Bridging the Gap: Template-Based Coding for Transitioning from Visual Logic to Text-Based Programming

Samer Y. Al-Imamy

Department of Management Information Systems, Prince Mohammad bin Fahd University, AlKhobar, Saudi Arabia
Email: salimamy@pmu.edu.sa (S.Y.A.-I.)

Manuscript received September 20, 2023; revised November 15, 2023; accepted November 27, 2023; published March 12, 2024

Abstract—**Visual programming environments, which utilize visual blocks or flowcharts to represent programming logic, have emerged as a key strategy to assist novice learners in overcoming the complexities associated with text-based programming. However, transitioning from these visual representations to full-text programming often presents a significant challenge for these learners. One solution to this challenge is the use of template-based coding, which has been shown to increase performance and reduce the number of errors made by students studying computer programming. This study used an experimental assignment and survey to evaluate the Code-By-Template (CBT) application among 82 students, revealing enhanced coding proficiency and positive student attitudes, highlighting the effectiveness of CBT in programming learning. The CBT, in particular, has been found to enhance students' performance in successfully solving programming problems. The observed improvement in performance can be attributed to a 17% increase in scores (correctness) during the same time frame, suggesting a decrease in the number of errors. It is important to acknowledge that the scores mostly depend on the number and type of errors. Beyond this performance improvement, students have also expressed interest, describing the CBT as "extremely helpful" and "making programming easier." This highlights the qualitative benefits of the tool in promoting appreciation for it, indicating its potential to enhance engagement as well as learning outcomes. The difficulty of learning programming code has had a persistent impact on retention rates in computer programming courses. However, the improved performance and learning ease facilitated by the CBT environment may offer a solution to this retention problem. By making the learning process more manageable and less error-prone, the CBT environment can help to ensure confidence-boosting and interest-fostering that lead to more students successfully completing their programming courses and continuing their studies in this field.**

Keywords—**code by template, programming for novice learners, visual programming, program logic, performance enhancement, interest and retention**

I. Introduction

Computer programming is a crucial and essential course for information technology-related students. Due to the language learning challenges, the selection of teaching methods is a frequent research topic [1, 2]. In addition to their indispensability in various fields, programming languages are used as tools for developing critical thinking skills in business [3], medicine [4], and other disciplines.

Programming languages are categorized by most students and researchers as one of the most challenging tasks due to several factors [5] that lead to a high rate of dropout [6] and failure [7, 8] in introductory programming courses. Logical thinking [9], reasoning and creativity [10], syntax control [5], and logical analysis [2] are some of these challenging factors that researchers assumed were the cause of the problem. Therefore, it is urgent to solve the issues facing the learners [11].

Numerous studies proposed enhancement of the understanding of the programming logic through the use of tools such as visual blocks or flowcharts [3]. However, the transition from the programming logic level into full-text programming represents a big challenge for the learners.

The purpose of this research is to ease the transition from the programming logic level, represented as visual blocks or flowcharts, into full-text programming using a purposefully built interactive scaffold. In addition to circumventing syntax memorization problems, our research endeavors to enhance student learning experiences and increase retention rates in computer programming courses. Furthermore, the present study aims to evaluate the general reception of the CBT tool among students and its ability to enhance confidence and proficiency in programming assignments. Therefore, the research question could be drafted as follows: *"How does the implementation of Code By Template (CBT) impact students' programming performance and increase their confidence levels?"*

The study uses the Technology Acceptance Model (TAM) and the Kirkpatrick model of training evaluation to measure students' acceptance and utilization of the CBT application. TAM emphasizes perceived ease of use and usefulness, while the Kirkpatrick model evaluates the tool's effectiveness across dimensions like student reactions, learning outcomes, behavioral changes, improved programming skills, and reduced dropout rates. The literature review and related work are introduced in the next section. Section III explains the methodology, which is based on an experiment and a survey. The results are listed in Section IV, followed by the discussion and limitations. The last section is devoted to the conclusions.

II. Literature Review

Programming is a complex subject that demands both theoretical and practical skills. A study conducted by Veerasamy *et al.* [12] found that high problem-solving skills could help students perform better in programming. Malik *et al.* [13] proposed the PAAM (Problem Analysis Algorithmic Model) to assist novices in their learning curve. On the other hand, coding enhances problem solving skills [14], which is one of the eight key competencies of the 21st century as defined by UNESCO (2017). Several approaches were used by researchers to improve the programming learners' abilities, such as Augmented Reality (AR) [15] and blended learning [16, 17]. Tsai *et al.* [9] used the AR-based logic programming system to prove its superiority in motivation and effectiveness over the traditional learning method.

Learning programming languages as a novice can be seen as an analogy to learning to drive a car. As well as the theoretical challenges, driving a manual car practically is a very difficult task for beginners because it requires, at the very least, road focus and gear-changing attention that includes clutch balance. Similarly, program coding requires a focus on both logic and syntax. Accordingly, introducing the logic first is similar to learning to drive an automatic car. Handling the logic that leads to a solution could be represented by a flowchart, pseudocode, or even mathematical logic. AI-Imamy *et al.* [3] proved the advantage of shielding the logic from the syntax through the measurements of the ease of use, usefulness, versatility, and performance produced by the logic-first approach.

