Investigating How Introductory Programming Students Apply Regulation Strategies

Deller James Ferreira[a] and Dirson Santos Campos[b]
Institute of Informatics, Federal University of Goiás, Alameda Palmeiras, Goiânia, Brazil

Keywords: Self-regulation, Co-regulation, Shared Regulation, Introductory Programming.

Abstract: Self-regulated learning is an important topic in introductory computer programming. Self-regulated learning is defined as the degree to which students are active participants in their own academic learning with respect to motivational, behavioral, metacognitive, and cognitive aspects. Another important aspect in programming learning is the social regulation of learning, in which students co-regulate or share regulation of their cognition, behavior, motivation and emotions, in situations of temporary coordination of regulation with colleagues or teachers. Therefore, teaching and learning approaches in programming do not prioritize skills aligned with self-regulation, co-regulation and shared regulation. Thus, the objective of this research is to unveil the extent to which introductory programming students apply regulation strategies during programming. An exploratory study involving 198 students, found evidence that a significant number of students do not engage themselves in regulatory strategies during learning introductory programming.

1 INTRODUCTION

Undergraduate courses in computing often face high levels of dropout and failure, especially in introductory programming courses. Programming is considered a difficult activity, due to the fact that programming is a complex problem-solving task that requires multiple cognitive demands on students (Loksa and Ko, 2021).

Introductory programming courses present many challenges for students, as they need to master a wide range of skills both in terms of developing programming skills and in terms of awareness and mastery of the program code development process (Falkner et al., 2014). The inefficient use of learning strategies, such as self-regulation and shared regulation, is one of the possible causes for a bad programming learning performance (Soares, 2021).

Self-regulated learning is an important topic in education, which involves the regulation of student motivation, engagement, cognition and metacognition. Self-regulated learning is defined as the degree to which students are active participants in their own academic learning with respect to motivational, behavioral, metacognitive, and cognitive aspects (Pintrich, 2000).

Bergin (2005) investigated the relationship between self-regulated learning and introductory programming performance and showed that self-regulated learning is a useful predictor of programming performance. Self-regulation includes skills such as monitoring one's processes, reflecting on whether the process is successful, monitoring understanding of important concepts, and identifying alternative strategies to solve problems (Loksa, 2020).

Another relevant aspect in programming learning is the regulation in the collaborative learning. Group regulation involves two aspects which are co-regulation and shared regulation. Shared regulation is understood as the social regulation of learning, in which students temporarily regulate their cognition, behavior, motivation and emotions in situations of temporary coordination of regulation with peers or teachers (Järvelä and Järvenoja, 2011). Co-regulation refers to the dynamic metacognitive processes through which one student helps regulate another student's cognition, behavior, motivation and

[a] https://orcid.org/0000-0002-4314-494X
[b] https://orcid.org/0000-0002-0878-8336

Ferreira, D. and Campos, D.
Investigating How Introductory Programming Students Apply Regulation Strategies.
DOI: 10.5220/0011659000003470
In *Proceedings of the 15th International Conference on Computer Supported Education (CSEDU 2023) - Volume 2*, pages 463-473
ISBN: 978-989-758-641-5; ISSN: 2184-5026

emotions, providing support in a transitional and flexible way (Hadwin et al., 2018).

In computer programming, shared regulated learning helps students improve their programming skills as it provides students with a set of external resources and skills, such as seeking social help, evaluating others' ideas, and monitoring tasks (Tsai, 2015).

Furthermore, in computer science teaching, it is important to prepare students for the challenges of later professional practice, as well as providing students with opportunities to develop self-regulation, shared regulation and co-regulation skills, through activities that enhance collaborative learning and active (Wang et al., 2013).

Some research suggests that regulated learning is a topic of great interest in computer education research, but there are few theories, models, or tools specific to the programming context (Prather et al, 2020; Szabo et al, 2020; Malmi et al, 2019).

Thus, self-regulation, co-regulation and shared regulation strategies for programming are still not well understood by computer science researchers or educators, making it difficult to successfully develop methods to promote self-regulation and shared regulation learning. Regarding programming teaching and learning, regulation is a recent topic, demanding further research (Soares, 2021).

2 RESEARCH QUESTION

Given the above mentioned, the objective of this research is to investigate in what extent introductory programming students apply regulation strategies during introductory programming. So, the research question that leads this work is:

Research Question: How introductory programming students apply regulation strategies during programming?

3 LITERATURE REVIEW

The regulation strategies in programming involve self-regulation, co-regulation and shared regulation. This section approaches some research regarding investigations on how students' self-regulation, co-regulation and shared regulation skills influence programming learning.

3.1 The Importance of Self-regulation for Programming Students

Two factors commonly associated with student success or retention are student engagement and motivation, which are linked to emotional and behavioral self-regulation skills. According to Abdullah and Yih (2014), motivation is one of the characteristics that influence the way students approach their learning, while engagement involves students spending time and effort on learning activities (Crisp et al., 2015). Programming students have inadequate time management skills, which leads them to complain about lack of time for their studies or problem solving (Pereira et al., 2021).

