



Teaching highly mixed-ability CS1 classes: A proposed approach

Abdallah Mohamed¹ 

Received: 13 January 2021 / Accepted: 11 April 2021 / Published online: 2 July 2021

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021

Abstract

With the increased reliance on technology, computer programming has emerged as an essential skill that is interesting to many audiences beyond merely computer scientists. As a result, many students from various disciplines take first-year computer science courses. This led to classrooms with a lot of diversity in student motivation, backgrounds, learning needs, and educational levels. Teaching the same material to such a diverse group is challenging. The aim of this paper is two-fold. Firstly, we present a flipped-based approach that benefits from the mixed-ability nature of first-year programming courses rather than considering it as a burden. Secondly, we present a study that evaluates the extent to which the proposed approach enhances student learning in such a mixed-ability environment. The study was conducted in a first-year course at the University of British Columbia – Okanagan, and it was based on three components: 1) a survey of 25 Likert items ($n = 46$), 2) class average grade and pass rate over 6 years ($n = 42 + 38 + 56 + 79 + 90 + 74$), and 3) student ratings of the course over 5 years ($n = 42 + 38 + 56 + 79 + 90$). Findings of the survey indicate an overall positive students' impression with no significant difference in the opinions of various student populations. Analyzing the course grades, pass rates, and student ratings confirmed the survey findings and showed an overall improvement in grades, pass-rates, and student satisfaction.

Keywords Programming for all · CS1 · Active learning · Flipped classroom · Mixed-ability classes

✉ Abdallah Mohamed
abdallah.mohamed@ubc.ca

¹ Department of Computer Science, University of British Columbia-Okanagan, Kelowna, BC, Canada

1 Introduction and motivation

Computer programming has been playing an integral role in many industries. As a skill, programming has increased in value across many disciplines and audiences beyond merely computer scientists. It is, therefore, no surprise that first-year computer science (CS1) programming courses attract a diverse crowd of students with mixed abilities and backgrounds (Dawson et al., 2018; Fassbinder et al., 2015). Records at the University of British Columbia – Okanagan (UBC-O), for example, show that students from Computer Science, Management, Engineering, Media Studies, Arts, and Human Kinetics were enrolled in one or more CS1 programming courses. Teaching the same material to such mixed-ability classrooms poses an interesting challenge to the instructor. This challenge must be carefully addressed as many students are already struggling to learn to program (Bennedsen & Caspersen, 2007; Lahtinen et al., 2005; Robins et al., 2003), and these courses already suffer from high failure and attrition rates (Beaubouef & Mason, 2005; Bennedsen & Caspersen, 2007; Watson & Li, 2014).

Traditional teaching style has been argued to be insufficient in supporting students in mixed-ability classrooms (Koutselini, 2006; Valiande & Koutselini, 2009). Instructors need to find creative ways to reach all students and increase their motivation and success. Several technical reports approach this problem from different perspectives and in different contexts. Tomlinson (Tomlinson, 2017) suggests using differentiated instruction for kids in grade school, and she also provides guidelines for teachers to tailor their instruction to reach individual learners. Haatainen et al. (Haatainen et al., 2013) considered differentiated instruction and social barriers (i.e., whether students felt comfortable requesting help) when teaching CS1 courses. Salli-Copur (Şalli-Çopur, 2005) suggests using class activities and extra homework, and to base the teaching on a meaningful context which appeals to all students. Roberts (Roberts, 2016) argues that collaborative learning can mutually benefit both lower-ability and more-capable students.

On the other hand, the flipped classroom model (Talbert, 2017) has shown a promise to enhance student learning and success in general (Bishop & Verleger, 2013; Knutas et al., 2016; Lage et al., 2000; Rosiene & Rosiene, 2015). In this model, students are introduced to learning materials before class to allow using the class time for problem-solving activities and peer instruction. The flipped model has been adopted in various disciplines (e.g., (Lockwood & Esselstein, 2013; Maher et al., 2015; Moravec et al., 2010; Toto & Nguyen, 2009)), and it has proven to be successful for teaching programming in the CS domain (e.g., (Çakıroğlu & Öztürk, 2017; Elmaleh & Shankaraman, 2017; Fassbinder et al., 2015; Indi, 2016; Zhuo & Qi-Xian, 2015)).

In this paper, we present a successful flipped-based course design, named T-MACS1, for Teaching Mixed-Ability CS1 programming classes. The design uses several pedagogical strategies that benefit from the mixed-ability nature of the student body. We have been successfully using this design for the past three

years in a highly mixed-ability first-year course, namely *COSC 123: Computer Creativity*.

The paper also presents a study that confirms the effectiveness of the approach in terms of student learning and satisfaction. The study involves a survey that is used to get insights into student perceptions of the flipped design and how it affected their learning inside and outside of the classroom. The survey also offers a platform to investigate whether there was a significant difference between the responses of different student populations. To validate the survey findings, we compared the class average grades, pass rates, and student ratings of the new design against previous offerings of the same course, taught by the same instructor using the traditional teaching style.

The rest of this paper is structured as follows: Section 2 gives a brief overview of COSC 123 and summarizes the variances among its student populations. Section 3 presents the proposed T-MACS1 approach. Sections 4 to 6 present the study and its findings. Section 7 discusses the limitations, Section 8 includes the conclusions and our future work.