According to Altadmri and Brown [18], the majority of semantic errors are followed by syntax errors, which is a difficult and painful task for new learners due to a number of factors, including the rigidity of the language structure and syntax [5]. The 18 syntax mistakes committed by students were identified by the researchers as contributing to the new learners' challenges [19] that may have provoked the 32% of learners who failed to pass the programming course during the first attempt, as found by Watson and Li [20]. The difficulty in learning the programming code had an impact on the retention statistics, which have remained affected over the past decade [21].

Changing the programming language is common for people working in the IT field, but it is daunting when they need such a shift due to courses' requirements or in the marketplace. Therefore, giving the learners the ability to work with syntax-independent constructs increases their confidence [22]. The syntax error warnings and compiler feedback both contribute significantly to the identification of the issues. However, such messages are frequently misleading to novice learners, even when generated by modern Integrated Development Environment (IDE), which are described as cryptic by such an audience [23]. In light of this, a new IDE is required. Such an environment is needed as a code generator (a template) with almost no errors.

Several approaches have been introduced by researchers during the last couple of decades to tackle this complexity and the discouraging retention statistics. According to the computer programming education community, research and development activities should be increased in order to improve the quality of instruction and increase the number of highly skilled instructors. A first step toward programming skills enhancement for schools was taken by former U.S. president Obama's Computer Science for All project (Obama White House Archives, 2016). Additionally, Former U.S. president Trump has allocated a sizable amount of federal funding for computer science education. Professional associations, governments, and private and non-profit organizations have all worked to increase participation and success in computer science education. Google, Facebook, and Amazon also contributed funding to expand the computer science major pipeline (EdSurge, 2017). Therefore, substantive research and development organizations are now involved in boosting pedagogical and technological strategies [24]. Visual programming environments are one of the main strategies used over the past couple of decades to help novice learners get over the complexity of text-based approaches [25]. For example, introductory programming learners perceived block programming [26], the Logo environment [27], and introductory programming languages such as "Alice" [28], "Greenfoot," and "Scratch" [29] as a step toward understanding full syntax-based languages. Although these techniques are helpful for comprehending programming principles, they provide little help with the move to code [19]. In their recent study, AbdulSamad and Romli [30] compared a number of block-based platforms. The objective of their study is to identify those that promote effective programming learning and have the potential to accelerate code development.

Recent studies have focused on the Natural Language Interface (NLI) as a visual/pseudocode to text transition. Ansari *et al.* [22] created the NLI application to interpret pseudocode-like statements into Java code in an attempt to let students focus on the problem-solving issue. The proposed environment focuses on the syntactical side of the entered sentences in the hope of recognizing the meaning and translating it into a runnable code. Thomas *et al.* [31] stressed the importance of handling the ambiguity issue when the natural language is processed to preserve the meaning of the sentences that are converted into precise programming language statements. Although the proposed applications assist users in writing source code with no programming knowledge, they make only a minor contribution to the development of critical thinking skills.

The integration of AI applications, such as ChatGPT, into the domain of coding and programming is experiencing a growing trend. The research investigated the effects of Artificial Intelligence (AI) tools, specifically ChatGPT and GitHub Copilot, on programming within an academic setting [32]. The researchers emphasized the existence of varying perspectives among educators on the incorporation of these tools into their instructional practices. While many educators may advocate for their prohibition in order to prioritize the teaching of programming principles, others recognize the significance of their inclusion as a means of equipping students with the necessary skills for prospective employment opportunities. The study offers an overview of the initial phases of incorporating and adjusting to the swiftly developing AI coding tools within educational environments.

As a result, summarizing the language in a limited number of phrases, as we did, may be more suitable at this stage for critical thinking and coding experience.

While critical, logical, or cognitive thinking forms one of the wings of the programming challenges, as cited above, syntax control represents the second. Students, after they understand the logic, are required to start the coding, where they face the syntax challenges. From learners' failure to meet the tutor's expectations [33] to significant improvements in students' results [34, 35], research in code writing has covered the gamut. The improvement in performance was mainly due to the reduction in the amount of overwhelming information that the learners needed to process. As a result, pedagogical IDE customization is required to provide enough information to help students focus on program development while reducing error message deciphering [11]. Accordingly, a new IDE that could produce perfect code (a template) with almost no errors is needed.

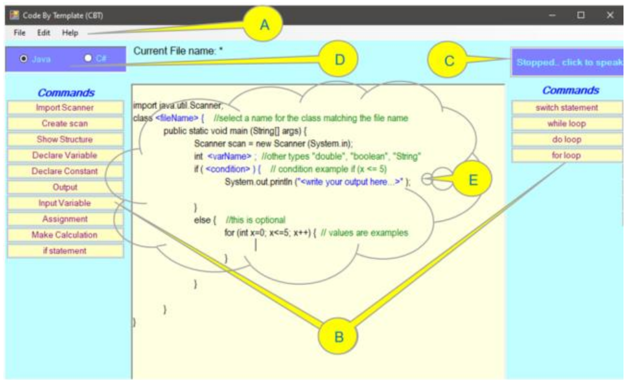
This work, using a customized IDE developed for the purpose, is trying to prove the effectiveness of the use of templates in enhancing learners' abilities and confidence. Researchers reported the frustration and discouragement of novice learners due to the unclear and sometimes misleading syntax errors they encountered during their first steps in the programming field [36, 37]. Most of the errors are not explained well enough to the learners, and a thorough investigation to find the root of the error is a vital task [38]. The majority of such errors made by new learners are naive and can be avoided by using the Code By Template (CBT) that we propose as an addition to the existing IDEs. Microsoft introduced snippets for some essential commands used in their Visual Studio environment [39] that need evaluation of their effect on the learning curve. Several studies during the past few years have tried to compare the efficiency of block-based computer programming against text-based computer programming [40, 41]. Since the learning of a block-based approach cannot replace a text-based approach, such a comparison doesn't lead to learners' coding improvement [42].

