Contents lists available at ScienceDirect

Computers in Human Behavior

journal homepage: http://www.elsevier.com/locate/comphumbeh

Full length article

Applying computational analysis of novice learners' computer programming patterns to reveal self-regulated learning, computational thinking, and learning performance

Donggil Song[a], Hyeonmi Hong[b,*], Eun Young Oh[c]

[a] Instructional Systems Design and Technology, Sam Houston State University, Huntsville, TX, USA
[b] Elementary Education Research Institute, Teachers College, Jeju National University, Jeju, South Korea
[c] Center for Languages and Intercultural Communication, Rice University, Houston, TX, USA

ARTICLE INFO

Keywords:
Self-regulated learning
Computational thinking skills
Computer education
Computational analysis
Computer programming

ABSTRACT

Educational research on predicting learners' computer programming performance has yielded practical implications that guide task designs in computer education. There have been attempts to investigate learners' computer programming patterns using high-frequency and automated data collection. This approach can be considered as process-based analysis as opposed to outcome-based analysis (i.e., the use of test or exam scores). In this process-based approach to investigate learners' computer programming process, we included two critical constructs in our research, self-regulated learning and computational thinking skills. We aimed to identify learners' computer programming patterns in the context that novice students learn a computer programming language, Python, in an online coding environment. We examined the relationships between the learners' coding patterns, self-regulated learning, and computational thinking skills. Initially, we adopted a traditional approach with the aggregate data of learners' computer programming behaviors. We then utilized a computational analytics approach to learner performance, self-regulated learning, and computational thinking skills, with ever-changing computer programming patterns. In our initial approach, the indicators of aggregate computer programming data were not associated with learners' learning performance and computational thinking skills. In the computational analysis approach, many indicators revealed significant differences between the identified patterns regarding computational thinking skills and self-regulated learning. Recommendations about the use of programming log data analysis methods and future scaffolding for computer programming learners are addressed.

Public attention has been drawn to computer programming education since the 1980s (Soloway et al., 1982). These days, since there are many professional areas that require programming knowledge and skills, computer programming or coding is considered as a skill needed by different types of professionals, not just computer programmers, but also engineers, data scientists, economists, and education analysts. Still, computer programming education is arguably lagging behind the societal demand for more computer programmers (Giannakos et al., 2017).

Educational research on predicting learners' computer programming performance has yielded implications that guide the identification of at-risk students and learner scaffolding in computer education (e.g., Hall et al., 2006; Yukselturk & Bulut, 2005). Many of these efforts focus on learning outcomes using quizzes, tests, and exams as a knowledge indicator. When we focus on learners' outcome-based performance, it might be challenging to understand their authentic learning process; that is, changes in computer programming learning over time. Thus, monitoring learners' programming behaviors to understand their genuine learning process is one of the significant research topics in computer science education (Azcona et al., 2019; Fields et al., 2016).

One of the difficulties in this research area is the challenging data collection process. Each individual learner needs to have their own machine equipped with a coding environment/software, usually in a computer lab or at home. Researchers should be able to retrieve learners' programming behaviors from their devices. Online code repositories, such as GitHub (https://github.com/), have made this data collection process more comfortable and manageable. On these online

* Corresponding author.
E-mail addresses: donggil.song@gmail.com (D. Song), hongspiderweb@gmail.com (H. Hong), oh@rice.edu (E.Y. Oh).

https://doi.org/10.1016/j.chb.2021.106746
Received 14 August 2020; Received in revised form 17 October 2020; Accepted 11 February 2021
Available online 15 February 2021

repositories, learners' codes are stored whenever they upload their codes while they are programming. Still, because it all depends on the learner's upload decision, the time points of data collection could be different for each student and each task. An alternative way to collect learners' coding data and accommodate the learners' programming process, online coding environments have been used (e.g., Price et al., 2019). Although online environments might not fully support all functions and features of a programming language, it would be enough to support novice programmers to learn the basics of coding.

Another crucial consideration in computer programming education research is data analysis methods. Traditionally, instructional methods and learners' pre and posttest scores have been frequently investigated in this field (Bishop-Clark et al., 2006). Along with test-based outcome measures, there have been attempts to investigate learners' coding styles and patterns (Watson et al., 2013, 2014). Specifically, due to high-frequency and automated data collection, novel analysis approaches and techniques have been offered new insights into the computer programming learning process and individual learner's patterns and behaviors (e.g., Blikstein et al., 2014; Fields et al., 2016).

In this study, focusing on a learning process-based analysis approach, we aimed to analyze learners' computer programming patterns. The research context includes how novice students learn a computer programming language, Python, in an online coding environment. We investigated the relationships between the learners' coding patterns and their learning performance. In addition, we subsequently aim to contribute to the field by adding learners' self-regulated learning aspects and computational thinking skills into the current findings in the literature.

1. Literature review

Computing education researchers have focused on learners' computer coding processes itself along with their performance (Blikstein et al., 2014; Watson et al., 2014). This approach can be considered as process-based analysis as opposed to outcome-based analysis. Mostly, the process-based analysis includes computational techniques to understand learners' learning status and monitor learners' behaviors and performance in the field of computer programming education. This trend also provides for the exploration of data-driven approaches, where the log data of learners' programming behaviors are available.

The primary potential of process-based analysis is that it would be able to guide and scaffold learners. This is because their ever-changing performance and behaviors would be monitored and analyzed without having to administer lengthy tests or exams. That is, the learners' computer programming behavior can be an indicator of their learning performance and, at the same time, their performance as an artifact. Thus, researchers have become increasingly interested in analyzing learners' programming log data as they work on computer programming tasks (Fields et al., 2016; Liu et al., 2017).

1.1. The programming process

To understand learners' computer programming process, a series of research focuses on learners' coding mistakes and their time to fix their errors (e.g., Altadmri & Brown, 2015, pp. 522–527). Learners' errors can be divided into syntactic (i.e., programming language's grammatical rules) and semantic (i.e., program logic or algorithms) errors. Focusing on learners' programming process data, Carter et al. (2015) suggested the Normalized Programming State Model (NPSM), which predicts learners' performance. NPSM includes syntactic and semantic correctness of learners' computer programs at any given point in time. Because semantic correctness is challenging to determine precisely, the researchers used unknown status; thus, a student's programming solution at a particular time point can be assigned to one of the four states: (1) syntactically correct and semantically incorrect, (2) syntactically incorrect and semantically incorrect, (3) syntactically correct and semantically unknown, and (4) syntactically incorrect and semantically unknown. The researchers collected 140 undergraduate students' programming log data and their course grades. As a predictor, NPSM was compared with other metrics, such as Error Quotient (Jadud, 2006) and Watwin Score (Watson et al., 2013), to estimate students' performance. Overall, the linear regression models with the selected time points by the researchers revealed that NPSM showed a decent performance when predicting students' individual assignment grades and final grades (Carter et al., 2015). As such, learners' programming process can be utilized as a predictor of their performance.