2 COSC 123: A highly mixed-ability course

COSC 123 (Computer Creativity) provides students with a hands-on introduction to programming and computer-based problem solving and creativity. The course aims to attract non-CS majors, women, and underrepresented groups in order to motivate them to try computer science by emphasizing the creativity aspect of programming. With the course growing significantly, it now has a very broad and mixed-ability student body, which can be very challenging to teach. Such large diversity can be described in terms of the following aspects:

- a) *Diversity in programming background:* COSC 123 has one of two prerequisites, either COSC 122 or COSC 111 (Fig. 1):

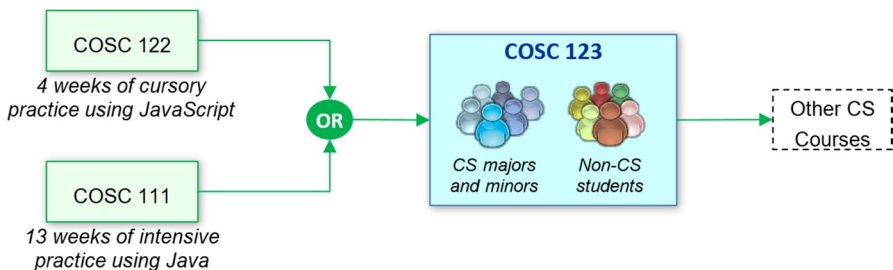


Fig. 1 Prerequisite structure resulting in the highly mixed-ability nature of COSC 123

- COSC 122 (Computer Fluency) teaches students the basics of programming using JavaScript during one-third of the semester. The remaining time is used to introduce students to other CS fundamentals such as databases and security. Students finishing this course usually have little programming proficiency. Most non-CS students and some CS students use COSC 122 as a prerequisite for COSC 123.
 - COSC 111 (Computer Programming I) uses the entire semester to teach algorithmic thinking and programming using Java. Students finishing this course are much more proficient in programming compared to those who only took COSC 122. Most CS students use COSC 111 as a prerequisite for COSC 123.
- b) *Diversity in motivation, learning needs, and educational level:* COSC 123 is part of several programs at UBC-O that require different levels of computer-programming proficiency. The course was mandatory in the Bachelor of Media Studies,¹ an elective in several disciplines, and part of CS majors and minors in Science, Arts, Engineering, Data Science, and Management. Observations and interviews with students over six years have shown that their learning needs and motivations to take the course are notably different. For example, while many students were interested in improving their programming skills, others were inclined to the creativity aspect of the course or just getting an easy credit (e.g. senior students).

Based on points (a) and (b) above, the student body in COSC 123 can be loosely classified into four populations:

- P1: CS students who finished COSC 111: usually good programmers with high motivation.
- P2: CS students who finished COSC 122: usually weak programmers with high motivation.
- P3: non-CS students who finished COSC 122: often weak programmers; motivation varies but usually low.
- P4 non-CS students who finished COSC 111: often good programmers; motivation varies but usually low.

2.1 The older course design

Traditional lecturing was used to teach COSC 123 from 2009 (when it was first offered) to 2017. This was a suitable method back then due to the small class sizes and little student diversity. The average number of registered students was 33 from 2009 to 2016. However, this number has significantly increased since then. We had 56 students in 2017, and then an average of 82 students from 2018 to 2020. Although this is a positive change, it introduced more diversity and challenge, causing the traditional lecturing style to become less and less suitable. Some students felt

¹ In 2020, the course became an elective in this program.

overqualified and unhappy about the redundancy the course has with its prerequisite, and others struggled to develop valid programs, despite their familiarity with the code syntax and semantics. Students expressed these concerns both verbally and in their written evaluations of the course.

3 T-MACS1: A proposed approach

In 2018, we completely redesigned COSC 123 (Mohamed, 2019) in order to address students' concerns and handle the emerging mixed-ability nature of the course. The design process focused on three questions:

1. *How to teach programming fundamentals in a way that is new and appealing to everyone (especially those who already have good programming background as with P1 and P4 students)?* The aim here is to reach all students and keep them motivated and engaged.
2. *Which programming language to use?* COSC 123 is a prerequisite for a few other CS courses that use Java (Fig. 1). Consequently, Java might seem to be the logical choice. However, depending on students' previous experiences, some had a good knowledge of Java while others were intimidated to learn it. Therefore, we needed a language that is similar to Java (to help those who take subsequent CS courses) and, at the same time, simple and interesting enough to engage everyone.
3. *How to address the large variance in students' skills when working on class exercises?* The use of classroom activities proved to be a successful active learning strategy in the original course design, and therefore we decided to keep using this technique. However, we noticed significant time differences among student groups when finishing these activities, which sometimes resulted in more experienced students getting bored and even misbehaving while waiting for others who were still struggling to finish the same activities.

In our effort to answer the above questions, we partially flipped the course and we integrated five pedagogical strategies into a novel design that benefits from the mixed-ability nature of the course rather than considering it as a burden. These strategies along with the overall teaching approach (called T-MACS1) are shown in Fig. 2.

a) *Partially flipped classroom* **FC**

The flipped model was a key component and a perfect fit for our new course design. Learning programming focuses on algorithmic thinking and writing code. Hence, the time spent in class would be better used for active learning, where students develop algorithms and programs while getting feedback from others, rather than passively receiving information from the instructor.

To address the mixed-ability nature of the class, we used the partially flipped model (Knutas et al., 2016; Talbert, 2017). In our design, we flipped lectures with