According to Schoeffel (2019), motivation is the stimulus for the desire to learn something or to participate and succeed in the learning process. Regarding motivational self-regulation strategies, Keller (2017) proposed four categories that are directly linked to it: attention, relevance, trust and satisfaction. The lack of regulation of motivation can cause a strong discrepancy between learning potential and performance. This explains why highly qualified students can perform poorly, while students with less potential can be among the best.

Regarding motivation in programming students, few studies have been conducted (Coto et al., 2022). However, there is research evidence suggesting that inappropriate teaching methods undermines learners' motivation during learning programming and pointing that there is a need for teaching methods involving self-regulation strategies to improve motivation (Darabi et al., 2022).

The literature also shows cognitive and metacognitive reasons for failure in programming courses. Among them, the need to face the challenge of mastering multiple concepts, skills and computing models to design, implement and test programs (Robins et al., 2003). Another cognitive reason why programming is difficult to learn, and one of recent interest in the computer science education research community, is the need to develop knowledge about the problem-solving process (Loksa, 2020).

Knowledge of the problem-solving process is the understanding of how to solve programming problems, and using a problem-solving strategy is a self-regulation skill of cognition. The problem-solving process includes skills such as knowing how to interpret and understand a programming problem (Wrenn and Krishnamurthi, 2019), designing and adapting algorithms, translating these algorithms into a programming language notation (Xie et al., 2019), verifying whether the implementation actually solves the problem through testing (Kaner and Padmanabhan, 2007) and how to debug a program when it does not do what was intended (Ko et al., 2019).

For example, even if someone is already mastering the basics of a programming language, strong self-regulation skills can help them recognize that they don't have a good understanding of how a for loop runs in Python. This can cause them to increase their understanding before continuing to write or review their code. Or, when someone is struggling to diagnose a flaw in their program, self-regulation skills can help them recognize they are struggling and seek expert guidance on how to more productively diagnose the problem. Programming research consistently shows that self-regulation skills are strongly associated with success in solving computational problems (Falkner et al., 2015).

Recent work shows, however, that most novices have poor self-regulation skills, which are associated with poor programming results (Hauswirth and Adamoli, 2017). An example of a cognitive self-regulation strategy for problem solving is problem interpretation. When students interpret the problem inaccurately, they are likely to use ineffective strategies or fail to solve the problem. It is reported in studies that students are often unable to identify and articulate the problem objective, requirements/constraints, and expected outcome. In other words, students lack self-regulation skills, especially related to task comprehension.

Among behavioral self-regulation strategies, effort management, time management, and help-seeking proved to be positively correlated with academic outcomes (Daradoumis, 2021). Effort management strategies help students focus their attention on the task at hand and use their effort to achieve it effectively. During this process, students acquire skills that enable them to deal with failure, persist and overcome difficulties. To this end, these strategies foster motivation and commitment to accomplishing your goals, even when there are problems or distractions.

Time management strategies allow students to acquire skills related to setting goals and priorities, planning, self-monitoring, conflict resolution, negotiation, task assignment, negotiation and problem solving. Students succeed in time management if they can maximize their use of time to facilitate academic performance, balance, and satisfaction (Daradoumis, 2021).

Help-seeking strategies involve processes of seeking help from other people, such as the teacher or peers, or other sources that facilitate the achievement of desired goals in a learning environment. These strategies are associated with student engagement and can help students not only to meet their immediate learning needs but also to improve their performance by acquiring knowledge and skills and alleviating difficulties, which ultimately improves understanding, performance and subsequent independence (Daradoumis, 2021).

Regarding metacognitive strategies, reflective learning helps students to become more aware of the learning process and its difficulties (Chang, 2019). When students do effective self-reflection, they analyze how they learned, how they understood the goals of the learning process, and what it takes to create the conditions for success.

Reflection also encourages students to think critically about their abilities and reflect on strategies to improve the learning process, making them aware of the advantages of learning in the future and helping them to develop transversal skills (Chang, 2019). The interaction between students' commitment, self-control, autonomy and self-discipline allows them to regulate their own actions to achieve their learning goals.

On the other hand, reflective learning provides feedback to teachers, allowing them to readjust their pedagogical experiences and tools. The use of the reflective diary is a technique that reinforces and stimulates reflection on the theoretical and practical component of the work. In this context, to be successful, students need the required disciplinary knowledge, as well as develop self-regulation strategies (Falkner et al., 2014). Developing self-regulatory strategies is vital to helping students achieve success. A self-regulated learner will define their goals, organize their resources, and then manage their time effectively. Without this fundamental level of metacognition, they cannot direct their knowledge in a useful and constructive way.

3.2 The Importance of Co-regulation and Shared Regulation for Programming Students

According to Cheng et al (2021), in addition to having computer skills, students must also have collaborative problem solving and teamwork skills. Most computer science students arrive in the job market without the necessary skills to meet employer expectations, such as teamwork and the ability to cooperate (Pedrosa, 2019).