While this line of research showed a moderate performance when predicting learning outcomes, it has a limitation when selecting a specific data point or time point to extract a measure of dynamic and ever-changing computer programming behaviors due to the limited number of data snapshots. There have been studies to overcome this limitation. Blikstein et al. (2014) compared different approaches of analysis density – from the regression of aggregate data and further disaggregation of the data to identify learners' computer programming patterns when predicting learning performance. The researchers collected undergraduate students' code snapshots of seven computer programming assignments and calculated the size and frequency of code updates. In their regression analysis, the averaged magnitude of the code updates did not show a significant relationship with course performance. When they changed their approach from the average size to the clusters of different time points (four and twelve) per assignment, the results showed a small effect size when predicting course performance. These approaches focused solely on the size dimension of students' codes. Then, the researchers added a frequency aspect into their dataset, which is a type of curve as opposed to aggregate data. Using standardization and normalization processes, they identified six different patterns of code updates, from the lowest code change to the greatest change. When they compared these six groups with the students' course performance, there was a significant relationship; specifically, the students who changed their programming patterns more achieved higher grades in their course (Blikstein et al., 2014). This analysis approach with fine-grained data shows a clearer view of the learners' coding process than the analysis with aggregate data does. The approach has shed light on identifying significant implications in computing education.

1.2. Towards scaffolding

One of the central reasons why computing education researchers have taken great interest in analyzing learners' coding patterns and their relationships with learning performance is to find an optimal way to provide scaffolds for the learner. Computing educators desire to design an appropriate intervention to meet specific instructional goals when teaching computer programming.

An initial step for designing scaffolds is to identify the difficulty of programming tasks. Due to the vast amount of tasks and different types, most importantly, the relativity of difficulty depending on each learner, researchers have investigated how to detect the difficulty of tasks automatically. As an exploratory attempt, Ihantola et al. (2014) examined learner aspects (e.g., prior coding experience) and different metrics of code snapshots (e.g., time, lines of code). It was found that the time spent on an assignment and the size of coding (e.g., events, keystrokes) were significant factors when determining the difficulty of a computer coding task. Still, this area needs further investigation.

There are practical approaches to support learners by giving prompts or hints when they are working on computer programming. Price et al. (2019) examined the quality of data-driven algorithms that were designed to provide code hints for learners while programming. Automatically or when a learner requests a hint, these algorithms analyze the learner's current code state and provide a useful example code or recommendation for the learner. The researchers invented an automatic evaluation framework (called QualityScore) and confirmed its reliability using human expert evaluation results. Their framework could differentiate the quality of each code hinting algorithm.

While these attempts have achieved modest success in supporting learners' performance, we still have challenges contributing to their actual learning process. First, the narrow focus on computer programming performance (e.g., task's difficulty, a certain code status at a time point) might overlook the crucial aspects of learning, such as learners' motivation and metacognition. Second, the mechanical coding guides and hints might pretermit the learners' ultimate learning goals, such as enhancing algorithmic and logical thinking skills. This is the main reason for us to include two critical constructs in our research, self-regulated learning and computational thinking skills.

1.3. Psychological traits

There has been an argument that learners' psychological or background traits are not directly associated with their computer programming behaviors, and psychological indicators might not reflect the changes in computer programming and coding patterns (Watson et al., 2014). We acknowledge the limitations of learners' psychological indicators in relation to their computer programming process. However, it can be argued that we should be revisiting the ultimate goals of computer education, which is not merely the improvement of computer coding skills (Schulte, 2013). One of the essential goals is that computer coding can be considered as a learning method to improve computational thinking skills (Voogt et al., 2015). Most importantly, as computer coding education extends to non-major students in online learning environments, many of the coding lessons have adopted self-paced learning modes. Thus, we argue that learners' self-regulated learning should also be included in the discussion of computer programming pattern analysis.

1.4. Computational thinking skills

Computational thinking can be defined as it "involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (Wing, 2006, p. 33). Lu and Fletcher (2009) described computational thinking as a conceptual way to solve real problems through thinking systematically, correctly, and efficiently about information and tasks. Other definitions have their roots in algorithmic thinking, abstraction, decomposition, generalization and pattern recognition, data representation (Burbaitė et al., 2018), and even creative and critical thinking (Korkmaz et al., 2017). Following these conventional definitions of the term, we define computational thinking as thought processes that contain knowledge, skills, and attitudes that are based on a computer system, which is necessary to thoroughly understand and fully utilize a computer system to solve problems.

There have been sizeable attempts to improve learners' computational thinking skills (e.g., Atmatzidou & Demetriadis, 2016; Gardeli & Vosinakis, 2017). In addition, researchers investigated predictors of computational thinking skills. Durak and Saritepeci (2018) investigated secondary school students' computational thinking skills and reported that the success indicators of different classes (e.g., math, science, information technologies) predict the participants' computational thinking skills. Still, the relationships between computational thinking skills and the computer programming process are under-explored.

1.5. Self-regulated learning

A similar under-exploration has been found in the field of self-regulated learning research. Self-regulation generally refers to "self-generated thoughts, feelings, and actions that are planned and cyclically adapted to the attainment of personal goals" (Zimmerman, 2000, p. 14). Although the concept of monitoring computer programming behavior has a long history (Altadmri & Brown, 2015, pp. 522–527), few studies addressed self-regulated learning in relation to computer programming behavior. Instead, there have been attempts to increase students' computer programming performance by scaffolding their self-regulated learning (e.g., Alharbi et al., 2012; Brito et al., 2011, pp. 1–9; Kumar et al., 2005; Pedrosa et al., 2016). In addition, research on relationships between self-regulated learning and learning performance has been frequently conducted. Chen (2002) examined the relationships between MSLQ (Motivated Strategies for Learning Questionnaire; Pintrich & DeGroot, 1990) scores and 197 undergraduate students' learning performance on computer concepts. The effort-regulation component of MSLQ has a significant relationship with learning performance. Yukselturk and Bulut (2005) measured undergraduate 38 students' self-regulated learning using MSLQ and computer programming achievements. It was found that among the MSLQ components, self-efficacy and self-regulation explained the significant amount of the variances of computer programming performance (28.9% and 27.7%, respectively). Cigdem (2015) investigated 267 military vocational college students' self-regulated learning skills and computer programming achievements. The self-efficacy component was correlated with programming achievement ($r = 0.181, p < .01$). Similar results were found in other studies (Bergin et al., 2005; Echeverry et al., 2018). Thus, it seems that there is an established relationship between self-regulated learning and learning performance in computer education.

