Education Tech Research Dev
https://doi.org/10.1007/s11423-023-10328-8

DEVELOPMENT ARTICLE

Block-based versus text-based programming: a comparison of learners' programming behaviors, computational thinking skills and attitudes toward programming

Dan Sun[1] · Chee-Kit Looi[2] · Yan Li[3] · Chengcong Zhu[4] · Caifeng Zhu[5] · Miaoting Cheng[6]

Accepted: 10 October 2023

Abstract
In the current era where computational literacy holds significant relevance, a growing number of schools across the globe have placed emphasis on K-12 programming education. This field of education primarily comprises two distinct modalities—the block-based programming modality (BPM) and the text-based programming modality (TPM). Previous research may not have provided a complete understanding of the differences between these two modalities as it did not take into account both the learning process and learning outcomes. This study aimed to compare secondary students' programming behaviors, computational thinking skills, and attitudes toward programming between the two modalities through a quasi-experimental design in a Chinese secondary school. The findings showed that (1) learners in TPM encountered more syntactical errors and spent more time between two clicks of debugging, while learners in BPM had more code-changing behaviors by adjusting programming blocks, made more attempts of debugging, and had more irrelevant behaviors; (2) learners in BPM achieved a higher level of computational thinking skills; (3) learners in both modalities experienced a slight decrease in confidence and enjoyment, while learners in BPM had higher interest levels in programming. (4) Code Changer, Minimal Debugger, Maximal Debugger, Distracted Coder and Average Coder were identified through students' programming behavior in the two programming modalities, and differences in their CT skills and attitudinal data were revealed. Lastly, pedagogical implications based on the findings are also discussed.

Keywords Computational thinking · Text-based programming modality · Block-based programming modality · Programming behaviors · Instructional strategies

Introduction

Computational thinking (CT) is an essential twenty-first-century competency inspired by computer science practice and should be a part of all K-12 students' analytic toolkits (Ma et al., 2021; Wing, 2014). To foster learners' CT, many countries (e.g., China, the United States, and the United Kingdom) have integrated computer programming into their K-12

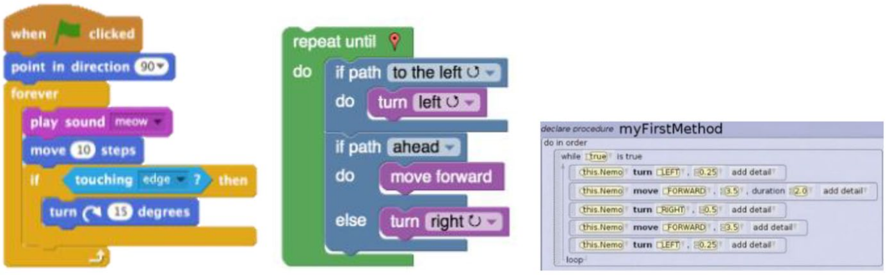
Extended author information available on the last page of the article curricula (Angeli et al., 2016; Bey et al., 2019; Jocius et al., 2021). Block-based programming modality (BPM) and text-based programming modality (TPM) are two major instructional modalities that have been widely used in K-12 programming education (Weintrop & Wilensky, 2019). BPM could provide visual cues to denote how and where to use a given command (Tempel, 2013), which makes computer science accessible and easy-to-understand to novice learners (Bau et al., 2017). Conversely, being a more conventional way, TPM requires learners to have the ability to write codes in various text-based programming languages (e.g., Python and Java), which would be useful in carrying out professional programming projects and pursuing computer science careers (HelloGitHub, 2023).

While previous research has explored the differences in programming performance between TPM and BPM in terms of their effectiveness in supporting student learning (Weintrop & Wilensky, 2017) and the prerequisite knowledge and skills required to use each modality (Duncan et al., 2014), there is still a research gap in investigating how the learning process affects learning performance in the context of using TPM and BPM. There is a need for a fine-grained investigation of the differences between TPM and BPM that takes into account both the learning process and learning performance (Grover, 2021; Scherer et al., 2020; Weintrop & Wilensky, 2019). Given the importance of CT and computer programming to K-12 curricula, it is important to understand the relative strengths and weaknesses of each modality and how they can be used to support student learning effectively. Our research aimed to fill this gap by comparing the effectiveness of TPM and BPM in a Chinese secondary school using a quasi-experimental design. We examined learners' programming skills, CT skills, and attitudes toward programming in both modalities to determine which modality was more effective in promoting students' programming practice. Our findings provide insights for educators on how to design more effective instructional designs to support students' programming skill acquisition and promote their CT development. By comparing the two modalities, we hope to inform educational practice to better prepare students for future programming challenges and careers.

