox, 2004; Garner et al., 2019), when the students are given too many opportunities to fail and retake

Table 19 Trends in module progressions organized by module at the end of TB5 by year

Module	Year	Number of progressions ending with module	Students with progression ending with module	Cohort size	Proportion	$\chi^2(1)$	p
1	2017	1	23	228	0.10	28.74	<0.001
	2018	1	5	243	0.02		
	2019	1	7	227	0.03		
	2020	0	0	262	0.00		
2	2017	5	30	228	0.13	32.18	<0.001
	2018	3	8	243	0.03		
	2019	3	7	227	0.03		
	2020	1	3	262	0.01		
3	2017	8	100	228	0.44	116.38	<0.001
	2018	7	88	243	0.36		
	2019	9	43	227	0.19		
	2020	5	13	262	0.05		
4	2017	6	33	228	0.14	1.00	<i>n.s</i>
	2018	9	68	243	0.28		
	2019	7	48	227	0.21		
	2020	7	36	262	0.14		
5 (Passed)	2017	4	42	228	0.19	216.06	<0.001
	2018	5	74	243	0.31		
	2019	5	122	227	0.54		
	2020	5	210	262	0.77		

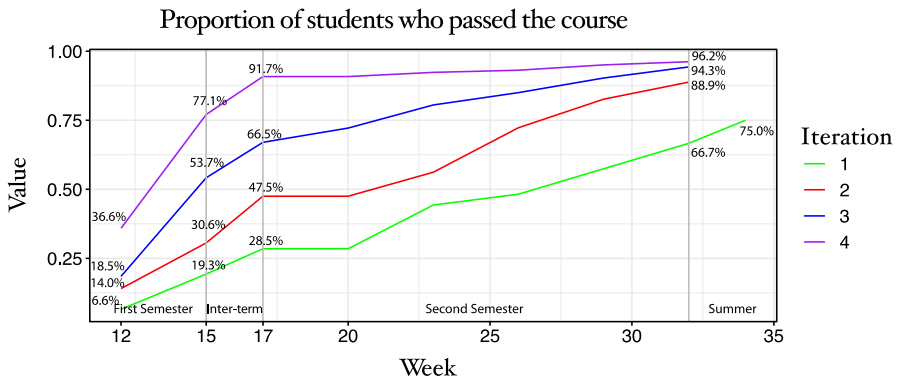


Fig. 4 Proportion of students who passed the course starting at the end of week 12 (TB4) in every course iteration

assessments, or are not faced with deadlines to be promoted to more advanced studies, they tend to procrastinate. Procrastination in the modular course became apparent in the second semester, as most students begin thereby in M3 or M4, and have a

total of five to six TBs available in the semester to pass one or two modules. Late in the second semester, procrastinating students encounter pressure to finish the course in the last few TBs ahead. Despite this, the proportion of students ending the following academic year without completing the course steadily dropped through course yearly iterations, from 25% of the cohort in 2017 to 3.8% in 2020.

Discussion

Challenges and difficulties

Introducing fundamental knowledge and skills

The fast-paced planning of the modular course made the first module intense, as students are expected to develop basic problem analysis, abstraction and algorithm design skills in no more than three weeks, while taking other parallel courses in the engineering curriculum. How skills are taught and assessed early in the course is of the utmost importance for students' academic success, as early course results impact their motivation and ability to make sustained progress in later modules. Finding ways to accomplish fruitful learning and successful academic results in M1 proved to be a challenge at the outset of the project. Clearly, the use of a visual block-based programming language in Iteration 2 onwards was found to be a more effective alternative than traditional hand drawn flowcharts, to support students' own thinking and solution modeling. This, however, was not obvious at the outset of the project and had to be realized by the teaching team after the results of the first iteration.

Managing increasing knowledge complexity and assessment difficulty

The epistemology of programming results in discernable pedagogical challenges in the modular course. As the course progresses, knowledge and skills that students are expected to master become more complex, while problem solving activity requires that students integrate and apply a growing number of skills. As a greater number of programming concepts and skills become available for formulating more sophisticated problems, creative possibilities multiply. Yet for the teaching staff, experience with the modular course design led to the realization that good assessment construction is not about presenting students with entirely new and unexpected problems in assessments (i.e., '*gotcha* problems'). Instead, assessments need to consist of problems in which students can identify and transfer schemata they have had a chance to build over their hours of prior study and practice, including with worked examples. Neglecting this results in undermining constructive alignment possibilities between teaching and learning activities, and assessment.