Because self-regulated learners "plan, set goals, organize, self-monitor, and self-evaluate at various points during the process of acquisition" (Zimmerman, 1990, p. 4), learners' self-regulated learning might be revealed in their computer programing process. A few studies were conducted to examine learners' programming codes and reveal their relationship with self-regulated learning. Castellanos et al. (2017) collected 205 undergraduate students' computer programming codes and extracted source code metrics of length (e.g., the count of lines) and complexity (e.g., amount of loops, amount of if-else clauses), and Halstead metrics (e.g., effort, time, difficulty). The results show that the length indicator is positively correlated with programming performance and self-regulated learning measured through MSLQ. However, the changes in learners' codes and coding patterns were not examined in these studies.

1.6. This study

The significant learning-related constructs (i.e., self-regulated learning and computational thinking skills) were under-investigated in terms of the relationship with the learners' computer programming process. In this study, we analyzed a continuous stream of learners' codes to identify their learning patterns, which could possibly be related to their self-regulated learning, computational thinking, and learning performance. Our primary research question is, "Is the learners' computer programming process related to self-regulated learning, computational thinking skills, and learning performance?" Initially, we were interested in traditional approaches to learning performance and learning-related constructs (i.e., self-regulated learning and computational thinking skills) with the aggregate data of learners' computer programming. This approach intends to identify a predictive model of learner performance. Then, we focused on computational analytics approaches (Song, 2018) to the performance and those constructs with ever-changing learner behaviors, patterns, and meaningful trajectories in students' computer programming log data.

2. Methods

2.1. Participants and the context

One hundred thirty-eight students enrolled in a course, Creative Extracurricular Activities and Software, which had five sections for senior undergraduate students from Teachers College (i.e., preservice teachers) at a mid-sized public university in an urban area in South Korea. The course is a face-to-face course, which meets one time (a 3-h

Table 1
The correlation table of computer programming indicators and learner factors.

	Computer Programming				Performance	Computational Thinking Skills						MSLQ					
	Chosen-Task	Trial-Run	Successful Task	Coding Time	Final Grade	Total	1. Creativity	2. Algorithmic Thinking	3. Cooperativity	4. Critical Thinking	5. Problem-solving	Total	1. Self-efficacy	2. Intrinsic Value	3. Test-anxiety Free	4. Cognitive Strategy	5. Self-regulation
Chosen-Task	1.00																
Trial-Run	[b].53	1.00															
Successful Task	[b].82	[b].46	1.00														
Coding Time	[b].50	[b].66	[b].40	1.00													
Final Grade	-.15	-.07	-.09	-.10	1.00												
Computational Thinking Total	.07	-.03	-.04	-.15	-.14	1.00											
1. Creativity	.12	.09	.02	-.04	-.11	[b].77	1.00										
2. Algorithmic Thinking	.08	-.09	-.04	-.19	-.02	[b].80	[b].51	1.00									
3. Cooperativity	-.04	-.05	-.14	.01	-.12	[b].57	[b].36	[b].33	1.00								
4. Critical Thinking	.04	.00	.01	-.11	-.08	[b].73	[b].71	[b].57	[a].20	1.00							
5. Problem-solving	.02	-.03	.02	-.11	-.02	[b].29	-.09	.05	-.04	-.11	1.00						
MSLQ Total	[a].21	[a].24	.14	.13	-.12	[b].55	[b].65	[b].42	[b].30	[b].61	-.18	1.00					
1. Self-efficacy	[a].24	.15	.15	.05	.12	[b].55	[b].59	[b].46	.18	[b].56	-.04	[b].82	1.00				
2. Intrinsic Value	.14	.17	.06	.14	.16	[b].54	[b].56	[b].33	[b].34	[b].47	.03	[b].82	[b].74	1.00			
3. Test-anxiety Free	.10	.18	.08	.19	.07	-.19[a]	.05	-.12	-.18	.08	[b]-.40	[a].23	.01	.01	1.00		
4. Cognitive Strategy	.16	[a].19	.10	.10	.00	[b].46	[b].54	[b].34	[b].32	[b].45	-.16	[b].83	[b].49	[b].55	.09	1.00	
5. Self-regulation	.06	.19	.09	.04	.09	[b].38	[b].43	[b].33	[a].24	[b].47	[a]-.23	[b].71	[b].40	[b].37	.09	[b].64	1.00

[a] Correlation is significant at the 0.05 level (2-tailed).
[b] Correlation is significant at the 0.01 level (2-tailed).

course) a week. Each student was asked to solve programming problems during a 15-week semester. Excluding the individuals who did not participate in the measurement process of self-regulated learning and computational thinking skills, 105 students who completed all computer programming tasks and surveys were included in the data analysis.

The course grades include students' class participation and different types of projects, such as computer coding-related extracurricular activity design and group discussion projects. The computer coding tasks were considered as class participation, and regardless of the successful completion of tasks, students received class participation scores when they accessed the online computer programming environment with their log-in credentials. All 105 subjects in this study participated in all computer programming tasks. Therefore, the course grades do not directly reveal participants' computer programming knowledge, skills, or task performance, but mainly represent their course achievements in their different types of projects, which were still related to computer/software use and computer programming for their future students.

2.2. Data collection

We focus exclusively on differences between learners' successive code compilation attempts in the online environment. We instrumented the online Python coding environment to capture snapshots of students' codes when they run (i.e., trial-run) their programs. This environment collects learners' codes in real-time when they compile and run their codes. The data consisted of programming tasks completed by 105 students, which include 23,099 times of trial-run. Participants completed 53 computer programming tasks during a 15-week period. Three tasks out of 53 were not used due to the nature of the tasks; for example, the first task included the overview of this programming project, and another task was a series of quick reviews; thus, 50 tasks were included in the data analysis.