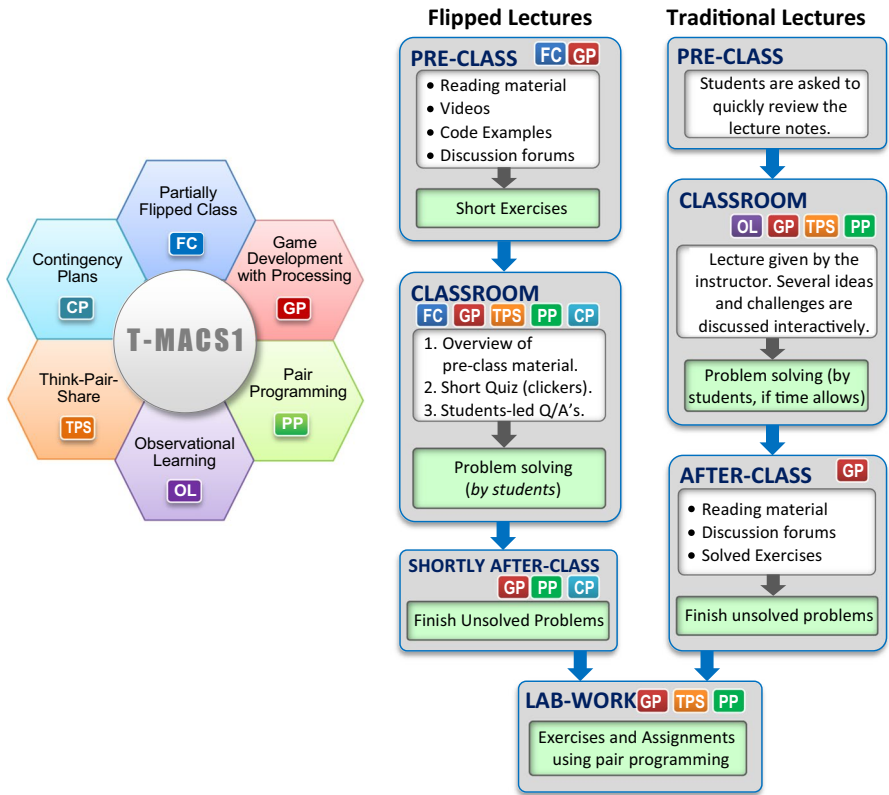


Fig. 2 T-MACS1 Approach: Key Elements (left) and Course Design (Right)

topics that we believed students could learn on their own, while we taught other topics using traditional teaching reinforced by observational learning (Shettleworth, 2010) (discussed in point (d) below).

The flipped lectures used pre-class material such as lecture notes, short videos, animations, and coding examples. The material was designed such that it: (1) lists specific learning objectives that guide students to what they need to learn, and (2) is not overwhelming, primarily keeping in mind the weaker programmers.

We used short quizzes (using clickers) at the beginning of each flipped lecture to assess student understanding and motivate them to finish the pre-class work. Students had to answer these quizzes individually. The first question was always on whether students had finished the pre-class material and understood them. Most students indicated that they had no issues finishing the pre-class work.

After the quizzes, we had open discussions on topics that were confusing to most students. Then, students were asked to use the remaining time of the class to finish coding assignments, with the deadline set at the end of the class or shortly after.

b) *Game development using processing* GP

We researched several programming languages and application domains and eventually chose Processing (Reas & Fry, 2006), an open source library and a programming environment that are used to code within the context of the visual arts. Processing uses simplified Java syntax to create visual, interactive media. The first steps in learning Processing involve creating and animating basic shapes using built-in functions (e.g. to display a circle located at (x,y) , use `circle(x,y,d)`, where d is the diameter). In our course, students were able to learn programming fundamentals while developing interactive animations and games, wherein they had to algorithmically solve problems related to game design, e.g., to use conditionals to check if two-game items collide – see the example in the [appendix](#).

By using Processing, we emphasized topics that were previously taught in the prerequisite courses while simultaneously introduced new topics within an engaging application domain that appeals to most, if not all, senses and competencies in the class (Guzdial, 2010; Leutenegger & Edgington, 2007). Despite the differences among students in terms of language proficiency or previous experience, this programming environment was new and interesting, and hence enhanced the motivation of almost everyone.

An additional benefit of using Processing was to prepare students to join other CS courses that also use Java. As Processing only uses simplified Java syntax, we dedicated one lecture at the end of the course to help students with the transition from Processing to the standard Java development environment.

c) *Pair programming* PP

This technique has been proven to be successful in supporting the flipped teaching in CS education (Maher et al., 2015). Studies show that using Pair Programming in CS courses leads to better learning and retention rates (McDowell et al., 2006; Porter et al., 2013). In this paradigm, students are paired together to develop a program. One student works as a driver and the other as an observer, frequently swapping roles while sharing ideas and giving instructions to each other.

In our design, we addressed the large variance in students' skills by pairing students such that each group had a strong programmer paired with a weak one. We used the prerequisite courses to identify strong and weak programmers. In a few cases, we did not have enough students who finished COSC 111 (i.e. strong programmers), so we asked about other prior programming experiences (e.g. high-school CS courses) and used that information for balancing the programming skills across all groups. Whenever students were engaged in class exercises, the instructor's role was to monitor and guide different groups while ensuring that students were alternating roles and supporting each other.

d) *Observational Learning* OL

Several topics were taught using the observational learning technique (Shettleworth, 2010) in which the instructor solved coding exercises step-by-step, and encouraged student teams to code with him and to provide suggestions on the next steps.

The new programming environment and application domain (i.e., Processing and visual arts) worked very well with the observational learning technique. Almost everyone was actively engaged as many students found it fun to work with the instructor and visualize the results almost instantly on their own machines, and then experiment with the code in order to alter or enrich their visual results.

We also found that observational learning was very beneficial in combination with pair-programming. Students were able to receive instant feedback from their peers while coding with the instructor. This helped them keep pace and motivated them to finish, and even extend, the work at hand.

e) *Think-pair-share* TPS

We frequently used the think-share-pair cooperative structure (Kothiyal et al., 2013; Lyman, 1987) in both traditional and flipped lectures in a way that benefits from the mixed-ability nature of the class. In our design, the pairing part was done at two levels: first, student-teams discuss internally, and then each team exchange ideas with other teams. This helped to actively involve all students and allowed them to see the problem from different angles and to learn from each other.

f) *Contingency plans* CP

To respond to those who finish the class exercises sooner than others, bonus tasks were prepared as contingency plans. Tasks included going deeper into the lecture topics and working on more creative designs or complex interactions within the animation or game at hand. These exercises were made available to students to submit either during the lecture or after class.