Although students acquire remarkable theoretical knowledge throughout the course, they lack transferable skills, such as soft skills, which are rarely addressed in project management teaching (Groeneveld, 2019). The ever-changing landscape of software development requires computer scientists to be equipped with skills beyond technical skills, such as self-reflection, conflict resolution, communication, teamwork, and creativity.

Collaborative learning is an approach that benefits everyone involved, both in developing the ability to work in groups and in sharing ideas and experiences. Collaborative learning brings some advantages over individual learning, mainly the possibility of exchanging ideas and clarifying doubts due to the interaction between students in a collective and social scenario (Cukierman and Palmieri, 2014).

Peer learning is an active learning approach where students simultaneously learn and share knowledge together, for example, engaging students by asking questions and promoting debate, providing appropriate and constructive feedback, promoting reflection and adapting practices. students' pedagogical practices to enrich their learning.

In higher education, collaborative learning is a useful and valuable strategy, as it trains students in professional activities in which they work in groups, providing students with several possibilities for interaction, for example, questioning, exchange of opinions and discussions. In addition, the synergy between the group allows students to improve their programming problem-solving skills (Chorfi, 2020).

Some authors have emphasized the usefulness of collaboration in programming learning activity (Hwang et al. 2012), especially with respect to motivating students and improving participation in activities. During collaborative learning sessions, students achieve their learning objectives through, for example, assignments, working together, and sharing skills.

When comparing collaborative learning with traditional learning, it is important to note that in the context of programming learning, collaboration encourages the exchange of ideas among students and allows them to develop better learning processes, skills, and outcomes (Hwang et al. 2008).

Regarding problem solving time, many authors (McDowell et al. 2002) indicate that students who learn in groups will consume less time to answer a programming problem and make better solutions than if they learned alone. Therefore, to succeed in collaborative learning, students must acquire skills in co-regulation and shared regulation.

4 RESEARCH METHOD

4.1 Procedures

In this work, we applied a 5-factor Likert scale (Likert, 1932) questionnaire for data collection. A questionnaire was designed to collect data on students' perceptions of the use of regulatory strategies, in order to assess the familiarity and frequency of use of self-regulation, co-regulation and regulation shared by introductory programming students.

The questionnaire is divided into two main parts. The first part was designed to assess whether self-regulation strategies are used by students, according to their perceptions. The questions in the first part were based on the Motivated Strategies for Learning Questionnaire (MSLQ) (Pintrich et al., 1983), being adapted to the context of programming learning. The second part was developed to measure group regulation in programming, again according to the students' own perceptions. The questions in the second part were based on the Adaptive instrument for Regulation of Emotions (AIRE) (Järvenoja et al., 2013).

Two essential aspects when designing a questionnaire are its validity and reliability. The validity of an instrument refers to its ability to measure what it was designed for, while reliability refers to the extent to which items on the test or instrument are measuring the same thing (Prous *et al.*, 2009). In order to attest to the validity and reliability of the questionnaire qualitative and quantitative methods were used.

First, there was the participation of six experts in the computer science area to analyze whether the selected questions were simple, clear, easy to understand, cover relevant aspects of self-regulation, co-regulation and shared regulation in programming and are comprehensive enough. The experts' evaluation is a validation of the questionnaire by observation.

Second, we applied the questionnaire to students in introductory programming courses for data collections and analysis. Third, in order to verify the reliability of the questionnaire, we applied the Cronbach's alpha statistical test (Cronbach, 1951) to check if the questions on self-regulation were consistent with each other, and also to check if the questions on co-regulation and shared regulation were consistent with each other.

4.2 Participants

Respondents to the questionnaire were 198 undergraduate students of computer science, computer engineering, medical physics, physical engineering, statistics and electrical engineering courses. The age range of the responding students was between 17 and 36 years old, predominantly between 18 and 19 years old. 86.6% of the students self-declared as being male, 12.5% of the students self-declared as being female and 0.9 of the students self-declared as being of the other sex, that is, neither male nor feminine.

Regarding professional and academic experience involving programming skills outside the classroom, 15.9% said they had no experience, 21.4% took extracurricular courses, 7.1 worked as internship, 3.6% participated in a scientific research, 0.9% participated in an extension project, 3.6% have 1 year of experience, 1.8% have 1 to 3 years of experience, 0.9% have more than 3 years of experience, 0.9 % are computer technicians, 0.9% took an online course and 0.9% had programming experience in high school.

4.3 Instruments

The questions of the questionnaire are described in sections 4.3.1 and 4.3.2.

4.3.1 Self-regulation Questions

1. During the programming course, did I monitor my performance and try to overcome any obstacles?
2. Did I motivate myself to participate in all individual and group programming activities, even when there was not much interest on my part? 3. Did I use as motivation the fact that programming is important for my course and my future profession? 4. Did I seek help from classmates or the teacher when I couldn't solve a programming problem? 5. Have I found ways to focus on programming even when there are sources of distraction? 6. Have I used time management strategies and managed to finish my programs? 7. Did I use the “divide and conquer” strategy by thinking about each part of the program in different modules? 8. Did I try to remain confident during programming, telling myself that I could do it? 9. When studying introduction to computing, did I look for different sources of information? 10. What study sources did you use? 11. Have I used sketches, diagrams or other types of drawings or sketches to organize my ideas about the logic of programming before coding? 12. Have I thought of different code alternatives for the same computational problem? 13. Did I review the lectures or look for supplementary material when I could not make a program of the practical class? 14. When studying Introduction to Computing, did I set goals for myself to direct my activities in each study period? 15. Did I adapt and match programming patterns when coding my programs? 16. Did I make an effort to participate in the practical classes?