To measure participants' self-regulated learning, MSLQ (Pintrich & DeGroot, 1990) was used. This instrument has been widely adopted to measure the levels of self-regulated learning in the computing education research field (e.g., Chen, 2002; Sun & Hsu, 2019; Yukselturk & Bulut, 2005). MSLQ consists of 44 items designed to measure self-efficacy, intrinsic value, test anxiety, cognitive strategy use, and self-regulation

Table 2
Anova results of clustered groups (N = 105).

Clustering Factors	Measures	Clustered Groups				*F*	Partial η^2	Post Hoc
		a	b	c	d			
Chosen-Task-3	N	57	27	21	–			
	MSLQ – Self-efficacy	4.69 (1.10)	4.17 (.93)	4.12 (.89)	–	3.643*	.067	Not found
Chosen-Task-4	N	6	51	27	21			
	MSLQ – Total	5.40 (.75)	4.83 (.60)	4.63 (.56)	4.60 (.66)	3.252*	.088	a > c a > d
	MSLQ – Self-efficacy	5.52 (.70)	4.59 (1.10)	4.17 (.93)	4.12 (.89)	4.056*	.108	a > c a > d
Trial-Run-3	N	10	19	76	–			
	MSLQ – Total	5.26 (.65)	4.89 (.58)	4.67 (.61)	–	4.569*	.082	a > c
	MSLQ – Intrinsic value	5.17 (1.03)	5.15 (.81)	4.71 (.90)	–	2.613*	.049	Not found
	MSLQ – Cognitive strategy	5.72 (.64)	5.17 (.61)	5.05 (.72)	–	4.102*	.074	a > c
	MSLQ – Self-regulation	5.15 (.72)	4.47 (.74)	4.52 (.68)	–	3.782*	.069	a > b a > c
Time-3	N	18	79	8	–			
	CT – Total	4.50 (.54)	4.54 (.65)	3.88 (.66)	–	3.965*	.072	b > c
	CT – Creativity	5.55 (.73)	5.37 (.69)	4.30 (.76)	–	9.699**	.160	a > c b > c
	CT – Algorithmic thinking	4.21 (1.01)	4.43 (1.15)	3.21 (1.14)	–	4.288*	.078	b > c
	CT – Critical thinking	4.98 (.99)	4.95 (.99)	3.95 (.86)	–	3.866*	.070	a > c b > c
	MSLQ – Total	5.10 (.66)	4.75 (.58)	4.14 (.64)	–	7.416*	.127	a > c b > c
	MSLQ – Self-efficacy	4.80 (1.04)	4.47 (1.00)	3.39 (.92)	–	5.575*	.099	a > c b > c
	MSLQ – Cognitive strategy	5.44 (.59)	5.14 (.69)	4.39 (.81)	–	6.486*	.113	a > c b > c
	MSLQ – Self-regulation	5.01 (.80)	4.55 (.64)	3.82 (.58)	–	9.010**	.150	a > b a > c b > c
Time-4	N	19	8	72	6			
	CT – Creativity	5.48 (.73)	5.33 (.86)	5.36 (.70)	4.34 (.88)	4.077*	.108	a > c a > d
	MSLQ – Total	5.01 (.66)	4.89 (.86)	4.74 (.56)	4.14 (.70)	3.203*	.087	a > d
	MSLQ – Cognitive strategy	5.40 (.57)	5.18 (.93)	5.12 (.69)	4.41 (.82)	3.066*	.083	a > d
	MSLQ – Self-regulation	4.84 (.79)	4.69 (1.05)	4.54 (.63)	3.94 (.60)	2.711*	.075	Not found
Success-3	N	35	43	27	–			
	Grades	89.49 (2.93)	89.37 (3.14)	87.48 (4.13)	–	3.366*	.062	Not found

*$p < .05$.
**$p < .001$.

on a 7-point scale, ranging from 1 (not at all true of me) to 7 (very true of me). In addition, to measure participants' computational thinking skills, we adopted CTS (Computational Thinking Scales) (Korkmaz et al., 2017), which has been used in recent computer education research (e.g., Doleck et al., 2017; Durak & Saritepeci, 2018). Finally, to measure the learners' course performance, their final grades were used.

2.3. Data analysis

We conducted a series of analyses adopting a traditional approach. The learners' coding pattern indicators are the number of chosen tasks, overall code-run trials, average code-run trials per task, the number of successful tasks, and time spent per task. Although the participation in the coding tasks (i.e., logging in each task) was the course requirement, trial-run attempts or successful completion of tasks were not graded. Thus, the participants were able to focus on a specific task on their own or skip some tasks without any restriction. This is the reason for having the number of chosen tasks as their coding pattern indicator. For the same reason, overall code-run trials or time spent per task varied depending on students' coding behaviors, styles, and patterns. We analyzed the relationships between these coding pattern indicators and learner constructs (i.e., self-regulated learning, computational thinking, and learning performance). Using computational analysis, we demonstrate learners' programming patterns. Instead of combining or averaging indicators, we utilized computational techniques to identify the patterns of learners' coding behaviors and find relationships between the patterns and learner constructs. In this approach, data can represent each learner's trial-run attempt at different points in time as they progress towards the goal of each programming task.

3. Results

3.1. Initial approach

In the initial analysis, we included the number of chosen tasks (Chosen-Task), overall code-run trials (Trial-Run), the number of successful tasks (Successful Task), and time spent for tasks (Coding Time). We analyzed the relationships between these coding pattern indicators and learner constructs (i.e., self-regulated learning, computational thinking, and learning performance).

3.2. Correlation

We calculated correlation coefficients of all relevant variables (see Table 1). All four indicators of computer coding were not associated with learners' grades or computational thinking skills. Two indicators, Chosen-Task ($r = 0.21$) and Trial-Run ($r = 0.24$) were correlated with MSLQ (both $ps < .05$). Specifically, Chosen-Task was positively correlated with self-efficacy ($r = 0.24$, $p < .05$), and Trial-Run was positively related with cognitive strategy ($r = 0.19$, $p < .05$).