4 Validation

In order to explore the extent to which the T-MACS1 approach enhanced students learning in COSC 123, as an example of a highly mixed-ability first-year programming course, we conducted a study that is based on three components:

1. *A survey* (25 Likert items completed by 46 students).
2. *Class average grade and pass rate* (3 years of the new design compared to 3 years of the old design).
3. *Student ratings of the course* (2 years of the new design versus 3 years of the old design).²

² Student evaluations of the course was suspended in 2020 due to COVID-19.

Section 5 covers point (1), and Section 6 covers (2) and (3).

5 The survey: Capturing student perceptions

The survey aims to get insights on how students perceived the usefulness of the flipped course design. With the study objective in mind, the following research questions were formulated:

- RQ1: *What are the overall student perceptions of the partially-flipped T-MACSI design?*
 RQ2: *How does the flipped model affect student learning inside a mixed-ability classroom?*
 RQ3: *To what extent does the flipped model improve a student’s self-directed learning process?*
 RQ4: *What are student perceptions of some of the techniques used to support the flipped model?*

In addition, we investigate whether there is a significant difference between the responses of different student populations enrolled in the course.

5.1 Survey design

We used an online questionnaire to collect data from students on the above questions. To ensure that we have no biased results, no incentives or penalties were given to participants, and the course instructor was not allowed access to collected data until the grade appeal process was completed. An unbiased third-person explained the study procedure to students and invited them to participate, and then collected and stored data on behalf of the course instructor.

The survey was completed by 46 students whose profiles are shown in Tables 1 and 3. Participants were composed of 25 (54.3%) good programmers and 21 (45.7%) weak programmers. 29 out of all participants (63%) were majoring in CS or DS. Of all students, only 47.8% were majoring in CS or DS and at the same time had a strong programming background. This means that we had a demographic consisting of students in the course

Table 1 Profile of participants and survey reliability

Participants' Profile		Participants #		Cronbach's α
		Count	%	
Programming background	Strong programmer	25	54.3%	0.81
	Weak programmer	21	45.7%	0.83
Major	Comp. Science (CS)	26	56.5%	0.79
	Data Science (DS)	3	6.5%	
	Non-CS/DS majors	17	37%	
Overall		46		0.82

Table 2 Relationship of research questions to questionnaire items

Research Questions	Aspect	Questionnaire Items
RQ1.General perception	Overall liking of the flipped design	1–4
	Partially-flipped or fully-flipped?	5, 6
RQ2.Classroom learning	Confidence level with study material	7, 8
	Engagement and communication	9, 10
RQ3.Self-learning	Ability to work on one’s own	11–13
	Understanding versus memorization	14, 15
	Time and effort requirements	16–19
RQ4.Support techniques	Observational learning	20
	Pair-programming, learning from peers	21–23
	Clickers	24
	Video lectures	25

who were majoring in CS or DS, but were considered weak programmers. 39 (85%) participants were in their first or second year of study; the other 7 (15%) were upper-year students, with 6 in their third year and 1 in the fourth year. We assume that students who finished COSC 111 (as a prerequisite course) are good programmers and others who did not finish COSC 111 do not have sufficient programming background (weak programmers). We also assume that Computer Science (CS) and Data Science (DS) majors have a higher motivation to learn to program than non-CS majors. These assumptions were confirmed based on several discussions with students.

The 25 Likert items used a scale from 1 to 5 (5 means strongly agree, and 1 strongly disagree), and they were distributed across the four research questions as listed in Table 2. The first group (6 items) evaluates the students’ general perception of the flipped design as implemented in this course (RQ1). The second group (4 items) focuses on the usefulness of the flipped model during the lecture time (RQ2). The third group (9 items) studies the effect of the flipped design of the course on the students’ self-directed learning process (RQ3). The fourth group (6 items) measures the students’ perception of the support techniques used in the course (RQ4).

The reliability of the data was evaluated using Cronbach’s alpha coefficient (Şallı-Çopur, 2005) which was computed 5 times: once based on the responses collected from all 46 participants, and then once for each student group. Table 1 shows Cronbach’s alpha scores which, as can be seen, indicate acceptable levels of internal consistency.

5.2 Survey findings and discussion

5.2.1 Overall results

Table 3 lists the questionnaire items and student responses. The first three columns include our research questions and related questionnaire items. The middle section shows the number of student votes under each Likert level, with a mini-chart included for convenience. Finally, the right section includes the mean and standard deviation based on all responses.

Table 3 Student responses

		SA (5)	A (4)	N (3)	D (2)	SD (1)	Mean (μ)	SD (σ)
RQ1: Overall perception	1 The flipped learning model has significantly contributed to improving my learning in this course.	9	22	12	3	0	3.8	0.82
	2 I would recommend flipped classroom (or partially flipped classroom) to a friend.	11	24	6	4	1	3.87	0.95
	3 More courses should use partially-flipped classrooms.	12	16	10	7	1	3.67	1.08
	4 I hated the flipped model the way it was implemented in this class.	0	2	7	11	26	1.67	0.89
	5 I wish all lectures in this course were flipped (as opposed to partially flipped as used in this course)	2	6	17	16	5	2.65	0.98
	6 I recommend partially-flipped classrooms for this course as opposed to fully-flipped classrooms.	12	24	7	1	2	3.93	0.94
RQ2: In-class learning	7 I felt very confident about the material after finishing the pre-class work and before coming to class.	6	20	14	6	0	3.57	0.88
	8 I felt very confident about the material after finishing the pre-class work and after practicing in class.	14	21	9	2	0	4.02	0.82
	9 Flipped learning is more engaging than traditional lectures.	16	18	8	4	0	4.00	0.93
	10 Flipped learning allows more opportunities to communicate with classmates compared to traditional mode.	16	20	7	3	0	4.07	0.87
RQ3: self-learning	11 The flipped model helped me develop the ability to plan my own work.	8	16	18	4	0	3.61	0.87
	12 The flipped model helped me be more confident about self-learning and tackling unfamiliar problems on my own.	6	24	11	5	0	3.67	0.84
	13 It is often hard to discover what's expected of you when finishing the pre-class readings	0	4	13	21	8	2.28	0.85
	14 The flipped model made me focus more on memorization rather than understanding.	1	4	4	28	9	2.13	0.9
	15 To do well in this course, all I really needed was a good memory	1	4	10	25	6	2.33	0.89
	16 The sheer volume of work required for the flipped model means I can't comprehend course materials thoroughly	3	5	12	18	8	2.50	1.1
	17 The work required for the flipped learning was too heavy	1	7	11	21	6	2.48	0.97
	18 I was generally given enough time to understand the material we have to learn in this course.	15	22	7	2	0	4.09	0.8
	19 The amount of time I spent on this class was more than the time I usually spend on a similar class that uses traditional lecturing.	4	12	15	12	3	3.04	1.06
RQ4: support techniques	20 Coding in class (as opposed to watching the instructor explain the code) helped me grasp course material.	23	17	3	3	0	4.30	0.86
	21 Working with a classmate helped me develop basic teamwork skills	16	21	9	0	0	4.15	0.72
	22 Pair programming was very helpful in this class.	17	21	7	1	0	4.17	0.76
	23 The support I received from my team-mate is not enough. We should have TAs in class to help us.	3	5	12	18	8	2.50	1.1
	24 Clicker quizzes at the beginning of class motivated me to finish the pre-class work before coming to the lecture.	18	19	5	4	0	4.11	0.91
	25 I wish we had more video lectures to support the pre-class readings.	3	15	18	9	1	3.22	0.91