III. Methodology

This research uses two instruments: the first deals with the students' results of an experiment using exercise questions prepared for this purpose and solved by the learners with and without the CBT application; the second is a survey to understand the students' opinions of the application.

The IDE was created by the researcher in C# using the Visual Studio platform. As a learning tool, our research uses sentences controlled by a limited set of phrases that could be selected as buttons or uttered as a human voice. Fig. 1 shows the application with its conventional menu that is used to save, open, edit, and so forth (A). The application contains a set of commands, marked (B). These commands can be activated by clicking or spoken by the learner if the speech feature is enabled. The latter can be enabled by using the right-top button (C). The speech-enabled application, using Microsoft speech recognition, is an additional function of the introduced platform. The commands (template generators), which are one of the main characteristics of the scaffold, are adjusted based on the language selected from the top-left radio button (D). Only Java and C# are currently available, but other languages can be added easily. The main area in the middle is used for the generated constructs (E). This editing area, which represents the core of the tool, is used to set and overlap (nest) the templates inevitably based on the user's requirements. To regenerate the code seen in the picture, the user may click or say the commands (Import Scanner, Show Structure, Create Scan, Declare Variable, If Statement, Output "in the true route of the if section," and For Loop "in the else route of the section"). The user can choose any combination of commands to generate the structure.

The proposed IDE provides learners with buttons for simple template generation. It is a perfect scaffold for the learners, especially when construct nesting is required. All the generated templates come with comments and examples when possible. Words requiring the user's interaction, such as variable names and conditions, are highlighted in a different color and surrounded by angle brackets.



```
import java.util.Scanner;
class <fileName> {    //select a name for the class matching the file name
        public static void main (String[] args) {
                Scanner scan = new Scanner (System.in);
                int  <varName> ;  //other types "double", "boolean", "String"
                if ( <condition> ) {    // condition example if (x <= 5)
                        System.out.println ("<write your output here...>" );

                }
                else {    //this is optional
                        for (int x=0; x<=5; x++) {  // values are examples

                        }

                }

        }
}
```

Fig. 1. The proposed Code-By-Template (CBT) integrated user interface.

The use of CBT application offers advantages over conventional IDEs, as it produces robust code that, otherwise, may produce trivial side-effect mistakes. It will increase focus and confidence, which will lead to better performance and a deeper understanding of the programming constructs. The application is simpler than the need for a natural language interface, though it is a step towards the fifth generation of artificial intelligence programming, especially with its speech interface. The switch between the different programming languages will be as simple as selecting the radio button for the selected language.

A. The Experiment

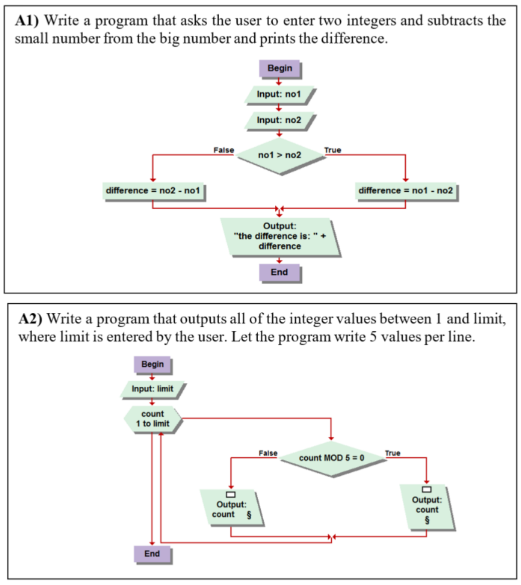
To apply the experiment to a reasonable number of students, several classes in more than one semester are needed because the capacity of the lab is limited to a maximum of 24 students. Accordingly, dividing the subjects into control and treatment groups is inappropriate. Therefore, within-subjects design is applied to 82 students in the introduction to programming course. To fix all the variables, including the individuals' abilities, the effect of CBT (the independent variable) on performance (the dependent variable) is measured for every student. This design can therefore be applied over several semesters and to many classes.

Within each class, the students are divided into two groups. Each group receives different but comparable questions (2 questions each). Since the objective of the research is to measure the impact of the CBT on the students' ability to write the code, the understanding logic must be neutralized. Therefore, each student within the two groups was given the flowchart of the questions as a handout (see Fig. 2 for a sample of the flowcharts).

Therefore, the task is reduced to translating the flowcharts into codes, which is the goal of the research. The questions solved by the two groups are different but almost equal in difficulty (typically, one of them contains the decision construct and the other has a looping construct). The programming languages used for coding were Java or C#.

When the students completed their first tasks without any assistance, a brief explanation of the CBT was introduced for use with the second round of the experiment. Students were assigned two new questions in this stage of the experiment, i.e., B1 and B2, which were given to the first group and A1 and A2 to the second, to solve with the assistance of the CBT application. The two questions were focused on the main programming constructs to be sure the CBT has the same effect on the different constructs. The purpose of splitting the class into two groups and switching the questions is to make sure that the questions have not been seen before and that the experiment is unaffected by the slight variations in the questions.