3.3. Regression analysis

We conducted a series of regression analyses to identify the relationships between computer programming indicators and learner factors. Similar to the correlation results, when learners' final grades and computational thinking levels were the outcome variables, the prediction power of four coding indicators were not statistically meaningful, $F(4,100) = 0.72$, $p = .58$, $R^2 = 0.028$, $F(4,100) = 2.28$, $p = .07$, $R^2 = 0.084$, respectively.

When MSLQ was the outcome variable, the regression models predicted the dependent variable significantly well. We tested different combinations of four coding indicators as independent variables, the Trial-Run indicator showed a greater prediction significance, $F(1,103) = 6.35$, $p = .013$, $R^2 = 0.058$, $\beta = 0.0009$, intercept = 4.58. Another model is the combination of Chosen-Task and Trial-Run, $F(2,102) = 3.66$, $p = .029$, $R^2 = 0.067$, MSLQ is equal to 4.44 + .01 (Chosen-Task) + 0.0007 (Trial-Run). However, when all four coding indicators were predictors, the prediction was not significant, $F(4,100) = 2.04$, $p = .09$, $R^2 = 0.075$.

3.4. Computational approach

To have a more in-depth look at the learners' computer coding behaviors, we analyzed their trial-run log data without combining or aggregating indicators. The first indicator is Chosen-Task, the second Trial-Run, the third Time, and the fourth Success. Each indicator of a participant could be represented as a multidimensional vector. For example, when a learner tested their Python codes (i.e., trial-run) 10 times in Task 1, 12 times in Task 2, 9 times in Task 3, …and 5 times in Task 50, the vector of Trial-Run for the student would be "(10, 12, 9, …, 5)." We used K-Means as a clustering method. We added the K-value after the indicator name; for example, when we used Chosen-Task as an indicator, and the K-value is 3, then the name was Chosen-Task-3. Thus, Chosen-Task-4 means we categorized learners' chosen-task indicators into 4 groups (i.e., clusters). Among these clustering processes, three attempts were not included in further analyses due to the lack of cases in a cluster. For example, in Trial-Run-4 (i.e., four clusters were identified using the trail-run indicators by the algorithm), the case numbers of each cluster are 86, 13, 5, and 1. Because the Trial-Run-5 and Time-5 had the same issue, further analyses did not proceed with these clusters.

3.5. Computational thinking skills

Most of the clustering approaches were not able to find significant differences between groups with regard to computational thinking skills – Chosen-Task-3: $F(2,102) = 0.926$, $p = .400$, Chosen-Task-4: $F(3,101) = 1.772$, $p = .157$, Trial-Run-3: $F(2,102) = 0.427$, $p = .653$, Success-3: $F(2,102) = 2.809$, $p = .065$, Success-4: $F(3,101) = 1.961$, $p = .125$, and Success-5, $F(4,100) = 1.610$, $p = .178$. However, the Time indicator found significant differences between groups in computational thinking skills (see Table 2). Time-3 showed a significant difference between three clusters, $F(2,102) = 3.965$, $p = .022$, Partial $\eta^2 = 0.072$, specifically, Creativity: $F = 9.699$, $p < .001$, Partial $\eta^2 = 0.160$, Algorithmic Thinking Skills: $F(2,102) = 4.288$, $p = .016$, Partial $\eta^2 = 0.078$, and Critical Thinking $F(2,102) = 3.866$, $p = .024$, Partial $\eta^2 = 0.070$. Although Time-4 did not show a difference between four clusters in Computational Thinking Skills, $F(3,101) = 1.420$, $p = .241$, there was a significant difference only in the Creativity dimension, $F(3,101) = 4.077$, $p = .009$, Partial $\eta^2 = 0.108$.

3.6. MSLQ

As shown in Table 2, many indicators showed significant differences between clusters regarding MSLQ. Chosen-Task-4 revealed a statistically significant difference between four clusters in MSLQ Total, $F(3,101) = 3.252$, $p = .025$, Partial $\eta^2 = 0.088$, specifically, Self-efficacy: $F(3,101) = 4.056$, $p = .009$, Partial $\eta^2 = 0.108$. Although Chosen-Task-3 did not show a significant difference in MSLQ Total, $F(2,102) = 2.473$, $p = .089$, there was a significant difference between three clusters in Self-efficacy, $F(2,102) = 3.643$, $p = .030$, Partial $\eta^2 = 0.067$. However, the post-hoc test was not able to reveal any meaningful differences between each pair of groups. This conflict is discussed in the limitation section.

Trial-Run-3 showed a difference in MSLQ, $F(2,102) = 4.569$, $p = .013$, Partial $\eta^2 = 0.082$, specifically, Cognitive Strategy: $F(2,102) = 4.102$, $p = .019$, Partial $\eta^2 = 0.074$ and Self-regulation: $F(2,102) = 3.782$, $p = .026$, Partial $\eta^2 = 0.069$.

The Time indicator also revealed a difference in MSLQ, Time-3: $F(2,102) = 7.416$, $p = .001$, Partial $\eta^2 = 0.127$. Specifically, Self-efficacy: $F(2,102) = 5.575$ $p = .005$, Partial $\eta^2 = 0.099$, Cognitive Strategy: $F(2,102) = 6.486$, $p = .002$, Partial $\eta^2 = 0.113$, and Self-regulation: $F(2,102) = 9.010$, $p < .001$, Partial $\eta^2 = 0.150$. In addition, Time-4 showed a difference between four clusters in MSLQ Total, $F(3,101) = 3.203$, $p = .026$, Partial $\eta^2 = 0.087$. Specifically, Cognitive Strategy: $F(3,101) = 3.066$, $p = .031$, Partial $\eta^2 = 0.083$, and Self-regulation: $F(3,101) = 2.711$, $p = .049$, Partial $\eta^2 = 0.075$. Because this Self-regulation component analysis showed a violation of homogeneity assumption (Levene Statistics = 3.212, $p = .026$), the Games-Howell was used for the post-hoc test.

The Success indicator did not reveal any meaningful differences, Success-3: $F(2,102) = 0.228$, $p = .796$, Success-4: $F(3,101) = 0.171$, $p = .916$, Success-5: $F(4,100) = 0.578$, $p = .680$.

3.7. Grades

While most of the clustered groups did not show any statistically significant differences in grades, one meaningful case was found. When the data were clustered by the participants' successful coding tasks into three groups, there was a significant grade difference between groups, $F(2,102) = 3.366$, $p = .038$, Partial $\eta^2 = 0.062$. However, the post-hoc test was not able to reveal any meaningful differences between each pair of groups. This is also addressed in the limitation section.