(SA: Strongly Agree, A: Agree, N: Neutral, D: Disagree, SD: Strongly Disagree)

Overall student perception of the partially-flipped model in a mixed-ability classroom As seen in Table 3, students had a positive impression of the flipped model. 67.4% agreed ($\mu=3.8$, $\sigma=0.82$) that the flipped design has significantly contributed to improving their learning in this course, and only 4.3% ($\mu=1.67$, $\sigma=0.89$) reported that they hated the flipped model. 76.1% of students agreed ($\mu=3.87$, $\sigma=0.95$) that they would recommend this model to a friend, and 17.4% disagreed with implementing the flipped design in other courses.

Most of the students preferred the partially-flipped model over fully flipping the course (items #5 and #6 in Table 3). The written feedback of the course evaluation showed that many students liked the idea of flipping the easy-to-self-learn topics while using the traditional lecturing style for the harder topics.

The flipped model as implemented in this course supports classroom learning As shown in items 9 and 10 in Table 3, the majority of participants agreed that the flipped model is more engaging and allows for better communication with their peers compared to the traditional lecturing style. Furthermore, 76.1% agreed ($\mu=4.02$, $\sigma=0.82$) that they feel more confident about course material after practicing in class versus only 53% of the students ($\mu=3.57$, $\sigma=0.88$) who felt confident after only reading the pre-class material.

The flipped course design positively affects student self-directed learning process The data shows that 30 students (65%) agreed ($\mu=3.67$, $\sigma=0.84$) that they were not only more confident with their self-learning but also better equipped to tackle unfamiliar problems as a result of the proposed design, and only 10.9% disagreed with that statement. Surprisingly, 4 out of the 30 students who agreed were upper-year students, which is over half of the upper-year students surveyed in this study.

In terms of study techniques, the majority (80.4%, $\mu=2.13$, $\sigma=0.9$) disagreed that the flipped model made them focus on memorization rather than understanding. The data also shows that most students agreed that the amount of time and effort that they had to put into this course was reasonable, with 52.2% agreeing that the flipped design helped them plan their work. However, there was no general consensus in terms of the time required in our course compared to similar courses that use traditional teaching methods.

Positive student perception of techniques used to support the flipped design The majority of students agreed that pair-programming helped them grasp the course material and develop teamwork skills (87%, 80.5%, and 82.7% for items #20, 21, and 22 respectively). In addition, only 17.4% agreed that they need a TA in the classroom to help them due to the lack of support they received from their classmates. On the other hand, 80.4% agreed that clicker quizzes on the pre-class readings motivated them to finish the pre-class work before coming to class. Finally, 60.9% agree that more video lectures should be prepared to support their pre-class work.

5.2.2 Comparing various student populations

Two independent sample t-tests (which assumes different variances) were conducted; the first was to compare responses from the two student groups: strong programmers (SP) and weak programmers (WP); and the second t-test for the two student groups: CS-majors and non-CS majors. The results are shown in Table 4 and Table 5. At 0.05 significance level, we found that:

Table 4 Comparing responses based on programming proficiency

	SP ($n=25$)		WP ($n=21$)		Significance	
	μ	σ	μ	σ	t-test	p (2-tailed)
RQ1	3.83	0.59	3.81	0.44	0.20	0.846
RQ2	4.08	0.55	3.71	0.56	2.22	0.035
RQ3	3.74	0.47	3.50	0.45	1.76	0.092
RQ4	3.85	0.39	3.98	0.44	-1.12	0.282

Table 5 Comparing students' responses based on their major

	CS (<i>n</i> = 29)		Non-CS (<i>n</i> = 17)		Significance	
	μ	σ	μ	σ	t-test	<i>p</i> (2-tailed)
RQ1	3.76	0.61	3.93	0.33	-1.18	0.243
RQ2	3.92	0.63	3.90	0.49	0.15	0.883
RQ3	3.68	0.55	3.55	0.30	1.04	0.303
RQ4	3.79	0.37	4.12	0.42	-2.83	0.013