The results of the experiment were recorded and analyzed in the results section.


A1) Write a program that asks the user to enter two integers and subtracts the small number from the big number and prints the difference.

A2) Write a program that outputs all of the integer values between 1 and limit, where limit is entered by the user. Let the program write 5 values per line.

Fig. 2. Samples of the questions along with their flowcharts.

B. The Survey

To support the above experiment, a survey was distributed to students who had tried the application to assess their attitudes and feelings towards it. These students were asked seventeen questions through a survey to find out their impressions of the application, as well as one open question for them to write their comments. The questions were derived from the Technology Acceptance Model (TAM) as a foundation but were contextually modified through the addition of additional variables that measure the main characteristics of the application. The Appendix shows the thematically classified questions in Table A.

In summary, our research approach encompassed a thorough examination of both qualitative and quantitative data. By giving programming assignments both before and after the CBT tool's adoption, the quantitative data included performance scores that also reflect the error rates. Additionally, the administration of surveys and feedback forms helped to facilitate the collection of qualitative data by eliciting insightful information about the students' experiences with and perceptions of the CBT instrument. The extensive dataset underwent a systematic examination utilizing statistical tests, such as t-tests for evaluating performance comparisons and theme analysis for analyzing qualitative replies.

The format in which our findings will be presented will be organized in a manner that effectively highlights the influence of the CBT tool on the development of students' programming abilities. The quantitative findings will be presented through a collection of tables and graphs, providing a visually accessible depiction of the enhancements in performance. Together with qualitative findings like direct quotes from student feedback, these will help us get a full picture of how the CBT tool works in the real world and how it affects learning. This approach will provide contextual information and in-depth analysis. The utilization of a dual strategy in evaluating the efficacy of the CBT tool in augmenting programming education guarantees a thorough comprehension of its efficiency.

IV. Results

A. Performance Analysis

The learners were divided into two groups. In the first round, both groups solved their first assigned set of questions (A for group 1 and B for group 2) without the CBT application's support, while in the second round, both groups solved the second set of questions (B for group 2 and A for group 1), where the application was used to assist them in writing the code in either Java or C# programming languages. The initial view of the results from 82 comparisons shows more than half of the learners improved their performance with the assistance of the CBT application. The t-test analysis of Table 1 shows a significant enhancement when the students use the CBT application.

Table 1. The t-Test of two-sample assuming unequal variances

Statistical Measure	With CBT	No CBT
Mean	7.25	6.19
Variance	4.70	7.81
Observations	82	82
df	153	
t-Stat	2.72	
$P(T \leq t)$ one-tail	0.004	
Std. Deviation	2.17	2.80

The result of a two-sample t-test demonstrated a significant difference between the two groups ($t = 2.17$, $p < 0.05$). This implies that CBT contributes to student performance in solving the questions successfully, proving the first half of the research question. As a useful tool, the students are encouraged to use the CBT scaffold as long as they feel the benefit of the application support. However, they are expected to move away gradually until they feel comfortable writing the different constructs without support.

B. The Survey

Following the experiment, 109 students who tried the application (including the 86 participants in the experiment) were asked to answer a survey that measures their acceptance of the CBT application. The number of males who participated in the survey was 73, while the number of females was 36. All the participants are from the college of business except one from IT, distributed as 53 Management Information Systems (MIS), 28 Business Administration (BA), and 27 Finance (FINA) students. The course under study is compulsory for the MIS, which means most of them are at the sophomore level (age 19–23). However, the programming is an elective course for the BA and FINA majors. Most of the students in these two majors are juniors or seniors (age 21–25).

Some of the students had a chance to practice the application without participating in the experiment. It was found that no significant difference between the responses of those who participated in the experiment and those who only tried the CBT application and answered the survey; $t(30) = -0.32$, $p = 0.75$ (Fig. 3).

The descriptive statistics are illustrated in Table 2.

Table 2. The descriptive statistics of the survey

Theme	Valid N	Missing	Mean	Std. Error	Std. Dev.	Variance
Enjoyment	109	0	4.35	0.08	0.89	0.79
Navigability	109	0	4.16	0.09	0.94	0.89
Complexity	109	0	4.27	0.10	1.01	1.01
Reliability	109	0	4.22	0.09	0.95	0.90
Confidence	109	0	4.18	0.10	1.01	1.02
Continuity	109	0	4.15	0.10	1.03	1.05
Understanding	109	0	4.06	0.09	0.97	0.95
Empowerment	109	0	4.20	0.10	1.00	1.00
Ease of use	109	0	4.28	0.09	0.98	0.96
Usefulness	109	0	4.30	0.09	0.97	0.94
Learnability	109	0	3.68	0.11	1.12	1.26
Preference	109	0	4.12	0.09	0.96	0.92
Practicability	109	0	4.19	0.10	1.02	1.05
Interest	109	0	4.18	0.09	0.95	0.91
Preference	109	0	3.83	0.11	1.13	1.27
Embarrassment	109	0	3.96	0.11	1.16	1.35
Usage through	Voice (4%)	Button (38%)	Both (45%)	None (13%)		