4.3.2 Co-regulation and Shared Regulation Questions

1. With respect to computational solutions, have I tried to question the teacher and colleagues looking for evidence? 2. Did you use social media and other forms of technology to communicate with classmates? 3. What communication and collaboration technologies did you use during the course? 4. In group projects, did I try to motivate colleagues so that everyone contributed to the construction of the programs? 5. Did I contribute to a good working atmosphere during the joint programming, facing difficulties with good humor? 6. Have I valued colleagues' code parts and contributed to improvements? 7. Have I treated my colleagues with respect and used positive phrases such as "Very good! Keep it up! Thank you! You've helped us a lot now!"? 8. Have I tried to reconcile your goals, priorities and learning style with those of my colleagues? 9. Was the group work organized together, trying to reconcile the preferences of the members? 10. Was any time management strategy used for group projects, such as Kanban or Scrum? 11. Was any tool used to manage collaborative programming, such as Trello or GitHub? 12. Did the group use the “divide and conquer” strategy by thinking about each part of the program in different modules? 13. In group projects, was the commitment of everyone in the group to compliance with the rules and participation in programming activities monitored and action taken if necessary? 14. In group projects, were roles assigned to be played by students during the writing of the program, such as writer, consultant, editor and reviewer? 15. Was any joint programming strategy used, such as the Coding Dojo? 16. In group programming projects, was there reflection on the quality of interactions and group performance, and action taken when necessary? 17. Have group interactions positively influenced my personal performance?

4.3.3 Instrument for Questionnaire Analysis by Experts

The experts answered the following Yes/No questions: Are the questions simple, clear, easy to understand? Do the questions cover relevant aspects of self-regulation, co-regulation and shared regulation in programming? Are the questions parsimonious enough to ignore irrelevant aspects, but do they sufficiently cover self-regulation, co-regulation and shared regulation strategies?

4.3.4 Instruments for Data Analysis

For data analysis we utilized descriptive statistics. In addition, we used a technique proposed by Tastle and Wierman (2007), that makes it possible to identify for each proposed statement, by means of a score, the direction of the responses of all respondents for agreement or disagreement. Therefore, firstly, for each of the answer alternatives (options), a different weight (P) is determined, being, respectively, for totally disagree (TD), disagree (D), neutral (N), agree (A) and totally agree (TA), the values 1, 2, 3, 4 and 5. Then, in order to identify the score for each statement, the following formula applies: Score = ((nTD / ntotal) x 1)) + ((nD / ntotal) x 2)) + ((nN / ntotal) x 3)) + ((nA / ntotal) x 4)) + ((nTA / ntotal) x 5)). Therefore, the final score of each statement is obtained from the sum of the score of each of the five answer options (TD; D; N; A; TA), which is achieved by the percentage of responses (number of responses of the alternative divided by the total number of responses), multiplied by the corresponding P. For the interpretation of the results found in the score, it is considered that an affirmative has a "high" score when the value is greater than or equal to four, as it indicates evidence of partial or total agreement, while a "low" score, with a value less than four, represents disagreement with the proposed statement. The closer the score value to five, the greater the tendency of participants to fully agree with the statement, and, consequently, the closer the value is to one, the more likely it is that participants will totally disagree with the statement.

5 RESULTS

5.1 Validity and Reliability of the Questionnaire Applied to the Students

The first result encompasses the analysis of the questionnaire by specialists. The validation by observation of the questionnaire, carried out by the six experts, obtained a very favorable result. All six experts answered "Yes" to all three questions of the evaluative instrument posed to them, that was described in subsection 4.3.3.

The second result concerns the internal consistency of the questionnaire. The internal consistency appraises the reliability of summated scores derived from a Likert scale. Internal consistency refers to the extent to which there is compatibility and correlation among the responses to multiple items comprising the Likert scale.

The Cronbach's alpha statistical test was applied to the first part of the questionnaire, involving the self-regulation of students, to verify if the questions are interrelated. Also, the Cronbach's alpha statistical test was applied to the second part of the questionnaire, that covers the co-regulation and shared regulation of students, to find out if the questions are cohesive. The Cronbach's alpha coefficient interpretation is described in Table 1.

Table 1: Cronbach's alpha coefficient interpretation (Cronbach, 1951).

0.9 <= Alpha	Excellent
0.8<= Alpha < 0.9	Good
0.7<= Alpha < 0.8	Acceptable
0.6<= Alpha < 0.7	Questionable

The values for the Cronbach's Alpha coefficient for the first and second part of the questionnaire are in Table 2.