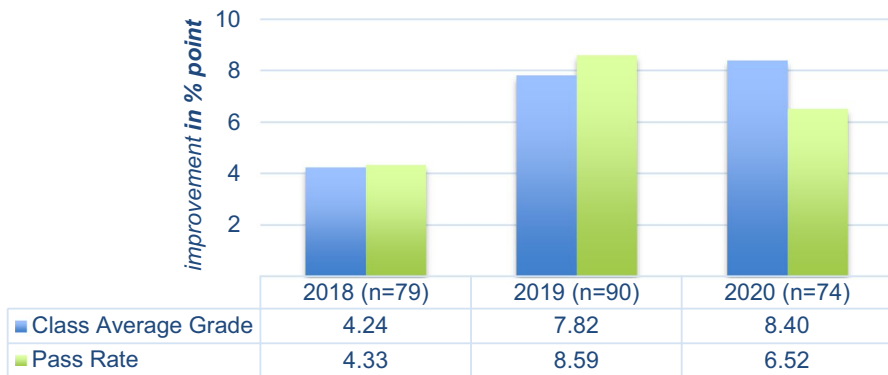
- There is no significant difference between the two groups SP and WP for questions RQ1, RQ3, RQ4. However, these two groups are significantly different in terms of RQ2 (classroom learning) with the SP group outscoring WP.
- There is no significant difference between CS and non-CS majors in terms of questions RQ1, RQ2, RQ3. On the other hand, there was a significant difference for RQ4 (support techniques) with the non-CS majors outscoring CS majors.

6 Student grades, pass rates, and evaluation of the course

We analyzed the course grades, pass rates, and the end-of-term course ratings in order to confirm the survey findings. Three course-offerings that used the new design (2018 to 2020) were compared to three previous offerings that used the older design (2015 to 2017). All offerings were taught by the same instructor, and the assessment components (e.g. lab work and exams) were prepared to ensure the same level of difficulty and to focus on the same learning outcomes.

6.1 Grades and pass rate

Figure 3 compares student performance in the new design versus the old design. Not only did the pass rate considerably increase compared to the mean of pass rates in the old design, but the class averages had also increased. As shown below, the class average

**Fig. 3** Improvement in New Course Offerings Compared to the Old Design

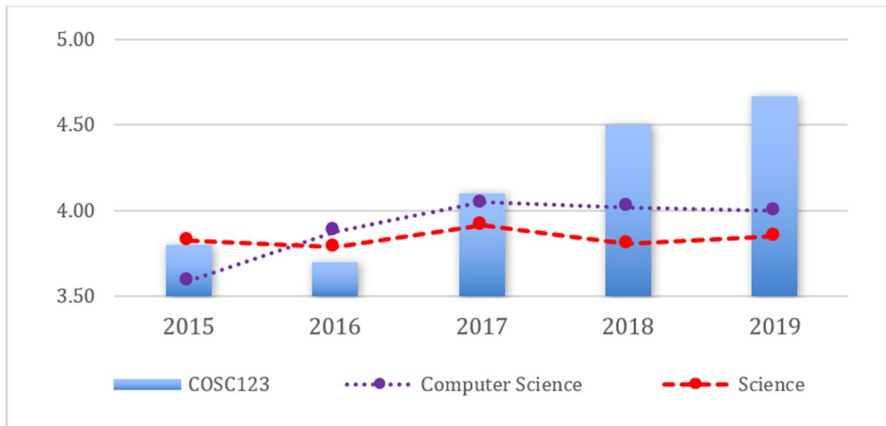


Fig. 4 Mean Ratings for COSC 123 from 2015 to 2019 ⁴

grade in the new design had a 4.24, 7.82, and 8.40 percentage point increase in 2018 to 2020 respectively, compared to the mean of class averages from 2015 to 2017. We believe the “jump” in results from 2018 to 2019 is due to instructor’s gained experience with the new design.

6.2 Student evaluation of the course

With regard to student satisfaction, the new design received a higher rating.³ Figure 4 shows the mean of student ratings for the course from 2015 to 2019 as measured by the question: “I would rate this course as very good”, with ratings from 5 (strongly agree) to 1 (strongly disagree). As can be seen, there was a notable increase in 2018 and 2019, when the new design was implemented.

Figure 4 also illustrates the norms in the Faculty of Science and the Department of Computer Science at UBC-O with the above question. Unlike the ratings of the old design, which were within the norms, the new design had a considerably higher rating.

In the written feedback of the course evaluations, most students praised the new design, teaching style, and application domain. Students indicated that while the course was an excellent introduction to coding for those with a little programming background, it was still enjoyable for those already familiar with programming because of the use of a “visually engaging environment” to develop animations and games. In addition, most students commended programming in pairs (a handful felt that teamwork was stressful or useless). Most students also felt the course pace and structure were appropriate except for a handful who complained that the course was too easy or slow, but they still liked the bonus parts, which presented them a challenge and helped them sharpen their skills.

³ Response rate is about 50%.

7 Limitations

While the survey shows positive results in favor of the proposed flipped design, we recognize that these results are based on the data collected from only 46 students. More studies and further research must be conducted to confirm the findings and gain more in-depth insights.

The results obtained from the course grades, pass-rates, and student evaluations are based on three course-offerings that use the new design versus three offerings of the old design. Given the fact that student mix changes from year to year and course to course, these results also need further validation, which is part of our future work.

Furthermore, the data from the year 2020 (column 3 in Fig. 3) were measured during the COVID-19 pandemic where we had to switch to online teaching in the midst of the semester. This caused had some impact on the accuracy of student evaluation (i.e. grades from that year may not be the most reliable). However, as the course grades were consistent with the previous two years, we decided to include them here.

8 Conclusions and future work

This paper introduced a flipped based approach for teaching highly mixed-ability CS1 courses. The approach is based on the partially-flipped classroom model, and it uses several techniques, including pair-programming, observational learning, and game development, in order to teach programming fundamentals in the visual arts domain using Processing language.

The paper extends existing findings on the usefulness of the flipped model by showing how combining it with other techniques is effective in teaching highly mixed-ability CS1 classrooms.

The proposed model was evaluated in terms of its impact on the student learning experience and performance. We used three evaluation components: a survey, the class average grade and pass-rate, and the student ratings of the course. The survey showed an overall positive student perception, and the course grades, pass rates, and student ratings confirmed the survey findings.