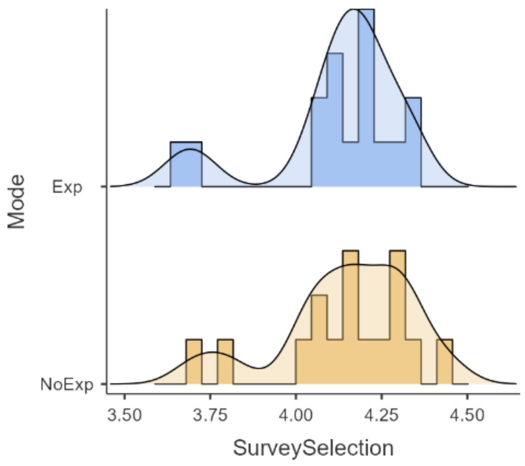


Fig. 3. Difference between the survey responses of the participants and non-participants in the experiment.

C. Quantitative Analysis

The results of the quantitative analysis are shown in Table 3.

The data reveals that all of the CBT application advantages mentioned previously are significantly met based on the students' opinions. These advantages are reflected in high scores for enjoyment, reliability, confidence, control, ease of use, usefulness, performance, and others, proving the second half of the research question. However, some students still need the assistance of the instructor and the classroom environment instead of self-studying (learnability t = 6.3). Most students prefer the use of CBT; however, a few of them (probably the experienced) are preferring direct coding (t = 7.7). Although the application helped in reducing the embarrassment (t = 8.7), few still feel it (if it ever existed). As for the use of buttons or voice, it looks like both are preferable.

Table 3. Survey's one-sample test (value = 3)

Theme	Strongly Disagree WT = 1	Disagree WT = 2	Neutral WT = 3	Agree WT=4	Strongly Agree WT = 5	t	df	Sig.
Enjoyment	3%	2%	11%	30%	54%	15.89	108	<.001
Navigability	4%	1%	13%	41%	41%	12.78	108	<.001
Complexity	3%	3%	10%	31%	53%	13.14	108	<.001
Reliability	4%	1%	11%	39%	45%	13.46	108	<.001
Confidence	3%	3%	13%	33%	48%	12.23	108	<.001
Continuity	3%	4%	16%	29%	48%	11.67	108	<.001
Understanding	4%	0%	22%	35%	39%	11.40	108	<.001
Empowerment	4%	4%	11%	34%	47%	12.57	108	<.001
Ease of use	4%	1%	13%	29%	53%	13.58	108	<.001
Usefulness	4%	1%	11%	29%	55%	14.14	108	<.001
Learnability	6%	7%	28%	32%	27%	6.32	108	<.001
Preference	3%	3%	15%	38%	41%	12.18	108	<.001
Practicability	3%	1%	14%	38%	44%	12.18	108	<.001
Interest	5%	3%	31%	25%	36%	12.95	108	<.001
Preference	6%	6%	16%	30%	42%	7.74	108	<.001
Embarrassment	4%	6%	25%	33%	32%	8.65	108	<.001
Usage through	Voice (4%)	Button (38%)	Both (45%)	None (13%)				

D. Surveyors' Comments

Samples of the students' comments about the application are as follows:

1) Very good, reliable and useful;
2) It needs some improvements for perfection;
3) Creative impressing, wonderful and amazing idea;
4) Nice and easy to use;
5) Extremely helpful, simple and efficient application that saves time;
6) Interesting and great experience to try;
7) Very useful and beneficial that make programming easy for beginners;
8) I hope we can use this app next term i like it;
9) It made writing code easier and a much pleasant experience;
10) The application can be very good for someone who is

confused, like me;

11) One of the best apps ever;
12) It is perfect program that helps us to learn and not to fail in coding;
13) The idea of this application is impressing. It was a nice experience, definitely will recommend it to my instructors and classmates in other courses if needed;
14) The app is extremely helpful and it made the code much easier and made me avoid silly mistakes and safe much time;

These comments confirm the extra advantages of the CBT over the conventional IDEs and prove the compliant of training through Kirkpatrick Model of Training evaluation framework [43] as shown in Table 4.

Table 4. Alignment of the students' comments with the Kirkpatrick model of training

Lvl#	Evaluation	Supporting comments
1	Reaction	(1) Very good, reliable and useful (3) Creative impressing, wonderful and amazing idea (4) Nice and easy to use (6) Interesting and great experience to try (11) One of the best apps ever
2	Learning	(7) Very useful and beneficial that make programming easy for beginners
3	Behavior	(9) It made writing code easier and a much pleasant experience
4	Results	(5) Extremely helpful, simple and efficient application that saves time

V. Discussion

The literature demonstrated that programming is a difficult subject, to the extent that introductory programming courses frequently have significant dropout and failure rates. Our results imply that the use of the CBT program can assist in overcoming these difficulties by raising students' achievements and lowering syntax errors, which are a major barrier for new learners. The scores are influenced by the quantity of errors; hence, marking is a process that involves quantifying the errors while maintaining consistent exercise durations for both the experimental and control groups. Similar IDEs, such as "Flowgorithm" and "Flowrun", were eventually created to translate the flowchart into code in order to address the transfer challenges, demonstrating the demand for such tools. Microsoft's well-recognized programming IDE, Visual Studio, now includes snippets, code creation, and reverse engineering. Nevertheless, empirical evidence suggests that inexperienced first-year students lacking prior programming knowledge or skill may have difficulties when endeavoring to accomplish tasks utilizing this particular software [44]. The The evaluation of IDE Visual Studio usability revealed a need for further improvement for novices, as it takes a lot of time and the interface is a bit tricky for them. This finding gives extra credit to the CBT as a simple, easy-to-use tool. However, there aren't any assessments of how well the students performed using these supplementary resources in the literature that may be considered as a suggestion for further study. The purpose of this empirical study was to evaluate the students' programming skills while using the CBT. It was discovered to be really helpful in raising students' performance.