Table 2: Cronbach's alpha coefficient for the first and second part of the questionnaire.

Coefficient	Self-regulation	Co-regulation and Shared regulation
Cronbach alpha	0.795	0.881

According to Table 2, the Cronbach alpha coefficient value obtained for the questions about student self-regulation is 0.795. Therefore, following the interpretation in Table 1, it can be said that the questions involving self-regulation are correlated, attesting to their internal consistency.

Similarly, according to Table 2, the Cronbach alpha coefficient value obtained for the questions about co-regulation and shared regulation among students is 0.881. Thus, according to the interpretation in Table 1, it can be said that the questions concerning co-regulation and shared regulation are correlated, indicating that they are internally consistent.

5.2 Students Perceptions of Their Use of Regulatory Strategies

The descriptive Table 3 shows the percentage of responses to self-regulation questions, while the descriptive Table 4 shows the percentages of responses to co-regulation and shared regulation questions. In Tables 3 and 4, "QN" means "question number".

Given the distribution of the percentage of answers in Table 3 and Table 4, we can observe that, on most questions, the students' answers to the items "Neutral", "Disagree" and "Strongly Disagree" of the Likert scale are more than 30% of the answers. This result indicates that, according to the students' perception, a considerable part of the students does not use it frequently and, in some cases, do not master regulatory strategies when learning initial computer programming.

The scenario is worse for co-regulation and shared regulation than for self-regulation. When comparing the answers to the self-regulation questions (Table 3) with the co-regulation and shared regulation questions (Table4), we found that the students perceive that they use even less strategies of co-regulation and shared regulation.

Table 3: Percentage of responses for self-regulation questions.

QN	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
1	19%	53%	23%	4%	1%
2	20%	53%	16%	8%	3%
3	33%	39%	18%	7%	3%
4	24%	36%	23%	11%	6%
5	13%	47%	27%	10%	3%
6	30%	30%	26%	12%	2%
7	14%	39%	26%	12%	9%
8	17%	34%	27%	15%	7%
9	30%	44%	16%	6%	4%
11	9%	27%	24%	20%	20%
12	17%	37%	26%	17%	4%
13	23%	42%	19%	9%	7%
14	10%	49%	27%	4%	10%
15	19%	35%	29%	13%	4%
16	27%	36%	20%	11%	6%

Concerning cognitive strategies of self-regulation, only 53% of the students strongly agree and agree that they use the "divide and conquer" strategy by thinking about each part of the program in different modules. The significant 26% of the students are neutral, disagree, or strongly disagree that they look for different sources of information when studying introduction to computing. Students scarcely, just 36% of the students, strongly agree and agree that they use sketches, diagrams or other types of drawings or sketches to organize my ideas about the logic of programming before coding. 47% of the students are neutral, disagree, or strongly disagree that think of different code alternatives for the same computational problem. 35% of the students are neutral, disagree, or strongly disagree that they review the lectures or look for supplementary material when they could not make a program. The symbolic amount of 46% of the students are neutral, disagree, or strongly disagree that they adapt and match programming patterns when coding. These results are worthy of attention, due to the fact they provide evidence that a not inconsequential number of students do not apply cognitive strategies of self-regulation.

With regard to emotional strategies of self-regulation, the considerable amount of 27% of the students are neutral, disagree, or strongly disagree that they motivate themselves to participate in all individual and group programming activities, even when there was not much interest on their part. 28% of the students are neutral, disagree, or strongly disagree that they use as motivation the fact that programming is important for their course and their future profession. 49% of the students are neutral, disagree, or strongly disagree that they try to remain confident during programming, telling themselves that they could do it. These results indicate that a significant number of students do not utilize emotional strategies of self-regulation.

About behavioral strategies of self-regulation, 28% of the students are neutral, disagree, or strongly disagree that they monitor their performance and try to overcome any obstacles during the programming course. 40% of the students are neutral, disagree, or strongly disagree that they seek help from classmates or the teacher when they couldn't solve a programming problem. 40% of the students are neutral, disagree, or strongly disagree that they use time management strategies and manage to finish their programs. 41% of the students are neutral, disagree, or strongly disagree that they set goals for themselves to direct their activities in each study period, when studying introduction to computing. 37% of the students are neutral, disagree, or strongly disagree that they make an effort to participate in the practical programming classes. These results show that a not negligible number of students do not utilize a not negligible number of students do not utilize behavioral strategies of self-regulation.

In the matter of contextual strategies of self-regulation, 40% of the students are neutral, disagree, or strongly disagree that they find ways to focus on programming even when there are sources of distraction. This result indicates that a relevant number of students do not utilize contextual strategies of self-regulation. These results point out that an expressive number of students do not use contextual strategies of self-regulation.

Table 4: Percentage of responses for co-regulation and shared regulation questions.