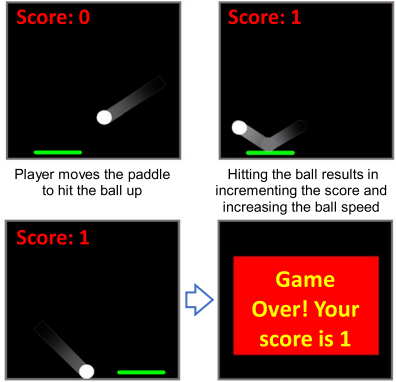
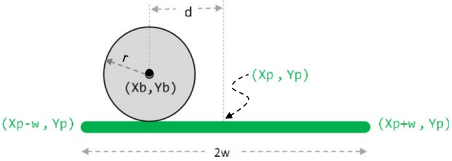
Our future work includes testing the approach in other classes taught by the current or other instructors. This includes testing the approach in a fully online course-mode.

It is worth mentioning that switching COSC 123 to online learning amid the pandemic semester was relatively easy. The flipped nature of the course, with much of its material already assigned as out-of-class work, helped students adapt to the new format quickly. The transition was not ideal, but it was smoother than other courses taught by the same instructor. A blend of synchronous and asynchronous lectures was used to implement the online format of T-MACS1, and we used breakout rooms to enable students to work in pairs.

Our future work also includes testing other motivational techniques and continuing to report more results as class size increases, student populations vary, and technology changes.

Appendix

To give a feeling of the programming environment used in the proposed design, we include in this appendix two examples of the programming exercises used when teaching conditionals, one from the old course design and one from the new one.

Example from the Old Course Design	Example from the New Course Design
<p><i>Below is an assignment question on validating the Social Insurance Number against a set of given rules.</i></p> <p>Most residents have a Social Insurance Number (SIN). The format of a SIN is as follows:</p> <ul style="list-style-type: none"> • Every SIN consists of nine digits • First Digit - a code indicating the region of origin <ul style="list-style-type: none"> 1 - Issued in Atlantic Provinces 2 or 3 - Issued in Quebec 4 or 5 - Issued in Ontario 6 - Issued in the Prairie Provinces 7 - Issued in the Pacific Region <p>Last Digit - a check digit Based on other 8 digits to indicate whether the other 8 digits are (probably) correct. SINs are long and often recorded incorrectly. Check digits can spot incorrect SIN quickly (tax programs for example, will not let you enter an invalid SIN). This check digit also (in a limited manner) makes forging SINs more difficult. This check digit is described below.</p> <p>Example: Consider the SIN 430 837 013</p> <ul style="list-style-type: none"> • consider the first 8 digits • multiply every second digit by 2: <ul style="list-style-type: none"> $2 \times 3 = 6$ $2 \times 8 = 16$ $2 \times 7 = 14$ $2 \times 1 = 2$ • add each digit of the above together: <ul style="list-style-type: none"> $6 + 1 + 6 + 1 + 4 + 2 = 20$ • go back to original number and add remaining alternate digits together: <ul style="list-style-type: none"> $4 + 0 + 3 + 0 = 7$ • add these previous two totals together: <ul style="list-style-type: none"> $20 + 7 = 27$ • subtract this total from the nearest (highest) multiple of 10, which in this case is 30: <ul style="list-style-type: none"> $30 - 27 = 3$ <p>Therefore 3 should be the final, so-called check digit, of this number. Hence it is a (potentially) valid SIN. Write a complete Java program that reads a SIN number from the user and determine if it is valid or not. A starter program is here (link).</p>	<p><i>Below is a question to create a simple game that uses conditionals. Before giving this exercise, students were introduced to the idea of creating an animation with a moving ball that bounces off the edges of the window.</i></p> <p>In this question, you are required to build a game with a moving ball that bounces off the left, right, and top edges. A player can use the mouse to horizontally move a paddle located at the bottom of the window, trying to hit the ball when it reaches the bottom edge. If the player hits the ball with the paddle, the score is incremented by one. If the player misses the ball, the game ends with a message that shows the score. To make the game a bit challenging, the ball speed increases a little bit (e.g. by 10%) every time the player hits the ball (see the figure below).</p>  <p>Score: 0 Player moves the paddle to hit the ball up</p> <p>Score: 1 Hitting the ball results in incrementing the score and increasing the ball speed</p> <p>Score: 1 If ball touches bottom edge, the game is over, and the animation stops with the message shown above.</p> <p>Game Over! Your score is 1</p> <p><i>How to check if the ball lands on the paddle? When the ball is at the lower edge, measure the distance d as illustrated below. The ball lands on the paddle if d is less than $(w+r)$.</i></p> 

Acknowledgments This work is partially funded by I.K. Barber Endowment fund at the University of British Columbia-Okanagan.

Authors' contributions Please see the cover letter below.

Funding This work is partially funded by the I.K. Barber Endowment fund at the University of British Columbia-Okanagan.

Data availability Questionnaire raw data available upon request.

Code availability Not applicable.

Declarations

Ethical statement The study included in this paper was approved by the research ethics board at the University of British Columbia-Okanagan. This work is based on (Mohamed, 2019, 2020) as explained in the cover letter.

Consent statement Not applicable.

Disclosure of potential conflicts of interest No conflicts of interest.