Literature review

Block-based and text-based programming modalities

As the Turing Award winner, referred to as the "Nobel Prize of Computing", Dijkstra (1982) argued, the tools we used had a far-reaching influence on transforming our thinking habits and abilities. Echoing this idea, programming modality might have an important impact on changing learners' thinking, which is a critical factor that instructors should consider in computer programming education. As two major programming modalities, BPM and TPM have been widely integrated into K-12 education. BPM offers a visual programming approach that utilizes a "programming-primitive-as-puzzle-piece metaphor" to design computer programming (Bau et al., 2017). The visual illustration of a block presents available commands in logically-organized drawers, denotes how and where a given command can be used and prevents learners from making syntax errors during programming (Tempel, 2013; Weintrop & Wilensky, 2015). Due to these easy-to-use attributes, BPM makes computer science accessible and easy-to-understand to novice learners (Grover & Basu, 2017). Several BPM environments (see Fig. 1) are designed to lower the barrier to learning programming by eliminating troublesome issues of syntax (Grover, 2021), including *Scratch* (an event-driven block-based programming language and online community developed by

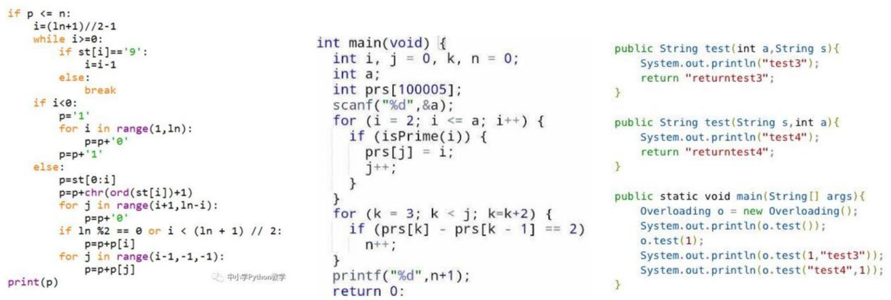

when clicked
point in direction 90
forever
play sound meow
move 10 steps
if touching edge ? then
turn 15 degrees
repeat until
do
if path to the left
do
turn left
if path ahead
do
move forward
else
turn right
declare procedure myFirstMethod
do in order
while true is true
this.Nemo turn LEFT , 0.25 add detail
this.Nemo move FORWARD , 3.5 , duration 2.0 add detail
this.Nemo turn RIGHT , 0.5 add detail
this.Nemo move FORWARD , 3.5 add detail
this.Nemo turn LEFT , 0.25 add detail
loop

(a) Scratch (b) Blockly (c) Alice

Fig. 1 Three typical block-based programming modalities

Massachusetts Institute of Technology, https://scratch.mit.edu/), *Blockly* (a web-based and block-based programming editor developed by Google, https://developers.google.com/blockly), and *Alice* (a block-based programming environment developed by Carnegie Mellon University, https://www.alice.org/). BPM has been extensively used in introductory computer science classes across K-12 education (Kölling et al., 2015).