The CBT application has received favorable student response, which is consistent with the literature's claim that fewer syntax errors can make learning programming simpler and more pleasant. This shows that the CBT environment can be used as a more convenient and beneficial substitute for traditional Integrated Development Environments (IDEs), which frequently issue syntax error warnings and compiler feedback that can be confusing to inexperienced users.

Additionally, our research adds to the corpus of knowledge on instructional strategies for programming education. While the use of technologies like Augmented Reality (AR) and mixed learning has been examined in the literature in various ways, our study introduces the CBT application as a further useful tool. The CBT application significantly improved student interest and performance, which implies that it can improve the learning process in programming education. Students expressed interest in and appreciation for the tool in addition to the performance enhancer. Comments from students back up the Kirkpatrick model's four levels. Online teaching during the COVID-19 epidemic, where every computer had a microphone available for usage, was shown to benefit from the added feature of generating templates through the optional audio commands. The template-based IDEs were consequently taken into account for our department's teaching method.

CBT wasn't designed to be commercialized; it was made for our students to utilize in the Windows lab and for this project. As a result, the application is only able to assist with code creation because it lacks an internal compiler. Due to this limitation, the IDE is an experimental environment, but it is suitable for the task.

The study presents intriguing similarities and contrasts when comparing our findings on the effectiveness of the CBT approach in improving the programming skills of novice learners with the literature. Previous research has underscored the significance of problem-solving skills in the field of programming, as evidenced by studies such as [12] and [13]. Our study contributes to this body of knowledge by examining a practical tool that specifically addresses the issue of syntax errors, a well-documented obstacle for novice programmers. This assertion not only corresponds with but also expands upon the theoretical comprehension of difficulties encountered in programming education, indicating that the CBT tool can have a crucial impact in tackling these obstacles. In contrast to conventional IDEs like Visual Studio, which have been seen to be overwhelming for first-year students [44], CBT presents a more accessible and user-friendly interface, suggesting a possible transformation in the approach to teaching beginning programming.

The results of this study suggest the need for a reassessment of the conventional pedagogical methods employed in programming instruction, particularly for individuals who are new to the subject. The efficacy and straightforwardness of CBT underscore the necessity for educational resources that better correspond to the learners' initial proficiency levels. Further investigation is required to examine the enduring effects of utilizing these simplified tools on students' proficiency in programming and their ability to adapt to more sophisticated programming principles in the long run. Moreover, conducting comparative studies that evaluate the efficacy of CBT in comparison to both classic and current IDEs across many educational settings would yield more profound understandings. This research has the potential to make a substantial contribution to the current endeavors aimed at mitigating dropout rates in programming courses and improving overall learning results.

VI. Conclusions

The use of templates by the students proved its effectiveness in the development of simple programs by the new learners. It guarantees the correct constructs that the students can integrate when they translate the logic that could be drawn as a flowchart into a clean, correct code, therefore enhancing the performance and consequently increasing the confidence and interest while reducing the number of failures and dropouts. It also helps in making the switch between different programming languages an easy job in academia and in the marketplace. The Code By Template (CBT) application was developed by the author as a student support tool. To the best of the writer' s knowledge, it is one of the first studies looking at how such a development environment could enhance student achievement and acceptance. As a teaching tool, CBT is not claimed to be complete, but it proves the concept of getting over simple mistakes in code writing and boosts the students' performance. Accordingly, we encourage the vendors to consider adding code-writing support functions to the current programming environments. The CBT can be used to produce clear and accurate code, just like any other calculator used to solve mathematical tasks. The tool can be used by the students as a scaffold for as long as necessary until they are comfortable writing their own correct codes. Such assistance lowers programming-related anxiety and, ideally, lowers the dropout rate.

Appendix

Table A. The questions and their thematic classifications

Question	Theme
I enjoyed the Code By Template (CBT) application	Enjoyment
CBT supports the navigation through the code	Navigability
CBT reduces the code writing complexity	Complexity
CBT provides a reliable and robust programming environment	Reliability
CBT increased my confidence in code writing	Confidence
I intend to continue using the CBT application	Continuity
CBT supports the understanding of the links between the different constructs	Understanding
I believe that CBT empowered my learning	Empowerment
I found CBT application easy to use	Ease of use
I found the CBT application useful for learning	Usefulness
I believe that I am capable of learning on my own through the CBT app	Learnability
Using CBT application would improve my course performance	Performance
CBT application supports practical teaching and learning	Practicability
CBT application increased my interest in the subject	Interest
I prefer using CBT application instead of the direct code writing	Preference
Learning by CBT answers the questions that I sometimes feel embarrassed to ask in the class	Embarrassment
I prefer using CBT application through, voice, buttons, both, or none	Usage through

Conflict of Interest

The author declares no conflict of interest.