References

- Beaubouef, T., & Mason, J. (2005). Why the high attrition rate for computer science students: Some thoughts and observations. *ACM SIGCSE Bulletin*, 37(2), 103–106.
- Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin*, 39(2), 32–36.
- Bishop, J. L., & Verleger, M. A. (2013). The flipped classroom: A survey of the research. In *American Society for Engineering Education Annual Conference and Exposition (ASEE)*, Atlanta, GA, (Vol. 30, pp. 1–18, Vol. 9).
- Çakıroğlu, Ü., & Öztürk, M. (2017). Flipped classroom with problem based activities: Exploring self-regulated learning in a programming language course. *Journal of Educational Technology & Society*, 20(1), 337–349.
- Dawson, J. Q., Allen, M., Campbell, A., & Valair, A. (2018). Designing an introductory programming course to improve non-majors' experiences. In *ACM symposium on Computer Science Education (SIGCSE)* (pp. 26–31). ACM.
- Elmaleh, J., & Shankararaman, V. (2017). Improving student learning in an introductory programming course using flipped classroom and competency framework. In *IEEE Global Engineering Education Conference (EDUCON)* (pp. 49–55). IEEE.
- Fassbinder, A. G. d. O., Botelho, T. G., Martins, R. J., & Barbosa, E. F. (2015). Applying Flipped Classroom and Problem-Based Learning in a CS1 Course. In *IEEE Frontiers in Education Conference (FIE)*.
- Guzdial, M. (2010). Does contextualized computing education help? *ACM Inroads*, 1(4), 4–6.
- Haatainen, S., Lakanen, A.-J., Isomöttönen, V., & Lappalainen, V. (2013). A practice for providing additional support in CS1. In *2013 Learning and Teaching in Computing and Engineering* (pp. 178–183). IEEE.
- Indi, T. S. (2016). An Experience Report of Flipped Classroom Strategy Implementation for Java Programming Course. In *IEEE 18th International Conference on Technology for Education (T4E)* (pp. 240–241). IEEE.
- Knutas, A., Herala, A., Vanhala, E., & Ikonen, J. (2016). The Flipped Classroom Method: Lessons Learned from Flipping Two Programming Courses. In *17th International Conference on Computer Systems and Technologies* (pp. 423–430). ACM.
- Kothiyal, A., Majumdar, R., Murthy, S., & Iyer, S. (2013). Effect of think-pair-share in a large CS1 class: 83% sustained engagement. In *ACM conference on International Computing Education research* (pp. 137–144). ACM.

- Koutselini, M. (2006). Towards a meta-modern paradigm of curriculum: Transcendence of a mistaken reliance on theory. *Educational Practice and Theory*, 28(1), 55–68.
- Lage, M. J., Platt, G. J., & Treglia, M. (2000). Inverting the classroom: A gateway to creating an inclusive learning environment. *Journal of Economic Education*, 31(1), 30–43.
- Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. In *10th SIGCSE conference on Innovation and Technology in Computer Science Education (iTiCSE)* (vol. 37, pp. 14–18, Vol. 3). ACM.
- Leutenegger, S., & Edgington, J. (2007). A games first approach to teaching introductory programming. In *ACM symposium on Computer Science Education (SIGCSE)* (pp. 115–118).
- Lockwood, K., & Esselstein, R. (2013). The inverted classroom and the CS curriculum. In *ACM symposium on Computer Science Education (SIGCSE)* (pp. 113–118). ACM.
- Lyman, F. (1987). Think-pair-share: An expanding teaching technique. *Maa-Cie Cooperative News*, 1(1), 1–2.
- Maher, M. L., Latulipe, C., Lipford, H., & Rorrer, A. (2015). Flipped classroom strategies for CS education. In *ACM symposium on Computer Science Education (SIGCSE)* (pp. 218–223). ACM.
- McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, 49(8), 90–95.
- Mohamed, A. (2019). Designing a CS1 programming course for a mixed-ability class. In the Western Canadian Conference on Computing Education (WCCCE'19), (pp. 1–6).
- Mohamed, A. (2020). Evaluating the effectiveness of flipped teaching in a mixed-ability CS1 course. In the 20th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'20), (pp. 452–458).
- Moravec, M., Williams, A., Aguilar-Roca, N., & O'Dowd, D. K. (2010). Learn before lecture: A strategy that improves learning outcomes in a large introductory biology class. *CBE-Life Sciences Education*, 9(4), 473–481.
- Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming: What works? *Communications of the ACM*, 56(8), 34–36.
- Reas, C., & Fry, B. (2006). Processing: Programming for the media arts. *AI & Society*, 20(4), 526–538 <https://processing.org>. Accessed Dec 2020.
- Roberts, J. (2016). The 'more capable peer': Approaches to collaborative learning in a mixed-ability classroom. *Changing English*, 23(1), 42–51.
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2), 137–172.
- Rosiene, C. P., & Rosiene, J. A. (2015). Flipping a programming course: The good, the bad, and the ugly. In *IEEE Frontiers in Education Conference (FIE)* (pp. 1–3). IEEE.
- Şalli-Çopur, D. (2005). Coping with the problems of mixed ability classes. *The Internet TESL Journal*, 11(8), 1–5.
- Shettleworth, S. J. (2010). *Cognition, evolution, and behavior*. Oxford University Press.
- Talbert, R. (2017). *Flipped Learning: A Guide for Higher Education Faculty*. Stylus Publishing, LLC.
- Tomlinson, C. A. (2017). *How to differentiate instruction in academically diverse classrooms* (3rd ed.). Alexandria: Association for Supervision and Curriculum Development (ASCD).
- Toto, R., & Nguyen, H. (2009). Flipping the work design in an industrial engineering course. In *IEEE Frontiers in Education Conference (FIE)* (pp. 1–4). IEEE.
- Valiande, S., & Koutselini, M. I. (2009). Application and evaluation of differentiation instruction in mixed ability classrooms. In *4th Hellenic Observatory PhD Symposium* (pp. 25–26): LSE, London School of Economics London, UK.
- Watson, C., & Li, F. W. (2014). Failure rates in introductory programming revisited. In *Innovation & technology in computer science education* (pp. 39–44). ACM.
- Zhuo, L., & Qi-Xian, G. (2015). The Application of Hybrid Flipped Classroom in the Course of Java Programming. In *7th International Conference on Information Technology in Medicine and Education (ITME)* (pp. 637–641). IEEE.