4. Discussion

In this study, we used a self-paced online learning environment to capture snapshots of students' code during Python programming tasks for the extracurricular activities and software course in a teacher education program. We used both aggregate and disaggregate data approaches to learners' computer programming patterns to identify the relationships with self-regulated learning, computational learning skills, and learning performance. The analysis of averaged indicators did not reveal any significant associations except for the Trial-Run measure, which showed its relationship with MSLQ scores. However, programming patterns, when a computational technique (i.e., k-means) utilized, revealed their relationships with self-regulated learning, computational learning skills, and learning performance. This is partially consistent with previous studies (e.g., Cigdem, 2015; Durak & Saritepeci, 2018; Yukselturk & Bulut; 2005) in that computer programming learning is related to self-regulated learning and computational thinking skills.

Since we have known that the time management aspect is closely related to self-regulated learning (Wolters et al., 2017), it is an interesting finding that the time spending patterns on specific coding tasks were related to learners' computational thinking skills, specifically the creativity dimension. It should be noted that this does not mean the total amount of time is related to creativity in computational thinking, as shown in our correlation and regression analyses. Instead, students with higher levels of creativity in computational thinking skills showed a specific pattern of time investment on certain tasks. The time parameter and temporal information could be an essential factor in the learning process, which is fundamentally ever-changing. Although there was a report that time management could be significant in computing education (Chaudhary et al., 2016), the relationship between computational thinking skills and time management has been under-explored. This requires further investigation.

Three aspects of coding behaviors (i.e., Chosen-Task, Trial-Run, and Time) were categorized into different groups that showed different levels of self-regulated learning. This means that the patterns of task selection, trial times on specific tasks, and time spent on specific tasks might be indicators of self-regulated learning. Since we were not able to intentionally design programming tasks with different characteristics, future research is needed to differentiate the features of each task so the learners' coding patterns on which task reveal the different levels of self-regulated learning.

Although limited, the patterns of successful tasks revealed the differences in course performance. Learners who showed a specific pattern of successful coding tasks achieved higher course grades than the other pattern groups. Still, the patterns identified by the computational algorithm were not labeled, characterized, or explained by the researchers due to the complexity. The success of a task did not influence the course grade because the coding tasks were the course participation components, and any students who logged in the online coding environment received the same points. Thus, it is quite interesting that higher performers showed a specific successful completion pattern. Besides, the success patterns did not reveal any differences in self-regulated learning or computational thinking skills, and the total number of successful tasks did not show any relationships with learning performance. We suspect that there might be another construct to explain this result, such as selective persistence (Prenzel, 1992). This warrants further research.

Overall, our investigation of the coding process is consistent with Castellanos et al. (2017), which showed the coding behavior (specifically, the length of codes) was associated with self-regulated learning. The difference between their results and ours is that we focused more on coding as a process rather than codes as a product. Our purpose is not to show any superiorities of an analysis approach over the other, but rather to reveal that a fine-grained data analysis might uncover the previously concealed relationships of learners' learning process and patterns with psychological indicators and learning performance. The fine-grained data could maintain learners' detailed trajectories throughout a learning activity. The use of fine-grained data with the time parameter to capture ever-changing learning patterns could offer novel insights into learning analysis. More importantly, we aimed to produce, although exploratory, a piece of theoretical evidence for the relationship of computer coding patterns with self-regulated learning and computational thinking skills. The implication of this study is that averaged predictors of learners' computer programming processes are less effective at reflecting learning performance, self-regulated learning, and computational thinking skills of the learners. Instead, the non-aggregated process-based indicators offer a clearer image to reveal those learner-side measures. It would be beneficial to utilize constantly shifting data of the learning process and discover any hidden patterns in the data (Blikstein et al., 2014), revealing relationships with existing constructs that education researchers have built. By collecting a stream of coding process data in an online coding program, we create opportunities to continuously assess their learning processes and progress.

If we identified clearer relationships between learners' coding behaviors/patterns and psychological constructs in future research, further investigations would demonstrate how these relationships can be used to provide proper scaffolds for the learners considering not only the better coding process and performance but also their current status of self-regulation and computational thinking skills. For example, these relationships would contribute to the field of programming hint, which has a growing number of algorithms with data-driven approaches (Price et al., 2019), such as supporting at-risk students in computer education programs (Tabanao et al., 2011). Our future intention is to take one step closer to providing individualized scaffolds that take into account not only the learner's codes but also learners' motivation, metacognition, cognitive strategy, algorithmic thinking skills. When scaffolding, the system not only provides a guide for the next coding step but also encourages learners' self-regulation process and facilitates their computational thinking. Automatic scaffolding should be influenced by learner-dependent psychological factors (Song & Kim, 2020). This is directly related to the computer education field of behavioral regulation, which highlights the importance of encouraging students' metacognitive activities (see Mangaroska et al., 2018). This field is the area where even expert human tutors might not be easily scaffolding computer programming learners.

4.1. Limitations and future research

Our work has important limitations. First, our narrow focus on the relationship with self-regulated learning and computational thinking skills might overlook the programming content; that is, semantic and syntactic aspects of learners' code. Different types of programming tasks need to be designed to examine learners' code contents in future studies. Second, the difference in the patterns of coding size changes that other researchers suggested (e.g., Blikstein et al., 2014) was not analyzed, and their predictive power was not tested in this study. The reason for not selecting the size approach was that most of the tasks were novice-levels, so we considered that the overall size of each task was not meaningful. More intentionally designed tasks should be studied in future research. Third, learners' codes were not analyzed qualitatively due to the massive amount of data. Still, we acknowledge that there should be sufficient information in their codes and qualitative approaches would reveal the original patterns of learners' programming behavior. More efficient and feasible methods for qualitative analysis of learners' computer codes are needed for future research. Last, the non-significant results of post-hoc tests might be caused by the small effect sizes and the uneven cases of each cluster. Empirical studies with a larger sample would be required.

5. Conclusion

This study differs from previous computer programming education research in three key ways. First, we presented a process model for organizing the learners' coding process. Second, we expanded the focus of the coding process to include computational thinking skills and self-regulated learning. Finally, we identified the relationships between learners' psychological constructs and the coding process.