As another widely-used modality in K-12 programming education, TPM is integrated especially for learners who can understand the syntax and logic of text-based codes (Yucer & Rizvanoglu, 2019). Figure 2 shows three typical text-based programming languages, including *Python* (a dynamic and interpreted language designed by Guido van Rossum, https://www.python.org/), *C* (a general-purpose programming language created by Dennis Ritchie), and *Java* (a class-based and object-oriented programming language developed by James Gosling, https://www.java.com/). Among these languages, Python is the official programming language in selective Information Technology Curricula for Chinese secondary and high schools (Ministry of Education, 2017). Compared with BPM, TPM is perceived by learners as a more authentic and powerful tool in learning programming (Weintrop & Wilensky, 2015). TPM enables learners to proceed to a higher level of programming expertise and provides opportunities for them to participate in professional programming projects (Weintrop & Wilensky, 2019). Hence, TPM is usually designed as the consequent step for learners after they are versed in BPM (Armoni et al., 2015), which cannot be substituted by BPM. However, learners usually encounter syntax errors, feel frustrated when stuck on these errors, and drop out easily in the higher-level programming courses in TPM


```python
if p <= n:
    i=(ln+1)//2-1
    while i>=0:
        if st[i]=='9':
            i=i-1
        else:
            break
    if i<0:
        p='1'
        for i in range(1,ln):
            p=p+'0'
        p=p+'1'
    else:
        p=st[0:i]
        p=p+chr(ord(st[i])+1)
        for j in range(i+1,ln-i):
            p=p+'0'
        if ln %2 == 0 or i < (ln + 1) // 2:
            p=p+p[i]
        for j in range(i-1,-1,-1):
            p=p+p[j]
print(p)
```

```c
int main(void) {
  int i, j = 0, k, n = 0;
  int a;
  int prs[100005];
  scanf("%d",&a);
  for (i = 2; i <= a; i++) {
    if (isPrime(i)) {
      prs[j] = i;
      j++;
    }
  }
  for (k = 3; k < j; k=k+2) {
    if (prs[k] - prs[k - 1] == 2)
      n++;
  }
  printf("%d",n+1);
  return 0:
```

```java
public String test(int a,String s){
    System.out.println("test3");
    return "returntest3";
}

public String test(String s,int a){
    System.out.println("test4");
    return "returntest4";
}

public static void main(String[] args){
    Overloading o = new Overloading();
    System.out.println(o.test());
    o.test(1);
    System.out.println(o.test(1,"test3"));
    System.out.println(o.test("test4",1));
}
```

(a) Python (b) C (c) Java

Fig. 2 Three typical text-based programming modalities

(Falloon, 2016). Thus, various instructional strategies have been used to alleviate frustrations and difficulties they may encounter (Taub et al., 2012), facilitate learners' text-based programming practice (Sun et al., 2021), and promote their interests and motivations to learn with TPM (Alshaigy et al., 2015).

The research community has exhibited a growing interest in exploring the differences and linkages between text-based programming (TPM) and block-based programming (BPM). While BPM has been considered an effective way to introduce foundational programming concepts and foster interest and motivation in novice learners (García et al., 2015; Howland & Good, 2014; Wilson & Moffat, 2010), it may pose challenges for learners in managing complex programming projects or applying programming knowledge to solve real-world problems (Duncan et al., 2014). TPM, on the other hand, is an essential skill for future professional programming careers (TIOBE, 2023), but novice learners in TPM may struggle with grammar and require more advanced programming knowledge and skills (Armoni et al., 2015; Mladenović et al., 2018; Price & Barnes, 2015). Our study aimed to compare the learning processes and outcomes in BPM and TPM for novice learners. By focusing specifically on the differences between the two programming modalities, our study aimed to provide empirical evidence for educators and scholars to better understand the strengths and weaknesses of each modality and how they can be used to support novice learners in developing programming skills. Through this comparison, we hoped to provide insights for educators to design more effective and efficient instructional designs to support novices' programming skill acquisition.

Computational thinking and programming

CT is a kind of analytical thinking that makes use of the common points with mathematical thinking, engineering, and scientific thinking (Wing, 2008), which could be fostered through divergent ways and contexts (i.e., participating in unplugged CS games (Taub et al., 2012), applying strategies to acquire information for language learning (Mannila et al., 2014), joining STEM practices (Jocius et al., 2021)). Among different strategies, programming has been proven to be one of the effective ways to improve learners' CT skills (Brennan & Resnick, 2012; Jiang et al., 2021; Pellas & Vosinakis, 2018). According to Tsai et al. (2021), previous literature on CT can be summarized using domain-specific and domain-general definitions. The domain-specific category indicates the domain-specific knowledge or skills that are required to systematically solve the problems in the subject domain of computer science or computer programming, researchers usually depend on a specific popular programming language, so *Dr. Scratch* (Garneli & Chorianopoulos, 2018), *Bebras test* (Rojas-López & García-Peñalvo, 2018), the *Computational thinking test* (CTt; Román-González et al., 2017) were widely used as the assessment tool.