As suggested by Robins et al., (2019), a programming pedagogy that better adjusts to students' cognition should consider worked examples, and problems establishing subgoals. In addition, for students to develop knowledge transfer abilities, they must be presented with examples of how a problem-solving strategy can

be transferred from one problem to another. These are pedagogical challenges especially for novice teachers of programming, who as in other disciplines tend to replicate the ways in which they were taught and assessed in their own educational experience.

Student procrastination

Consistent with the literature on mastery learning and reported experiences in mastery learning (Fox, 2004), student procrastination is an issue that occurred in the present modular course. This behavior is observed in Fig. 4, where a plateau in the progress of the students remains evident in each second semester. A possible workaround to the procrastination issue in the second semester is to introduce the students incentives upfront at the start of TB7, so that they commit dedication and effort to pass the course as early as possible. A maximum number of reattempts to pass each pending module could be defined. For instance, three reattempts could be allowed for M3 and M4 in the second semester. That is, if the student fails a module for the third time, fails the course. Rules such as this will be tested in future course iterations.

Team coordination and teaching freedom

Unlike traditional courses, in which the course can be taught by a single teacher, under the modular approach, several teachers need to collaborate. As the semester progresses, modules need to be taught in parallel, and in every time block each module requires different learning activities, content and assessments. Therefore, transition from the traditional course format to the modular requires a culture wherein the teaching staff is willing to relinquish some of their individual teaching freedoms for the sake of the collective effort that underpins the modular course. In this regard, the role of the coordinator proved essential in the modular course, not only for planning and assigning teaching staff's duties and supervising course activities in a daily basis, but also for collaborating with the teaching staff in creating course content, and assessments well-aligned with prior teaching and learning activities for each module.

Practical implications and recommendations

Planning of learning activities

The modular system presented here is based on the division of the academic term into time blocks of fixed duration. This facilitates that student finish a given module and move on to the next without waiting. However, the division of the course into four three-week modules makes the rate with which students are exposed to the content faster than that of a traditional course. In the modular course, a student who passes all modules without failing any of them can finish the course in 12 weeks, versus 15 weeks in a traditional semester course. With this accelerated pace of

learning, planning of learning activities must be organized carefully so that sufficient time passes after the student is presented with basic skills in order for them to exercise the skills properly and solve enough practice problems before they are assessed. On the other hand, the modular course must offer differentiated learning activities for students repeating a module from those taking a module for the first time. Repeating students can spend less time in lecture and focus most of their time exercising on problems from previous assessments, problems they were unable to solve before, and have further opportunities to seek for help and guidance from more capable peers and the teaching staff.

Block programming before text-based programming

The process of learning computational thinking and programming demands the concurrent display of a complex mix of skills, including reading comprehension, problem analysis, abstraction, and elaborating the solution representation. According to learning edge momentum theory (Robins, 2010), mastering these basic skills is key to academic success in the course. Traditional ways in which computational thinking and algorithm design skills have been taught in CS1 courses have been based on the generation of pseudocode, flowcharts or the use of visual modeling tools. More recently, the use of block programming languages, such as MIT Scratch and code.org, has become common, especially in K12 education. Despite the fact that these languages are frequently used for recreational purposes (e.g., for creating video games and multimedia), experience in the modular course shows that they are an powerful means for introducing post-secondary students to algorithm design and programming. Through these languages, students do not have to type code, including all the syntactic subtleties involved, such as indenting code with tabulation, and correctly typing each line. Instead, students use programming blocks that are easily recognizable (i.e., visual, with definite shapes and colors), and syntactically helpful (pluggable). The fact that with block programming students do not have to type code from the very beginning in a blank file brings the benefit of offloading some of the cognitive load of the task, so that they can rather dedicate more cognition and effort to model the solution, test it, and try different ways to solve the problem. In addition, other desirable skills, such as program tracing and debugging can be developed by the students early in the course through the use of block-based programming. The introduction of block-based programming and text-based programming worked well in a spiral fashion, meaning that students can be introduced to text-based programming through the same examples they saw before with block-based programming. The intent of this is to facilitate schema building and knowledge transfer, and again, offloading some of the cognitive load, so that when they come to learn the text-based programming language they can focus on learning the syntax and the mechanics of typing code sentences, through examples whose algorithmic and computational underpinnings they are already familiar with.