Credit author statement

Donggil Song: Software, Validation, Methodology, Formal analysis, Writing. **Hyeonmi Hong**: Investigation, Resources, Data Curation. **Eun Young Oh**: Conceptualization, Methodology, Visualization.

Funding sources

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Availability of data and material

The data will not be shared due to the confidentiality issues.

Declaration of competing interest

No potential conflict of interest was reported by the authors.

References

Alharbi, A., Henskens, F., & Hannaford, M. (2012). Student-centered learning objects to support the self-regulated learning of computer science. *Creative Education, 3*(6), 773–783. https://doi.org/10.4236/ce.2012.326116

Altadmri, A., & Brown, N. C. C. (2015). *37 million compilations: Investigating novice programming mistakes in large-scale student data. Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15), 4-7 March 2015, Kansas City, MO, USA*. New York, NY: ACM. https://doi.org/10.1145/2676723.2677258

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems, 75*, 661–670. https://doi.org/10.1016/j.robot.2015.10.008

Azcona, D., Hsiao, I. H., & Smeaton, A. F. (2019). Detecting students-at-risk in computer programming classes with learning analytics from students' digital footprints. *User Modeling and User-Adapted Interaction, 29*(4), 759–788. https://doi.org/10.1007/s11257-019-09234-7

Bergin, S., Reilly, R., & Traynor, D. (2005). Examining the role of self-regulated learning on introductory programming performance. *Proceedings of the International Workshop on Computing Education Research*, 81–86. https://doi.org/10.1145/1089786.1089794

Bishop-Clark, C., Courte, J., & Howard, E. V. (2006). Programming in pairs with Alice to improve confidence, enjoyment, and achievement. *Journal of Educational Computing Research, 34*(2), 213–228. https://doi.org/10.2190/CFKF-UGGC-JG1Q-7T40

Blikstein, P., Worsley, M., Piech, C., Gibbons, A., Sahami, M., & Cooper, S. (2014). Programming pluralism: Using learning analytics to detect patterns in novices' learning of computer programming. *International Journal of the Learning Sciences, 23* (4), 561–599. https://doi.org/10.1080/10508406.2014.954750

Brito, S. R., Silva, A. S., Tavares, O. L., Favero, E. L., & Francês, C. R. L. (2011). *Computer supported collaborative learning for helping novice students acquire self-regulated problem-solving skills in computer programming*. Athens: Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS). https://search.proquest.com/docview/1270655747.

Burbaitė, R., Drąsutė, V., & Štuikys, V. (2018). *Integration of computational thinking skills in STEM-driven computer science education. Proceedings of the Global Engineering Education Conference (EDUCON), 2018 IEEE* (pp. 1824–1832). IEEE. https://doi.org/10.1109/EDUCON.2018.8363456

Carter, A. S., Hundhausen, C. D., & Adesope, O. (2015). The normalized programming state model: Predicting student performance in computing courses based on programming behaviour. In *Proceedings of the 11th annual Conference on international computing education research (ICER 2015), 9-13 july 2015* (pp. 141–150). USA: Omaha, Nebraska. https://doi.org/10.1145/2787622.2787710.

Castellanos, H., Restrepo-Calle, F., González, F. A., & Echeverry, J. J. R. (2017). Understanding the relationships between self-regulated learning and students source code in a computer programming course. In *Proceedings of the 2017 IEEE frontiers in education conference (FIE), 18-21 Oct. 20*. Indianapolis: IEEE. https://doi.org/10.1109/FIE.2017.8190467.

Chaudhary, V., Agrawal, V., Sureka, P., & Sureka, A. (2016). An experience report on teaching programming and computational thinking to elementary level children using lego robotics education kit. In *Proceedings of the 2016 IEEE eighth international conference on technology for education (T4E)* (pp. 38–41). IEEE. https://doi.org/10.1109/T4E.2016.016.

Chen, C. S. (2002). Self-regulated learning strategies and achievement in an introduction to information systems course. *Information Technology, Learning, and Performance Journal, 20*(1), 11–25. https://pdfs.semanticscholar.org/d92f/d0d0207191f7806852f93c3f37d61a4438eb.pdf.

Cigdem, H. (2015). How does self-regulation affect computer-programming achievement in a blended context? *Contemporary Educational Technology, 6*(1), 19–37. https://doi.org/10.30935/cedtech/6137

Doleck, T., Bazelais, P., Lemay, D. J., Saxena, A., & Basnet, R. B. (2017). Algorithmic thinking, cooperativity, creativity, critical thinking, and problem solving: Exploring the relationship between computational thinking skills and academic performance. *Journal of Computers in Education, 4*(4), 355–369. https://doi.org/10.1007/s40692-017-0090-9

Durak, H. Y., & Saritepeci, M. (2018). Analysis of the relation between computational thinking skills and various variables with the structural equation model. *Computers & Education, 116*, 191–202. https://doi.org/10.1016/j.compedu.2017.09.004

Echeverry, J. J. R., Rosales-Castro, L. F., Restrepo-Calle, F., & González, F. A. (2018). Self-regulated learning in a computer programming course. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje, 13*(2), 75–83. https://doi.org/10.1109/RITA.2018.2831758

Fields, D. A., Quirke, L., Amely, J., & Maughan, J. (2016). Combining big data and thick data analyses for understanding youth learning trajectories in a summer coding camp. In *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 150–155). https://doi.org/10.1145/2839509.2844631

Gardeli, A., & Vosinakis, S. (2017). Creating the computer player: An engaging and collaborative approach to introduce computational thinking by combining 'unplugged' activities with visual programming. *Italian Journal of Educational Technology, 25*(2), 36–50. https://www.learntechlib.org/p/183469/.

Giannakos, M. N., Pappas, I. O., Jaccheri, L., & Sampson, D. G. (2017). Understanding student retention in computer science education: The role of environment, gains, barriers and usefulness. *Education and Information Technologies, 22*(5), 2365–2382. https://doi.org/10.1007/s10639-016-9538-1

Hall, D. J., Cegielski, C. G., & Wade, J. N. (2006). Theoretical value belief, cognitive ability, and personality as predictors of student performance in object-oriented programming Environments. *Decision Sciences Journal of Innovative Education, 4*(2), 237–257. https://doi.org/10.1111/j.1540-4609.2006.00115.x

Ihantola, P., Sorva, J., & Vihavainen, A. (2014). Automatically detectable indicators of programming assignment difficulty. In *Proceedings of the 15th annual conference on information technology education (SIGITE/RIIT 2014), 15-18 October 2014* (pp. 33–38). Atlanta, GA, USA: ACM. https://doi.org/10.1145/2656450.2656476. New York.