While in the domain-general definition, researchers defined CT as the competencies required for solving problems systematically in humans' daily lives and all learning domains, so CT covered a set of thinking skills of creativity, algorithmic thinking, critical thinking, problem solving, establishing communication and cooperation (ISTE, 2015; Mannila et al., 2014; Riley & Hunt, 2014). Wing (2011) defined CT as a thought process to effectively and efficiently deal with the problem, many researchers (Doleck et al., 2017; Katai, 2015) have even viewed CT as an integrated ability including algorithmic thinking, social cooperative capacities, creative thinking, and critical thinking. It is obvious that the CT definition goes beyond problem solving contexts, and may need to be regulated by higher-level metacognitive skills (Allsop, 2019). Drawing on ISTE

(2015) and a range of research studies, Korkmaz et al. (2017) synthesized a definition of CT as the capacity to use an algorithmic approach to address challenges, while also fostering communication and collaboration in a team-based setting. This process calls for utilizing innovative strategies to develop solutions that can tackle complex issues. The authors also identified five key elements of CT: algorithmic thinking, problem-solving, creativity, critical thinking, and cooperativity. Furthermore, Korkmaz et al. (2017) also developed and validated a CT scale for assessing students' CT skills. This instrument has been widely utilized by various researchers to evaluate students' CT proficiency (Durak & Saritepeci, 2018; Ma et al., 2021; Pellas & Vosinakis, 2018).

Multiple analysis of the programming process

Prior empirical research has utilized multiple analytical methods to explore varied perspectives of learners' programming processes. Wu et al. (2019) used a quantitative ethnography approach to analyze the collaborative programming between a high-performing and a low-performing team. Pereira et al. (2020) conducted a clustering analysis based on the students' logs to inspect the patterns of programming behaviors in each student cluster and explored how these behaviors reflect on the evaluative factors (effective or ineffective behaviors). Sun et al. (2021) applied mixed methods, including click stream analysis and lag-sequential analysis, to analyze three contrasting pairs' collaborative programming behaviors, discourses, and perceptions. As a complementary, the traditional, summative assessment (e.g., final tests, attitudinal test) can help reveal learners' direct performances of computer programming knowledge or attitude. Those studies indicated that multiple analytical methods could be used to conduct the multidimensional analysis, which is beneficial to demonstrate varied dimensions of programming learning.

As for the effectiveness of different programming modalities, many previous studies measured learners' mastery of knowledge and skills or learners' attitudinal changes, rather than learners' higher-order thinking (Nolan & Bergin, 2016; Noone & Mooney, 2018). For example, in the quasi-experiment reported by Eid and Millham (2012), learners started with console-based procedural programming (e.g., COBOL) to gain a strong understanding of basic programming concepts such as procedures, variables, and loops without needing to learn the added complexities of a visual interface. The study by Weintrop (2015) revealed that high school learners in BPM achieved higher scores on knowledge performance and reported higher levels of confidence, enjoyment, and interest. Weintrop and Wilensky (2017) designed a quasi-experimental study and found that high school learners in BPM achieved greater learning gains in programming concepts (e.g., variables, loops, conditionals, and functions) than in TPM in an introductory programming class.

Taken together, there is a current research trend to understand the difference in block-based and text-based programming modalities in K12 formal education to better serve secondary school students' programming learning and improve their engagement in the computer science field. In addition, following the analytical trend, this research collects multi-modal data to analyze students' programming behaviors, CT skills, and attitudes toward programming in the two programming modalities.

Methodology

Research purpose and questions

This research aimed to gain a deeper understanding of how learning occurs in each modality and to what extent each programming modality fosters their CT skills and their positive attitudes toward programming. This study conducted a quasi-experiment design in a secondary school. Multi-modal data (including learning platform log files, learners' CT skills and their attitudinal data) were collected and different analytics approaches (i.e., statistical analysis, clustering analysis) were used to compare learners' programming behaviors, CT skills, and attitudes toward programming in TPM and BPM. Based on the results, this research will also seek to distill some pedagogical implications for future instructional design and empirical computer programming research. The specific research questions (RQ) are:

RQ1: What were the differences in learners' behaviors in learning via TPM versus BPM?
RO2: What were the differences in learners' CT skills in learning via TPM versus BPM?
RO3: What were the differences in learners' perceived attitudes towards programming in learning via TPM versus BPM?