Assessment construction

There has been debate about the convenience of incorporating cover stories or detailed contexts into programming problem statements (Morrison et al., 2015). The experience in the modular course studied here was that verbose problem statements in the tests and lab assignments created a greater need for complex reading comprehension, adding extraneous cognitive load and leading students to incorrectly interpret the requirements of the problem (Robins et al., 2019). In addition, reading unnecessarily lengthy problem statements can be detrimental to students' performance in time-constrained exams. Problem statements that minimize the initial context presentation and lead the student to the problem requirements directly, stated as sub-goals, are thus preferred. Each subgoal can add more context to the problem statement if required. Subgoals can progressively increase skill complexity, and can allow incremental progress in solving the given problem. However, it is recommended that there are sub-goals that do not depend on fully completing others, so that the student has a greater opportunity to demonstrate their mastery of independent skills. Finally, if it is desired that students face less structured problems, this can be done through homework assignments, for which they can have much more time, and they can also turn to the teaching staff and their peers to resolve doubts and exchange ideas.

Constructive alignment

In the modular course, it is essential that students know how they will be assessed and that they can prepare in advance to take a summative exam successfully, as summative exams largely determine their success or failure in the modular course. For this, it is essential that students can have study material with abundant worked examples, and that tutoring sessions are focused on supporting students to solve problems at the exam's required level of proficiency. Through these activities, students must be capable of building their computational thinking and problem solving schemata, and complementarily, assessments must be constructed in ways that foster students' activation of that same schemata, and promote transfer of their learned abilities to the problems that are presented.

Teaching team and culture

In a CS1 course, as in other contexts where there is teaching with technology, effective teaching depends on the technological pedagogical content knowledge (Koehler & Mishra, 2009) that the teachers possess. With regard to pedagogical knowledge, teachers must have the ability to teach using the programming language, performing live coding examples that students can follow and perform simultaneously with them. Teachers must also have the ability to review what students program, give them feedback, and help them overcome what is keeping them from moving forward. In addition, teachers must know and be aware of the nature of the knowledge and skills that are required to be taught in each module of the course, since this is essential if a constructive alignment strategy is to be developed. From an

organizational standpoint, teachers must be willing to work as a team and to perform under a coordinating role. Lastly, the course coordinator together with the teaching team must have the support of the Faculty leadership to have freedom to make decisions about how to innovate and continuously improve the course.

Conclusions

This paper presents the development of a modular CS1 course based on mastery learning, over a period of four years, with a total of 959 engineering freshmen in a Latin American university. The development of this intervention reduced yearly course attrition from 25 to 3.8% of the cohort, decreased the average time spent by students in the course from 23.2 ($SD=7.38$) weeks to 14.9 ($SD=3.64$), and improved the course pass rate in the first semester from 19.3 to 77.1%.

The modular course based on mastery learning fulfilled the goal of allowing students to move through different module progressions according to their own learning ability and effort. The course gave the students the possibility to retake modules in which they had greater learning difficulties, without failing the entire course, as had happened to 30 to 40% of the cohort in the previous traditional course. Despite procrastination observed in the second semester, in the final iteration of the intervention, over 95% of the students passed the complete course by the end of the year.

The implementation of a CS1 course in the format here presented requires a well-aligned teaching team coordinated around the project and its objectives. In addition, it demands teachers with a high level of pedagogical, technological and epistemological knowledge about the teaching of programming. In the present project, this knowledge had to be developed and systematized over the course of the four years. Some practices that emerged in the present research could help researchers and practitioners from avoid some of the traps the present authors fell into at the beginning of the project. First, the way in which the initial course content is introduced influences the motivation and academic performance of students as they move towards more complex contents and skills. This was found in the first two iterations of the course and is consistent with the theory of learning edge momentum (Robins, 2010). Second, the constructive alignment of learning goals, with teaching and learning activities and assessment must be achieved through the development of teaching materials, worked examples, and assessments in a coordinated manner by the teaching team. The teaching materials must cover all relevant problem solving strategies and schemata that students need to learn in each module. Third, assessments must focus on validating students' possession of intended problem solving schemata and actual programming skills, both which must be clearly identifiable and in explicit connection with students' past learning experiences. In addition, given that skills and knowledge grow progressively more complex in the course, planning of teaching and learning activities must allow sufficient time for students' deep learning and preparation. These recommendations are consistent with recent research at the intersection of cognitive science and computer science education (Robins et al., 2019). Lastly, the

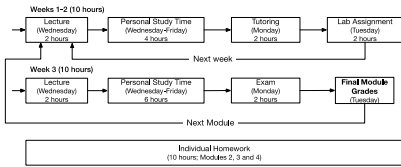
support and trust of the Faculty leadership, in terms of granting creative autonomy and decision-making to the teaching team for continuous improvement of the course is key to academic success.

In the future, the present authors aim to propose a formal process for the design of constructively aligned teaching, learning activities and assessments in the context of mastery learning-based CS1 courses. Also, the present authors are developing predictive models to detect students at greater risk of failing the course in the first semester, aiming to provide these students with greater support and personalization of their learning experiences. Lastly, a mobile intelligent programming tutor will be evaluated as a means to provide students with a complement to regular teaching and learning activities in the course, for constant programming practice, anywhere and anytime.