References

[1] R. M. Siegfried, D. Liporace, and K. G. Herbert-Berger, "What can the reid list of first programming languages teach us about teaching CS1?" in *Proc. the 50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 1256–1257. https://doi.org/10.1145/3287324.3293830

[2] M. Kesselbacher, "Supporting the acquisition of programming skills with program construction patterns," in *Proc. 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion, ICSE-Companion* 2019, pp. 188–189. https://doi.org/10.1109/ICSE-Companion.2019.00077

[3] S. Y. Al-Imamy, "Computer programming course for non-mis business students: Curriculum, perception and enrichment," *IOSR Journal of Business and Management*, vol. 19, no. 03, pp. 87–95, 2017. https://doi.org/10.9790/487X-1903018795

[4] C. E. Morton, S. F. Smith, T. Lwin, M. George, and M. Williams, "Computer programming: should medical students be learning it?" *JMIR Medical Education*, vol. 5, no. 1, 11940, 2019. https://doi.org/10.2196/11940

[5] A. Azad and D. T. Smith, "Teaching an introductory programming language in a general education course," *Journal of Information Technology Education. Innovations in Practice*, vol. 13, no. 57, 2014. https://doi.org/10.28945/1992

[6] A. Yadin, "Reducing the dropout rate in an introductory programming course," *ACM Inroads*, vol. 2, no. 4, pp. 71–76, 2011. https://doi.org/10.1145/2038876.2038894

[7] W. M. Kunkle and R. B. Allen, "The impact of different teaching approaches and languages on student learning of introductory programming concepts," *ACM Transactions on Computing Education (TOCE)*, vol. 16, no. 1, pp. 1–26, 2016. https://doi.org/10.1145/2785807

[8] T. Koulouri, S. Lauria, and R. D. Macredie, "Teaching introductory programming: A quantitative evaluation of different approaches," *ACM Transactions on Computing Education (TOCE)*, vol. 14, no. 4, pp. 1–28, 2014. https://doi.org/10.1145/2662412

[9] C. Y. Tsai and Y. C. Lai, "Design and validation of an augmented reality teaching system for primary logic programming education," *Sensors*, vol. 22, no. 1, 2022. https://doi.org/10.3390/s22010389

[10] M. Figueiredo, M. A. Cifredo-Chacon, and V. Goncalves, "Learning programming and electronics with augmented reality," in *proc. International Conference on Universal Access in Human-Computer Interaction*, 2016, Springer, Cham, pp. 57–64. https://doi.org/10.1007/978-3-319-40238-3_6

[11] I. Karvelas, "Investigating novice programmers' interaction with programming environments," *Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, pp. 336–337, 2019. https://doi.org/10.1145/3304221.3325596

[12] A. K. Veerasamy, D. D'Souza, R. Linden, and M. J. Laakso, "Relationship between perceived problem-solving skills and academic performance of novice learners in introductory programming courses," *Journal of Computer Assisted Learning*, vol. 35, no. 2, pp. 246–255. 2019. https://doi.org/10.1111/jcal.12326

[13] Malik, S. Iqbal, R. Mathew, A. Al-Sideiri, J. Jabbar, R. Al-Nuaimi, and R. M. Tawafak, "Enhancing problem-solving skills of novice programmers in an introductory programming course," *Computer Applications in Engineering Education*, vol. 30, no. 1, pp. 174–194, 2022, https://doi.org/10.1002/cae.22450

[14] H. J. B. Rocha, P. C. D. A. R. Tedesco, E. D. B. Costa, "On the use of feedback in learning computer programming by novices: a systematic literature mapping," *Informatics in Education*, 2022. https://doi.org/10.15388/infedu

[15] S. Y. Al-Imamy, "Blending printed texts with digital resources through augmented reality interaction," *Education and Information Technologies*, vol. 25, no. 4, pp. 2561–2576, 2020. https://doi.org/10.1007/s10639-019-10070-w

[16] S. Al-Imamy, J. Alizadeh, and M. A. Nour, "On the development of a programming teaching tool: The effect of teaching by templates on the learning process," *Journal of Information Technology Education: Research*, vol. 5, no. 1, pp. 271–283, 2006. https://doi.org/10.28945/247

[17] M. A. Nour, S. Al-Imamy, and J. Alizadeh, "An empirical investigation of student perceptions of the effect of a blended learning environment on the learning outcomes," *International Journal of Excellence in e-Solutions for Management*, vol. 1, no. 2, pp. 27–39, 2007.

[18] A. Altadmri and N. C. C. Brown, "37 million compilations: Investigating novice programming mistakes in large-scale student data," in *Proc. the 46th ACM Technical Symposium on Computer Science Education*, PP. 522–527, 2015. https://doi.org/10.1145/2676723.2677258

[19] Z. Xu, A. D. Ritzhaupt, F. Tian, and K. Umapathy, "Block-based versus text-based programming environments on novice student learning outcomes: a meta-analysis study," *Computer Science Education*, vol. 29, no. 2, pp. 177–204, 2019. https://doi.org/10.1080/08993408.2019.1565233

[20] C. Watson, and F. W. Li, "Failure rates in introductory programming revisited," in *Proc. the 2014 conference on Innovation & technology in computer science education*, pp. 39–44, 2014. https://doi.org/10.1145/2591708.2591749

[21] J. Bennedsen, and M. E. Caspersen, "Failure rates in introductory programming," *AcM SIGcSE Bulletin*, vol. 39, no. 2, pp. 32–36, 2007. https://doi.org/10.1145/1272848.1272879

[22] A. A. R. H. Ansari and D. R. Vora, "NLI-GSC : A natural language interface for generating SourceCode," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 1, 2022. https://doi.org/10.14569/IJACSA.2022.0130198

[23] T. Barik, "Error messages as rational reconstructions". *North Carolina State University ProQuest Dissertations Publishing*, 2018.