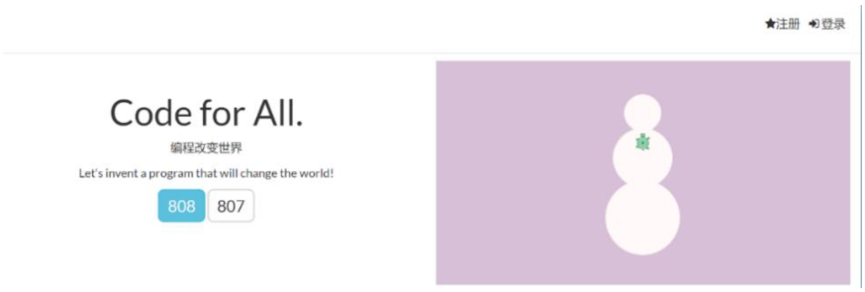
Educational context and participants

The research was conducted in a compulsory course titled "Information Technology," which was carried out in a Chinese secondary school during the autumn of 2020. A quasi-experimental design was utilized to investigate the differences in learners' programming behaviors, CT skills, and attitudes toward programming in TPM and BPM. There were 32 learners (13 female and 19 male) in the TPM class and 32 learners (15 female and 17 male) in the BPM class. Students were around 13 years old, and most of them did not have programming experience in formal education. Classes were taught by the same instructor, who maintained a similar teaching style under two modalities, offered the same instructional materials to learners, and used the same teaching guidance for each class, except the materials were presented via TPM or BPM.

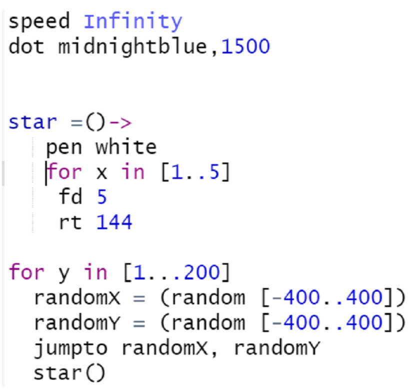
Instructional procedures

The instructor designed three phases and six instructional sessions in this course, with each session lasting 45 min. In Phase I (the first session), the instructor introduced basic concepts of programming to learners and illustrated the TPM or BPM to each class. In Phase II (the second to fifth sessions), the instructor required learners to practice programming, including sequential, selective, looping, and function structures in the text-based or block-based programming environment. In Phase III (the sixth session), the instructor asked learners to complete a series of programming projects. The series of programming projects tested learners' basic programming knowledge taught in Phase II. During the instruction and learning processes, the instructor taught concepts, algorithms, and coding operations with text-based programming in the TPM class and with block-based programming in the BPM class.

Code4all (see Fig. 3a) was utilized as the programming platform which supports both TPM and BPM. It is an online programming environment developed from Pencil Code, which allows users to freely move back and forth between text-based and block-based versions of their programs. The two programming modalities are isomorphic, which means anything that can be done in one interface can also be done in the other. Unlike Pencil Code, Code4all prevents learners from moving between the two modalities. Instead, learners either use the block-based interface or the text-based interface. Thus, for the duration of this study, learners were introduced to programming using either a text-based version of Code4all (see Fig. 3b) or a block-based version of Code-4all (see Fig. 3c). This means learners in the BPM class programmed via the drag-and-drop mechanism supported by the block-based interface, while the TPM class authored programs by typing in commands character-by-character. Aside from the programming modality, everything else about the two versions of the programming environment is identical, including the programming language (including keywords and syntax), the visual execution environment, and the programming capabilities and other environmental scaffolds. For both versions of Code4all, the underlying programming language was