Appendix A

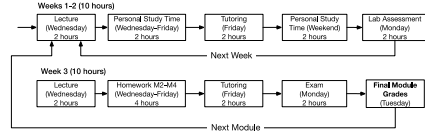
See Fig. 5.

Course Module Structure (Iteration One, 2017)



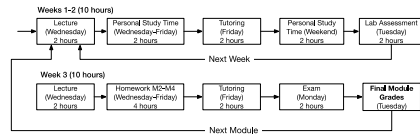
(a)

Course Module Structure (Iteration Two, 2018)



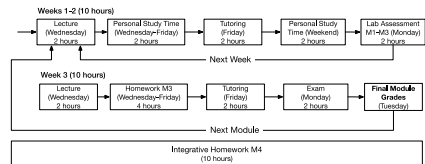
(b)

Course Module Structure (Iteration Three, 2019)



(c)

Course Module Structure (Iteration Four, 2020)



(d)

Fig. 5 Design of course modules in course iterations **a** One (2017), **b** Two (2018), **c** Three (2019), and **d** Four (2020)

Appendix B

See Table 20.

Table 20 Course contents over time in all iterations

Module	Course contents	Iteration 1	Iteration 2	Iteration 3/Iteration 4
1	Introduction to computational problems and algorithms Algorithm representation and modeling with flowcharts Python: Basic input–output, data types, variables, operators, expressions Python: Conditional flow control (if) Python: Pseudo-random number generation and applications	Introduction to computational problems and algorithms Algorithm representation and modeling with MIT Scratch Basic concepts: sprites, backdrop, scripts and blocks, input–output, operators, variables, pseudo-random numbers, and conditional and iterative flow control Scratch: Advanced flow control (nested looping), use of lists Python: Basic input–output, data types, variables, operators, expressions Python: Pseudo-random number generation and applications	Introduction to computational problems and algorithms Algorithm representation and modeling with MIT Scratch Basic concepts: sprites, backdrop, scripts and blocks, input–output, operators, variables, pseudo-random numbers, and conditional and iterative flow control Scratch: Advanced flow control (nested looping), use of lists Python: Basic input–output, data types, variables, operators, expressions Python: Pseudo-random number generation and applications	Introduction to computational problems and algorithms Algorithm representation and modeling with MIT Scratch Basic concepts: sprites, backdrop, scripts and blocks, input–output, operators, variables, pseudo-random numbers, and conditional and iterative flow control Scratch: Advanced flow control (nested looping), use of lists Python: Basic input–output, data types, variables, operators, expressions Python: Pseudo-random number generation and applications
2	Python: Iterative flow control (while) Python: Functions	Python: Conditional statement (if-elif-else) Python: Iterative flow control (while) Python: Functions	Python: Conditional statement (if-elif-else) Python: Iterative flow control (while) Python: Functions	Python: Conditional statement (if-elif-else) Python: Iterative flow control (while) Python: String manipulation Python: Introduction to Lists
3	Python: String manipulation Python: Lists, nested lists and slices Python: File access Python: Looping over collections (for)	Python: String manipulation Python: Lists, nested lists and slices Python: File access Python: Looping over collections (for)	Python: String manipulation Python: Lists, nested lists and slices Python: File access Python: Looping over collections (for)	Python: Nested lists and slices Python: Loops over collections (for) Python: Dictionaries Python: Functions
4	Python: Dictionaries Python: Numeric Python (NumPy) Python: charts (Matplotlib), Python: introduction to recursion	Python: Dictionaries Python: Numeric Python (NumPy) Python: charts (Matplotlib) Python: introduction to recursion	Python: Dictionaries Python: Numeric Python (NumPy) Python: charts (Matplotlib) Python: introduction to recursion	Python: File access Python: Numeric Python (NumPy) Python: charts (Matplotlib) Python: introduction to recursion

Appendix C

See Figs. 6, 7.

CSI - Exam - MIV1

Module 1 Exam
2017-TB1

Time: 2 hours

Problem 1

Introduction

Two elements are used to pay the ticket in our public transportation system: a metro transferable, and prepaid cards that store information and can only be written.

A passenger can transfer from one means of transport to another, so a trip on the first leg of a trip, the ticket corresponding to that leg is paid, and in case if there is a difference between the total paid for the trip and the cost of the Metro has a higher cost than the Bus, so a trip that consists only of Bus in which there is at least one Metro section.

When boarding the bus, the passenger presents their prepaid card and the charged or not, if there is enough balance and responds by turning on a red B to the failed or successful result, respectively.