[24] C. Kelleher, and R. Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," *ACM Computing Surveys (CSUR)*, vol. 37, no. 2, pp. 83–137, 2005. https://doi.org/10.1145/1089733.1089734

[25] M. Idrees, F. Aslam, K. Shahzad, and S. M. Sarwar, "Towards a universal framework for visual programming languages," *Pak. J. Engg. Appl. Sci.*, vol. 23, pp. 55–65, 2018.

[26] A. Repenning, "Moving beyond syntax: Lessons from 20 years of blocks programing in agentsheets," *Journal of Visual Languages and Sentient Systems*, vol. 3, no. 1, pp. 68–91, 2017. https://doi.org/10.18293/VLSS2017-010

[27] N. Bubica and I. Boljat, "Teaching of novice programmers: Strategie, programming languages and predictors," in *Proc. V International Conference of Information Technology and Development of Education, Lanuary*, 2014.

[28] T. Daly, "Minimizing to maximize: An initial attempt at teaching introductory programming using Alice," *Journal of Computing Sciences in Colleges*, vol. 26, no. 5, pp. 23–30, 2011.

[29] I. Utting, S. Cooper, M. Kolling, J. Maloney, and M. Resnick, "Alice, greenfoot, and scratch--a discussion," *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, pp. 1–11, 2010. https://doi.org/10.1145/1868358.1868364

[30] U. AbdulSamad and R. Romli, "A comparison of block-based programming platforms for learning programming and creating simple application," in *Proc. International Conference of Reliable Information and Communication Technology*, 2022, pp. 630–640. https://doi.org/10.1007/978-3-030-98741-1_52

[31] J. J. Thomas, V. Suresh, M. Anas, S. Sajeev, and K. S. Sunil, "Programming with natural languages: A survey," in *Proc. Computer Networks and Inventive Communication Technologies*, 2022, Singapore, pp. 767–779. https://doi.org/10.1007/978-981-16-3728-5_57

[32] S. Lau, and P. Guo, "From 'Ban it till we understand it' to 'Resistance is futile': How university programming instructors plan to adapt as more students use AI code generation and explanation tools such as ChatGPT and GitHub Copilot," in *Proc. the 2023 ACM Conference on International Computing Education Research-Volume 1*, 2023, pp. 106–121. https://dx.doi.org/10.1145/3568813.3600138

[33] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, and T. Wilusz, "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students," in *Proc. Conference on Innovation and technology in computer science education*, 2001, pp. 125–180. https://doi.org/10.1145/572133.572137

[34] R. McCartney, J. Boustedt, A. Eckerdal, K. Sanders, and C. Zander, "Can first-year students program yet? A study revisited," in *Proc. the Ninth Annual International ACM Conference on International Computing Education Research*, 2013, pp. 91–98. https://doi.org/10.1145/2493394.2493412

[35] I. Utting, A. E. Tew, M. McCracken, L. Thomas, D. Bouvier, and R. Frye, "A fresh look at novice programmers' performance and their teachers' expectations," in *Proc. Conference on Innovation and Technology in Computer Science Education*, 2013, pp. 15–32. https://doi.org/10.1145/2543882.2543884

[36] B. A. Becker, "An effective approach to enhancing compiler error messages," in *Proc. the 47th ACM Technical Symposium on Computing Science Education*, 2016, pp. 126–131. https://doi.org/10.1145/2839509.2844584

[37] B. A. Becker, K. Goslin, and G. Glanville, "The effects of enhanced compiler error messages on a syntax error debugging test," in *Proc. the 49th ACM Technical Symposium on Computer Science Education*, 2018, pp. 640–645. https://doi.org/10.1145/3159450.3159461

[38] V. J. Traver, "On compiler error messages: what they say and what they mean," *Advances in Human-Computer Interaction*, 2010, https://doi.org/10.1155/2010/602570

[39] P. Bouna, "Visual studio code for C developers," *Packt Publishing*, 2022.

[40] S. Gul, M. Asif, W. Ahmad, and U. Ahmad, "Teaching programming: A mind map based methodology to improve learning outcomes," in *Proc. 2017 International Conference on Information and Communication Technologies (ICICT)*, 2017, pp. 209–213. https://doi.org/10.1109/ICICT.2017.8320192

[41] M. Mladenovic, I. Boljat, and Z. Zanko, "Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level," *Education and Information Technologies*, vol. 23, no. 4, pp. 1483–1500, 2018. https://doi.org/10.1007/s10639-017-9673-3

[42] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, "Habits of programming in scratch," *in Proc. the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, 2011, pp. 168–172. https://doi.org/10.1145/1999747.1999796

[43] D. Kirkpatrick, "Evaluating training program," *Alexandria, VA: American Society for Training and Development*, 1975.

[44] A. Muhammad, and I. Ashraf, "A survey on evaluating usability of visual studio," *Pakistan Journal of Engineering and Technology*, vol. 5, no. 1, pp. 23–28, 2022.