QN	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
1	10%	30%	37%	16%	7%
2	44%	35%	10%	4%	7%
4	13%	27%	35%	11%	14%
5	13%	53%	26%	1%	7%
6	13%	50%	22%	2%	13%
7	25%	44%	19%	4%	8%
8	13%	38%	29%	12%	8%
9	19%	35%	26%	8%	12%
10	4%	9%	1%	19%	57%
11	4%	14%	15%	16%	51%
12	12%	33%	30%	12%	13%
13	13%	33%	32%	11%	11%
14	5%	16%	20%	16%	43%
15	3%	7%	19%	17%	54%
16	12%	26%	30%	11%	21%
17	17%	50%	24%	1%	8%

With respect to socio-cognitive strategies for co-regulation and shared regulation, only 40% of the students strongly agree and agree that they try to question the teacher and colleagues looking for evidence regarding computational solutions. 37% of the students are neutral, disagree, or strongly disagree that they value colleagues' code parts and contribute to improvements. Only 51% of the students strongly agree and agree that they try to reconcile your goals, priorities and learning style with those of their colleagues. Just 45% of the students strongly agree and agree that they use the "divide and conquer" strategy by thinking about each part of the program in different modules. These results indicate that a suggestive number of students do not utilize socio-cognitive strategies for co-regulation and shared regulation.

Concerning emotional strategies for co-regulation and shared regulation, 34% of the students are neutral, disagree, or strongly disagree that they contribute to a good working atmosphere during the joint programming, facing difficulties with good humor. 33% of the students are neutral, disagree, or strongly disagree that group interactions positively influence their personal performance. 21% of the students are neutral, disagree, or strongly disagree that they use social media and other forms of technology to communicate with classmates. Only 40% of the students strongly agree and agree that they try to motivate colleagues so that everyone contributes to the construction of the programs in group projects. 21% of the students are neutral, disagree, or strongly disagree that they treated my colleagues with respect and used positive phrases. These results reveal that an indicative number of students do not use emotional strategies for co-regulation and shared regulation.

Respecting behavioral strategies for co-regulation and shared regulation, only 10% of the students strongly agree and agree that they use joint programming strategies. Just 38% of the students strongly agree and agree that they reflect on the quality of interactions and group performance and take action when necessary during group projects. Hardly 13% of the students strongly agree and agree that they apply time management strategy in group projects. Merely 18% of the students strongly agree and agree that they use tools to manage collaborative programming. These results reveal that an evidential number of students do not apply behavioral strategies for co-regulation and shared regulation during introductory programming.

Regarding contextual strategies for co-regulation and shared regulation, 54% of the students are neutral, disagree, or strongly disagree that the group commitment agrees to the group rules and they monitor participation in programming activities and take action if necessary. 20% of the students are neutral, disagree, or strongly disagree that the group works together, trying to reconcile the preferences of the members. Only 10% of the students strongly agree and agree that, in group projects, the roles are assigned to be played by students during the writing of the program, such as writer, consultant, editor and reviewer. These results show that a significative number of students do not use contextual strategies for co-regulation and shared regulation when learning introductory programming.

Table 5: Scores of self-regulation questions.

Question Number	Self-regulation Score
1	3.84
2	3.81
3	3.95
4	3.63
5	3.59
6	3.76
7	3.38
8	3.42
9	3.95
11	2.85
12	3.49
13	3.68
14	3.45
15	3.55
16	3.7

Table 6: Scores of co-regulation and shared regulation questions.

Question Number	Co-regulation and Shared Regulation Score
1	3.2
2	4.08
4	3.13
5	3.65
6	3.49
7	3.75
8	3.37
9	3.41
10	1.87
11	2.08
12	3.18
13	3.26
14	2.23
15	1.88
16	3.01
17	3.69

Table 5 exhibits the score of responses to self-regulation questions and Table 6 presents the scores of responses to co-regulation and shared regulation questions. The scores were calculated according to the formula described in the section 4.3.

With reference to self-regulation strategies, no score in Table 5 is greater than 4 nor equal to 4, indicating that students judged that they do not use self-regulation strategies during introductory programming. A similar result was found when we analyzed the scores of the responses on the strategies of co-regulation and shared regulation. No score in Table 6 is greater than 4 nor equal to 4, revealing that students perceived that they do not apply co-regulation and self-regulation strategies when learning introductory programming.

Table 7: Global scores.

Score of All Self-Regulation Questions	3.6
Score of All Co-regulation and Shared Regulation Questions	3.08

Table 7 shows the global score for self-regulation questions and the global score for co-regulation and shared regulation questions. The global score was calculated as the mean of the scores. Table 7 displays a smaller global score for co-regulation and shared regulation questions, unveiling that the students perceive that they are even worse in utilizing co-regulation and self-regulation strategies during introductory programming.

6 CONCLUSIONS

The present study highlights the importance that regulation strategies are effective ways to improve the students' learning experience. Self-regulated learning is a relevant topic in introductory computer programming, which involves the regulation of student motivation, engagement, cognition and metacognition. The same way as shared regulated learning, which is understood as the social regulation of learning and an important topic when students learn programming in groups.

Therefore, programming teaching and learning approaches do not prioritize skills aligned with self-regulation, co-regulation and shared regulation. Students trying to learn to program do not always receive explicit training or support to develop the regulatory skills necessary for programming.