Jadud, M. C. (2006). Methods and tools for exploring novice compilation behaviour. In *Proceedings of the 2nd international workshop on computing education research (ICER 2006), 9-10 september 2006, canterbury, United Kingdom* (pp. 73–84). New York: ACM. https://doi.org/10.1145/1151588.1151600.

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the Computational Thinking Scales (CTS). *Computers in Human Behavior, 72*, 558–569. https://doi.org/10.1016/j.chb.2017.01.005

Kumar, V., Winne, P. H., Hadwin, A. F., Nesbit, J. C., Jamieson-Noel, D., Calvert, T., & Samin, B. (2005). Effects of self-regulated learning in programming. *Proceedings of the 5th IEEE International Conference on Advanced Learning Technologies (ICALT '05)*, 383–387. https://doi.org/10.1109/ICALT.2005.131

Liu, Z., Zhi, R., Hicks, A., & Barnes, T. (2017). Understanding problem solving behavior of 6–8 graders in a debugging game. *Computer Science Education, 27*(1), 1–29. https://doi.org/10.1080/08993408.2017.1308651

Lu, J. J., & Fletcher, G. H. (2009). Thinking about computational thinking. *ACM SIGCSE Bulletin, 41*(1), 260–264. https://doi.org/10.1145/1508865.1508959

Mangaroska, K., Sharma, K., Giannakos, M., Træteberga, H., & Dillenbourg, P. (2018). Gaze-driven design insights to amplify debugging skills: A learner-centred analysis approach. *Journal of Learning Analytics, 5*(3), 98–119. https://doi.org/10.18608/jla.2018.53.7

Pedrosa, D., Cravino, J., Morgado, L., & Barreira, C. (2016). Self-regulated learning in computer programming: Strategies students adopted during an assignment. In *International conference on immersive learning, iLRN 2016: Immersive learning research network* (pp. 87–101). Springer. https://link.springer.com/chapter/10.1007/978-3-319-41769-1_7.

Pintrich, R. R., & DeGroot, E. V. (1990). Motivational and self-regulated learning components of classroom academic performance. *Journal of Educational Psychology, 82*(1), 33–40.

Prenzel, M. (1992). The selective persistence of interest. In K. A. Renninger, S. Hidi, & A. Krapp (Eds.), *The role of interest in learning and development* (pp. 71–98). Lawrence Erlbaum Associates, Inc. https://psycnet.apa.org/record/1992-97926-004.

Price, T. W., Dong, Y., Zhi, R., Paaßen, B., Lytle, N., Cateté, V., & Barnes, T. (2019). A comparison of the quality of data-driven programming hint generation algorithms. *International Journal of Artificial Intelligence in Education, 29*(3), 368–395. https://doi.org/10.1007/s40593-019-00177-z

Schulte, C. (2013). Reflections on the role of programming in primary and secondary computing education. *Proceedings of the 8th Workshop in Primary and Secondary Computing Education*, 17–24. https://doi.org/10.1145/2532748.2532754

Soloway, E., Lochhead, J., & Clement, J. (1982). Does computer programming enhance problem solving ability? Some positive evidence on algebra word problems. In R. Seidel, R. Anderson, & B. Hunter (Eds.), *Computer literacy* (pp. 171–185). Academic Press. https://doi.org/10.1016/B978-0-12-634960-3.50023-3.

Song, D. (2018). Learning Analytics as an educational research approach. *International Journal of Multiple Research Approaches, 10*(1), 102–111. https://doi.org/10.29034/ijmra.v10n1a6

Song, D., & Kim, D. (2020). Effects of self-regulation scaffolding on online participation and learning outcome. *Journal of Research on Technology in Education*. https://doi.org/10.1080/15391523.2020.1767525. Advance online publication.

Sun, J. C. Y., & Hsu, K. Y. C. (2019). A smart eye-tracking feedback scaffolding approach to improving students' learning self-efficacy and performance in a C programming course. *Computers in Human Behavior, 95*, 66–72. https://doi.org/10.1016/j.chb.2019.01.036

Tabanao, E. S., Rodrigo, M. M. T., & Jadud, M. C. (2011). Predicting at-risk novice Java programmers through the analysis of online protocols. In *Proceedings of the 7th international workshop on computing education research* (pp. 85–92). New York, NY: ACM. https://doi.org/10.1145/2016911.2016930.

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies, 20*(4), 715–728. https://doi.org/10.1007/s10639-015-9412-6

Watson, C., Li, F. W. B., & Godwin, J. L. (2013). Predicting performance in an introductory programming course by logging and analyzing student programming behaviour. In *Proceedings of the 13th international Conference on advanced learning technologies (ICALT 2013), 15-18 july 2013* (pp. 319–323). Beijing, China: IEEE Computer Society. https://doi.org/10.1109/ICALT.2013.99.

Watson, C., Li, F. W., & Godwin, J. L. (2014). No tests required: Comparing traditional and dynamic predictors of programming success. In *Proceedings of the 45th ACM technical symposium on computer science education* (pp. 469–474). New York, NY: ACM. https://doi.org/10.1145/2538862.2538930.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wolters, C. A., Won, S., & Hussain, M. (2017). Examining the relations of time management and procrastination within a model of self-regulated learning. *Metacognition and Learning, 12*(3), 381–399. https://doi.org/10.1007/s11409-017-9174-1

Yukselturk, E., & Bulut, S. (2005). Relationships among self-regulated learning components, motivational beliefs and computer programming achievement in an online learning environment. *Mediterranean Journal of Educational Studies, 10*(1), 91–112. https://www.um.edu.mt/library/oar/handle/123456789/19417.

Zimmerman, B. J. (1990). Self-regulated learning and academic achievement: An overview. *Educational Psychologist, 25*(1), 3–17. https://doi.org/10.1207/s15326985ep2501_2

Zimmerman, B. J. (2000). Attaining self-regulation: A social cognitive perspective. In M. Boekaerts, P. R. Pintrich, & M. Zeidner (Eds.), *Handbook of self-regulation* (pp. 13–39). Academic Press.