The leg information is stored on the prepaid card as a log list, including the f

- **service**: A string identifying the name of the line or service (Example: direction: A string identifying the direction of travel (Example: 'City case of Metro, it is not possible to know the direction of travel beforea always 'Metro', and the passenger will always travel in the same direction
- **time**: An integer registering the time at which the prepaid card was too measured in minutes since 01-01-2007 at 00.00.00.
- **payment**: An integer telling the amount paid in the leg of the trip.

Between the beginning of the first and the beginning of the last leg of a trip, i pass, otherwise it is considered that a new trip has started and the user is charged take a bus route in the opposite direction within the same trip, in whi its corresponding charge.

Requirements

Draw a flow chart for the algorithm that a validator installed on a bus or me must determine how much to charge in a leg and turn on the red light in case due to insufficient balance. Assume that there are the following variables that

- Data about the current trip leg, which becomes available after setting ' list': **service, direction, time, payment**
- Data about the bus or Metro in which the validator is installed: **serVal diVal** with the direction of the current service, **ticketVal** with the installed.

Problem 1 continues in the next page...

CSI - Exam - MIV1

Problem 1 (continued)

- **currentTime** with the current time as an integer (counted in minutes metroTicket indicating the metro fare, busTicket indicating the bus busTicket < metroTicket < 2*busTicket.
- **balance** indicating funds available in the prepaid card.

At the end of execution, the following variables must be set:

- **resVal**: the value 'green' must be set on successful card validation, 'red'
- **paymentVal**: value that must be paid according to validation, or zero
- **balance**: remaining funds in the prepaid card (no charges are made t

Hint

- Given the way in which time is represented in variables **time** and **ct** counts, sum and add minutes to hours like with integer numbers.

Problem 2

Introduction

A clandestine bank has been operating in the country for years, which to authorizes processes customer transaction orders only through ordinary mail indicating the account numbers to / from where they are transfer money. TI of officials who process the envelopes manually. Over time, the capacity has b is implementing an optical character recognition (OCR) system to process o system identifies checking account numbers in the letters received, and send corresponding branch, which is deducted from each specific account number. So far the OCR system works perfectly and manages to identify all the 9- letters, without generating errors. However, most of these numbers represent t checking accounts, thus the improvement in productivity is not what is exp

Requirements

You are asked to write a program that can discard those numbers that are le a bank account number. In case of representing an account, the program is a branch and the responsible area manager, to whom this letter will be sent to l been discarded, the word 'Discarded' must be printed on the screen. The prog account number to be analyzed and then process it according to the following:

- No account begins with the digit '0', that is, the digit '0' never appears
- The first 3 digits of an account correspond to the branch, but there is a outside the country and has the first 4 digits dedicated to identifying th
- The branches have identifications within the ranges 101 to 221, 450 to both numbers inclusive).
- The next digits after the branch number identify the account within th
- An account can be in national or foreign currency. All foreign currency

Problem 2 continues in the next page...

CSI - Exam - MIV1

Problem 2 (continued)

- The identification of accounts in national currency has 4 digits, while th digits.
- For accounts in foreign currency, after the account identification the next The code for Chilean pesos is 99, but it is not part of the account numb
- The penultimate digit always represents the area within the branch that except for branch 3016 which only has area 1 and is not added to the ac
- The last digit is a Check Digit (CD). Note that it is only necessary to e not necessary to do any verification or validation of the account number

The output of the program must include a line indicating each of the followi

- Branch
- Account
- Currency
- Area
- CD

Examples of expected output for different inputs

Input	Branch	Account	Currency	Area	CD
055945297	Discarded	-	-	-	-
102345678	102	3456	99	7	3
301686537	3016	865	3	1	7
301670342	3016	7034	99	1	2
301570342	Discarded	-	-	-	-
43254983	Discarded	-	-	-	-
251268504	Discarded	-	-	-	-
221868543	221	868	5	4	3

Hints

- An integer can never have '0' as the first digit. If a string starting with '0' all leading zeros are simply ignored. Ex: '00456' becomes 456, that is, digits.
- Sketch a paper flow chart to structure your algorithm before program evaluation, it can help you sort through your ideas and make program

Fig. 6 Module 1 exam from Iteration 1 (2017), Time Block 1. Translated from original Spanish

CSI - Exam - MIV1

Module 1 Exam
2018-TB1

Instructions

This exam has four problems and you must solve only three. Problems 1 and 2 ar can freely choose a problem between 3 and 4. You will have 2 hours to solve the 1 Bonus points: If you choose to solve problem 4 and submit it complete and con with additional 0.5 points.