The main goal of the present study was to explore the students' perspective of their use of regulation strategies during programming. An exploratory study involving 198 students, found evidence for the fact that programming novices use regulation strategies to a limited extent, calling attention to a demand for the development and application of teaching approaches to promote self-regulation, co-regulation and shared programming in introductory programming courses.

Understanding students' perspectives on their utilization of regulation strategies during programming is an important addition to studies in the computer science education field, because results can broaden our understanding of regulation learning approach. The results of this work will help when it comes to designing future teaching and learning approaches.

ACKNOWLEDGEMENTS

This work was partially funded by the Federal University of Goiás.

REFERENCES

Bergin, S., Reilly, R., & Traynor, D. (2005). Examining the role of self-regulated learning on introductory programming performance. In *Proceedings of the first international workshop on Computing education research*, 81–86. ACM.

Burridge, J., & Fekete, A. (2022). Teaching programming for first-year data science. In *ITiCSE'22, 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 1.*

Chang, B. (2019). Reflection in learning. *Online Learning*, 23(1), 95–110. http://doi.org/10.24059/olj.v23i1.1447

Cheng, Y., Shen, P., Hung, M, Tsai, C., Lin, C., & · Hsu, L. C. (2021) Applying online content-based knowledge awareness and team learning to develop students' programming skills, reduce their anxiety, and regulate cognitive load in a cloud classroom. *Universal Access in the Information Society*, 21, 557–572. https://doi.org/10.1007/s10209-020-00789-6

Chorfi, A., Hedjazi, D., Aouag, S. & Boubiche, D. (2020). Problem-based collaborative learning groupware to improve computer programming skills. *Behaviour & Information Technology*, 139-158. https://doi.org/10.1080/0144929X.2020.1795263

Coto, M., Mora, S., Grass, B., & Murillo-Morera, J. (2022). Emotions and programming learning: systematic mapping. *Computer Science Education*, 32(1), 30–65. http://doi.org/10.1080/08993408.2021.1920816

Crisp, G., Taggart, A., & Nora, A. (2015). Undergraduate Latina/o Students: A Systematic Review of Research Identifying Factors Contributing to Academic Success Outcomes. *Review of Educational Research*, 85(2), 249–274.

Cronbach, L.J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika*, 16 (3), 297–334. http://doi.org/10.1007/BF02310555

Cukierman, U. R., & Palmieri, J. M. (2014). Soft skills in engineering education: A practical experience in an undergraduate course, in Interactive collaborative learning. In *2014 International Conference on Interactive Collaborative Learning*. 237–242.

Darabi, K., Gholamzadeh Jofreh, M., & Shahbazi, M. (2022). The role of self-regulation training in self-efficacy and academic motivation of male tenth graders in Ahvaz, Iran. *International Journal of School Health*, 9(2), 106-112. http://doi.org/10.30476/intjsh.2022.94543.1210

Daradoumis, T., Marquès Puig, J.M., & Arguedas, M. (2021). A distributed systems laboratory that helps students accomplish their assignments through self-regulation of behavior. *Educational technology research and development,* 69**,** 1077–1099. https://doi.org/10.1007/s11423-021-09975-6

Falkner, K., Vivian, R., & a Falkner, N. (2014). Identifying computer science self-regulated learning strategies. In *2014 Conference on Innovation & technology in computer science education* (*ITiCSE '14*). Association for Computing Machinery, New York, NY, USA, 291–296.

Falkner, K., Szabo, C., Vivian, R., & Falkner. N. (2015). Evolution of software development strategies. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2, 243–252. IEEE, 2015.

Groeneveld, W., Vennekens, J., & Aerts, K. (2019). Software Engineering Education Beyond the Technical: A Systematic Literature Review. In *Proceedings of the 47th SEFI Conference 2019.*

Habib, A., Abdullatif, M., & Alzayani, N. J. (2021). Use of Rubric and Assessment to Encourage Self-Regulated Learning. In *IEEE Integrated STEM Education Conference (ISEC)*, 195-200.

Hadwin, A., Järvelä, S., & Miller, M. (2018). Self-regulation, co-regulation, and shared regulation in collaborative learning environments. In D. Schunk, & J. Greene, (Eds.). *Handbook of self-regulation of learning and performance* (2nd ed.). New York, NY: Routledge.

Hauswirth, M., & Adamoli, A. (2017). Metacognitive calibration when learning to program. In *17th Koli Calling International Conference on Computing Education Research*, Koli Calling '17, 50–59, New York, NY, USA. ACM.

Hwang, G., Wang, S., & Lai, C. (2021). Effects of a social regulation-based online learning framework on students' learning achievements and behaviors in mathematics. *Computers and Education*, 160, 1–19.

Järvenoja, H., Volet, S., & Järvelä, S. (2013). Regulation of emotions in socially challenging learning situations: An instrument to measure the adaptive and social nature of the regulation process. *Educational Psychology*, *33*(1), 31–58.