Problem 1

Develop a Python program (name it p1.py) that simulates the operation of a receive the current time and the alarm time separately. The alarm lasts for 90 activation time. The program should print True if the current time is within 90 seconds and False otherwise. Two executions of the program are illustrated below:

```

Enter the current time
Enter hour: 3
Enter minute: 2
Enter second: 1
Enter the alarm time
Enter hour: 3
Enter minute: 2
Enter second: 1
False

Enter the current time
Enter hour: 2
Enter minute: 3
Enter second: 4
Enter the alarm time
Enter hour: 2
Enter minute: 2
Enter second: 0
True
    
```

Tip: Your program can convert the hours entered to seconds, and perform compar entered by the user falls within the alarm activation interval.

Problem 2

Develop a Scratch program (name it p2.sb2) that simulates the operation of a c For this, the user enters the starting time, including hour, minute and second, se

Problem 2 continues in the next page...

CSI - Exam - MIV1

Problem 2 (continued)

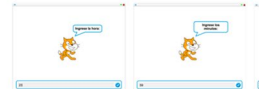


Figure 1: The start time is entered separately. In this example, 23:59:58

The cat displays a message with the time entered for one second, then advances t




Figure 2: Counting begins at 23:59:58. Images of the first three seconds

Tip: Remember to consider what happens when the seconds reach 60, when the hours reach 24.

Problem 3

Develop a Scratch program (name it p3.sb2) that implements a clock that rec city, and displays the time according to the chosen time zone.

First the start time is requested separately, as in the previous problem:




Figure 3: The start time is entered separately. In this example, 23:59

Then the city where the user is located is requested:

Problem 3 continues in the next page...

CSI - Exam - MIV1

Problem 3 (continued)




Figure 4: The city is entered.

The program must have two lists (you must create them yourself): one with citi respective time zones, the latter indicating the number of hours to be added to t

Cities	hours
1 Santiago	0
2 Lima	2
3 Buenos Aires	3
4 New York	4
5 Sydney	5 +11
length: 5	length: 5

Figure 5: List of cities and time zones.

Problem 4

Develop a Scratch program (name it p4.sb2) that simulates the operation of a cl must enter the starting time of the clock, with hour, minute and second separate




Figure 6: The start time is entered separately. In this example, 23:59

Internally it has a list with previously entered alarms:

Problem 4 continues in the next page...

Fig. 7 Module 1 exam excerpt from Iteration 2 (2018), Time Block 1. Translated from original Spanish

Appendix D

See Fig. 8.

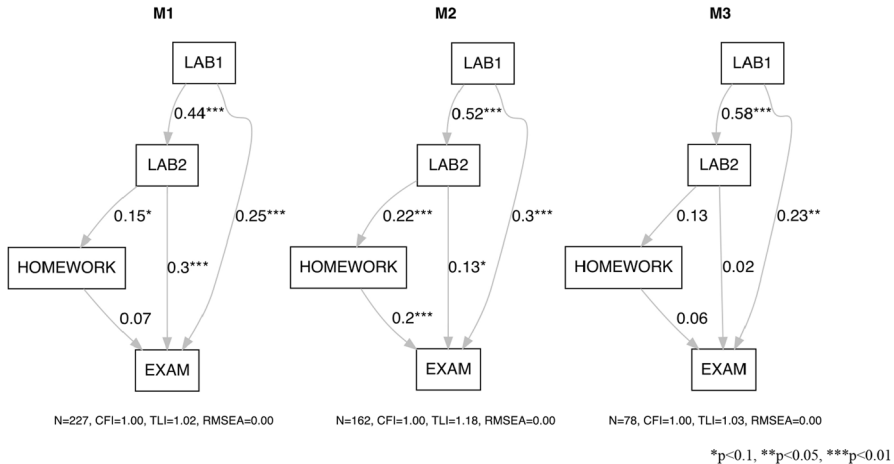


Fig. 8 Path analyses conducted with assessments in M1-M3, in the first instance of each, in course Iteration 3

Funding Funding was provided by ANID FONDECYT, (Grant Number: 11160211), Claudio Alvarez

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

- Ahadi, A., & Lister, R. (2013). Geek genes, prior knowledge, stumbling points and learning edge momentum. In *Proceedings of the ninth annual international ACM conference on International computing education research* (pp. 123–128). ACM.
- Ahadi, A. L. (2014). Falling behind early and staying behind when learning to program. In *PPIG*, 14.
- Aiken, A. (2020). *Measure of Software Similarity (MOSS)*. Retrieved 12 2020, from <https://theory.stanford.edu/~aiken/moss/>
- Armoni, M., & Gal-Ezer, J. (2014). High school computer science education paves the way for higher education: The Israeli case. *Computer Science Education*, 24(2–3), 101–122.
- Basnet, R. B., Payne, L. K., Doleck, T., Lemay, D. J., & Bazalais, P. (2018). Exploring bimodality in introductory computer science performance distributions. *Eurasia Journal of Mathematics, Science and Technology Education*, 14(10), 1591.
- Bayu Bati, T., Gelderblom, H., & van Biljon, J. (2014). A blended learning approach for teaching computer programming: Design for large classes in Sub-Saharan Africa. *Computer Science Education*, 24(1), 71–99.

- Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, 20(1), 45–73.
- Bennedsen, J., & Caspersen, M. E. (2007). Assessing process and product: A practical lab exam for an introductory programming course. *Innovation in Teaching and Learning in Information and Computer Sciences*, 6(4), 183–202.
- Bennedsen, J., & Caspersen, M. E. (2019). Failure rates in introductory programming: 12 years later. *ACM InRoads*, 10(2), 30–36.
- Biggs, J. B. (2011). *Teaching for quality learning at university: What the student does*. UK: McGraw-Hill Education.
- Bloom, B. S. (1968). Learning for mastery. *Evaluation comment*, 1(2).
- Bocconi, S., Chicciariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). *Developing computational thinking in compulsory education-implications for policy and practice*. Seville: Joint Research Centre.
- Carroll, J. B. (1963). A model of school learning. *Teachers College Record: The Voice of Scholarship in Education*, 64(8), 1–9. <https://doi.org/10.1177/016146816306400801>
- Code.org, CSTA & ECEP Alliance. (2021). In *2021 State of computer science education: Accelerating action through advocacy*. Retrieved 4 2022 from <https://advocacy.code.org/stateofcs>
- Cohen, L., Manion, L., & Morrison, K. (2013). *Research Methods in Education*. Routledge.
- CompTIA. (2021). *IT Industry Outlook 2022*. CompTIA Properties, LLC. Retrieved 5 2022 from <https://connect.comptia.org/content/research/it-industry-trends-analysis>
- Dochy, F. J. R. C. (1989). Modularisation and Student Learning in Modular Instruction in Relation with Prior Knowledge. ERIC.
- Dorn, B., & Tew, A. E. (2015). Empirical validation and application of the computing attitudes survey. *Computer Science Education*, 25(1), 1–36.
- Emery, A., Sanders, M., Anderman, L. H., & Yu, S. L. (2018). When mastery goals meet mastery learning: administrator, teacher, and student perceptions. *The Journal of Experimental Education*, 86(3), 419–441.
- Eurostat. (2022). ICT specialists in employment. Retrieved 5 2022 from https://ec.europa.eu/eurostat/statistics-explained/index.php?title=ICT_specialists_in_employment
- Eyre, H. L. (2007). Keller's personalized system of instruction: was it a fleeting fancy or is there a revival on the horizon? *The Behavior Analyst Today*, 8(3), 317.
- Fink, D. L. (2013). *Creating significant learning experiences: An integrated approach to designing college courses*. Hoboken: Wiley.
- Fox, E. J. (2004). The Personalized System of Instruction: A Flexible and Effective Approach to Mastery Learning. *Evidence-based educational methods*, 201–221.
- Garner, J., Denny, P., & Luxton-Reilly, A. (2019). Mastery Learning in Computer Science Education. *Proceedings of the Twenty-First Australasian Computing Education Conference* (pp. 37–46). ACM.
- Goldshmid, B., & Goldshmid, M. L. (1973). Modular instruction in higher education. *Higher Education*, 2, 15–32.
- Guzdial, M. (2019). Technical perspective: Is there a geek gene? *Communications of the ACM*, 63(1), 90–90.
- Jenkins, A., & Walker, L. (2014). *Developing Student Capability through Modular Courses*. Routledge.
- Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior*, 52, 200–210.
- Keller, F. S. (1968). Good-bye teacher. *Journal of Applied Behavior Analysis*, 1(1), 79.
- Koehler, M., & Mishra, P. (2009). What is technological pedagogical content knowledge (TPACK)? *Contemporary Issues in Technology and Teacher Education*, 9(1), 60–70.
- Koulouri, T., Stanislau, L., & Macredie, R. D. (2014). Teaching introductory programming: A quantitative evaluation of different approaches. *ACM Transactions on Computing Education (TOCE)*, 14(4), 1–28.
- Kurhila, J., & Vihavainen, A. (2015). A purposeful MOOC to alleviate insufficient cs education in finnish schools. *ACM Transactions on Computing Education*, 15(2), 1–18.
- Liao, S. N., Valstar, S., Thai, K., Alvarado, C., Zingaro, D., Griswold, W. G., & Porter, L. (2019). Behaviors of higher and lower performing students in CS1. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, 196–202. <https://doi.org/10.1145/3304221.3319740>
- Luxton-Reilly, A., Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., ... & Szabo, C. (2018). Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)* (pp. 55–106).
- Maggiolo, S., & Mascellani, G. (2012). Introducing CMS: A contest management System. *Olympiads in Informatics*, 6.