Kaner, C., & S. Padmanabhan, S. (2007). Practice and Transfer of Learning in the Teaching of Software Testing. In *CSEET'07, 20th Conference on Software Engineering Education & Training*, Dublin, Ireland, 157-166. http://doi.org/10.1109/CSEET.2007.38.

Keller M. (2017). Motivation, learning, and technology: applying the ARCS-V motivation model. Participatory Educational Research, 3(2), 1–13. http://dx.doi.org/10.17275/per.16.06.3

Ko, A. J., LaToza, T. D., Hull, S., Ko, E. A., Kwok, W., Quicho-cho, J., Akkaraju, H., & Pandit, R. (2019). Teaching explicit programming strategies to adolescents. In *P 50th ACM Technical Symposium on Computer Science Education*, 469–475. ACM.

Loksa, D. (2020). Explicitly training metacognition and self-regulation for computer programming. A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy University of Washington.

Loksa, D., & Ko, A. J. (2016). The role of self-regulation in programming problem solving process and success. In *ICER'16, 12th ACM Conference on International Computing Education Research.*

Likert, R. A. (1932). Technique for the measurement of attitudes. *Archives in Psychology, 140,* 1–55.

McDowell, C., Werner, L., Bullock, H., & Fernald., J. (2002). The effects of pair programming on performance in an introductory programming course. In *33rd SIGCSE Technical Symposium on Computer Science Education.* ACM SIGCSE Bulletin 34(1), 38–42.

Malmi, L., Sheard, J, Kinnunen, P., Simon, & Sinclair, J. (2019). Computing education theories: what are they and how are they used?. In *2019 ACM Conference on International Computing Education Research,* (Toronto ON, Canada) (ICER '19). ACM, New York, NY, USA, 187–197.

Pedrosa, D., Cravino, J. Morgado,L., & Barreira, C. (2019). Co-regulated learning in computer programming: students co-reflection about learning strategies adopted during an assignment. Avalable at https://www.scielo.br/j/prod/a/fZqTXHD3BVhchGH9553GSVP/?lang=en

Pereira, F. T. S. S., Rosa, N. S., Silva, D. C., Pereira, C. P., & Bittencourt, R. A. A. (2021). Remote CS0 workshop based on peer learning: motivation, engagement and self-regulation of novice programmers. In *12th IEEE Global Engineering Education Conference*, Viena, Áustria.

Pintrich P. R. (1993). Reliability and predictive validity of the motivated strategies for learning questionnaire (MSLQ). *Educational and Ppsychological Measurement,* 53(3), 801–813.

Pintrich P. R. (2000). The Role of Goal Orientation in Self-Regulated Learning. The University of Michigan, Ann Arbor, Michigan.

Prather, J., Becker, B. A., Craig, M., Denny, P., Loksa, D., & Margulieux. L. (2020). What Do We Think We Think We Are Doing? Metacognition and Self-Regulation in Programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (*ICER '20*). Association for Computing Machinery, New York, NY, USA, 2–13.

Prous, G., Salvanés, R., & Ortells, C. (2009). Validation of questionnaires. *Reumatología Clínica, 5*(4), 171–177. https://doi:10.1016/j.reuma.2008.09.007

Schoeffel, P. (2019) A method to predict at-risk students in introductory computing courses based on motivation. Thesis submitted to the Graduate Program in Computer Science at the Federal University of Santa Catarina to obtain the title of Doctor in Computer Science.

Szabo, C., Falkner, N., Petersen, A., Bort, H., Cunningham, K., Donaldson, P., Hellas, A., Robinson, J., & Sheard, J. (2019). Review and Use of Learning Theories within Computer Science Education Re- search: Primer for Researchers and Practitioners. In *Proc. of the WG Reports on Innovation and Technology in Comp Sci Education,* (Aberdeen, Scotland Uk) (ITiCSE- WGR '19). ACM, NY, USA, 89–109.

Soares, L. (2021). Fostering programming student's regulation of learning using a computer-based learning environment. In *SIIE'21, 23rd International Symposium on Computers in Education.*

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer science education*, 13(2), 137–172.

Tastle, W. J., & Wierman, M. J. (2007). Consensus and dissention: a measure of ordinal dispersion. *International Journal of Approximate Reasoning, 45*, 531–545. https://doi.org/10.1016/j.ijar.2006.06.024

Tsai, C.W. (2015). Applying web-based co-regulated learning to develop students' learning and involvement in a blended computing course. *Interactive Learning Environments,* 23(3), 344–355.

Wang, C.H., Shannon, D., Ross, M. (2013). Students' characteristics, self-regulated learning, technology self-efficacy, and course outcomes in online learning. *Distance Education*, 34(3), 302– 323.

Wrenn, J., & Krishnamurthi, S. (2019). Executable examples for programming problem comprehension. In *2019 ACM Conference on International Computing Education Research*, 131–139. ACM.

Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D. Kwik, H., Tan, A. H., Hwa, L., Li, M. & Ko, A. J. (2019). A theory of instruction for introductory programming skills. *Computer Science Education*, 29(2), 205-253. http:// doi.org/10.1080/08993408.2019.1565235.