- McCartney, R. B. (2017). olk pedagogy and the geek gene: geekiness quotient. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, (pp. 405–410).
- Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, 23(3), 239–264.
- Morrison, B., Margulieux, L. E., & Guzdial, M. (2015). Subgoals, context, and worked examples in learning computing problem solving. *Conference on International Computing Education Research* (pp. 21–30). ACM.
- OECD. (2015). *Chile, in education at a glance 2015*. OECD Publishing.
- OGrady, M. J. (2012). Practical problem-based learning in computing education. *ACM Transactions on Computing Education (TOCE)*, 12(3), 1–16.
- Ott, C., Robins, A., & Shepard, K. (2016). Translating principles of effective feedback for students into the CS1 context. *ACM Transactions on Computing Education (TOCE)*, 16(1), 1–27.
- Patitsas, E., Berlin, J., Craig, M., & Easterbrook, S. (2019). Evidence that computer science grades are not bimodal. *Communications of the ACM*, 63(1), 91–98.
- Petersen, A., Craig, M., Campbell, J., & Taffioviich, A. (2016). Revisiting why students drop CS1. In *ACM International Conference Proceeding Series* (pp. 71–80).
- Robins, A. (2010). Learning edge momentum: A new account of outcomes in CS1. *Computer Science Education*, 1, 37–71.
- Robins, A. V., Margulieux, L., & Morrison, B. B. (2019). Cognitive sciences for computing education. *The Cambridge handbook of computing education research* (pp. 231–275). Cambridge University Press.
- Stein, L. (1999). Challenging the computational metaphor: Implications for how we think. *Cybernetics and Systems*, 30(6), 473–507.
- van Eijil, P. (1986). Modular programming of curricula. *Higher Education*, 15(5), 449–457.
- van Meel, R. (1993). *Modularization and Flexibilization*. ERIC.
- Vegas, E., Hansen, M., & Fowler, B. (2021). *Building Skills for Life: How to expand and improve computer science education around the world*. The Brookings Institution, Center for Universal Education Center for Universal Education.
- Watson, C., & Li, F. W. (2014). Failure Rates in Introductory Programming Revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 39–44). ACM.
- Webb, M., Davis, N., Bell, T., Katz, Y., Reynolds, N., Chambers, D. P., & Sysło, M. M. (2017). Computer science in K-12 school curricula of the 21st century: Why, what and when? *Education and Information Technologies*, 22(2), 445–468.
- Weurlander, M., & Soderberg, M. (2012). Exploring formative assessment as a tool for learning: Students' experiences of different methods of formative assessment. *Assessment & Evaluation in Higher Education*, 37(6), 747–760.
- Zuber-Skerritt, O. (2003). *New directions in action research*. Routledge.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Claudio Alvarez is an Associate Professor of Computer Science at Facultad de Ingeniería y Ciencias Aplicadas, Universidad de los Andes, Chile. He was awarded his Ph.D. in Engineering Sciences at Pontificia Universidad Católica de Chile, Chile.

Maira Marques Samary is an Assistant Professor of the Practice at the Department of Computer Science at Boston College, MA, USA. She was awarded her Ph.D. in Computer Science at Universidad de Chile, Chile.

Alyssa Friend Wise is Professor of Educational Technology & Learning Sciences in the Steinhardt School of Culture, Education, and Human Development, and Director of the Learning Analytics Research Network (LEARN), New York University, NY, USA. She holds a Ph.D. in the Learning Sciences from Indiana University, USA.

Authors and Affiliations

Claudio Alvarez^{1,2}  · Maira Marques Samary³  · Alyssa Friend Wise⁴ 

Maira Marques Samary
marquemo@bc.edu

Alyssa Friend Wise
alyssa.wise@nyu.edu

- ¹ Facultad de Ingeniería y Ciencias Aplicadas, Universidad de los Andes, Santiago, Chile
- ² Centro de Investigación en Educación y Aprendizaje, Universidad de los Andes, Santiago, Chile
- ³ Computer Science Department, Boston College, Boston, MA, USA
- ⁴ Steinhardt School of Culture, Education, and Human Development, New York University, New York, NY, USA