(a) Main page of Code4all

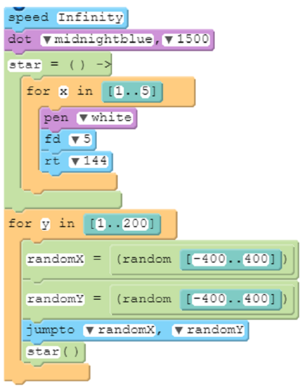


```
speed Infinity
dot midnightblue,1500


star =()->
  pen white
  for x in [1..5]
    fd 5
    rt 144

for y in [1...200]
  randomX = (random [-400..400])
  randomY = (random [-400..400])
  jumpto randomX, randomY
  star()
```

(b) TPM interface



```
speed Infinity
dot midnightblue, 1500
star = () ->
  for x in [1..5]
    pen white
    fd 5
    rt 144
for y in [1..200]
  randomX = (random [-400..400])
  randomY = (random [-400..400])
  jumpto randomX, randomY
  star()
```

(c) BMP interface

Fig. 3 The Code4all programming platform and two programming interfaces used in this study

CoffeeScript. CoffeeScript was chosen because it is syntactically light and has an active professional user base (Weintrop & Wilensky, 2019).

Data collection

To capture learners' programming learning performance, we collected multi-modal data, including platform logs, students' CT skills, and attitudinal data (see Table 1). Secondly, we collected data about learners' CT skills before and after the intervention, and learners in TPM and BPM took the same test. The test instrument for CT skills was adapted from the CT scale (CTS) developed by Korkmaz et al. (2017). This 5-point Likert scale contains five factors (creativity, algorithm thinking, cooperativity, critical thinking and problem solving) and 22 test items (see Table 1). Notably, the CTS was validated and applied among Chinese K-12 learners by Bai and Gu (2019).

Thirdly, we collected learners' self-reported pre- and post-survey about their attitudinal changes (see Table 2) during the study, and the survey was comprised of 10-point Likert scale questions. The contents in the pre- and post-surveys were largely the same except for the item related to changes in tense (past/future). The survey was based on items from the Attitudinal Survey that have been validated and widely used in computing education research studies (Weintrop & Wilensky, 2019).

Data analysis

We used multiple analytical approaches, such as statistical and clustering analysis, to examine the impact of learners' programming learning quality between TPM and BPM (see Table 3).

Firstly, according to extracted variables and previous studies (Pereira et al., 2020), this study identified five programming behaviors: *Average number of code-changes* (AnC), *Number of Irrelevant behaviors* (NoIB), *Number of debugs* (NoD), *Average time between two debugs* (AtD), and *Number of errors* (NoE) (see Table 4). It should be noted that, despite the learner's coding in different programming modalities, the codes recorded by the Code4all platform were in a text-based format. Descriptive statistics were used to provide an overall view of distribution of programming behaviors between TPM and BPM.

Additionally, clustering algorithms can reveal hidden patterns in complex datasets. In many prior studies (Dutt et al., 2015; Shi & Cristea, 2018; Shi et al., 2019), unsupervised learning methods were used to analyze novel relationships of educational variables. As learners' behaviors were heterogeneous, we clustered them based on their logs and examined patterns of their programming behaviors within each student cluster. To accomplish this, we utilized the widely-used k-means algorithm (MacQueen, 1967), determine optimal number of clusters, select variables, standardize data and randomly select initial cluster centers. Assign each point to the closest center, calculate new centers, and repeat until convergence. Evaluate results and interpret cluster characteristics based on research question and hypotheses. This method is effective for identifying cluster centroids and analyzing the distance between each pair of clusters, which enabled us to use the mean silhouette coefficient (Rousseeuw, 1987) for selecting the most appropriate number of clusters for our data. RapidMiner and Python language were used to conduct clustering analysis.

Secondly, we conducted a T-test to examine the difference in CT skills between the TPM and BPM groups. Furthermore, we utilized Analysis of Variance (ANOVA) to explore the difference in learners' CT skills among different clusters.

Table 1 Sample items in scale of CT skills

CTS	*N*	Two items of the scale
Creativity	4	I have a belief that I can solve the problems possible to occur when I encounter a new situation
		I trust my intuitions and feelings of “trueness” and “wrongness” when I approach the solution to a problem
Algorithm thinking	4	I can immediately establish the equity that will give the solution to a problem
		I can mathematically express the solutions ways of the problems I face in daily life
Cooperativity	4	In cooperative learning, I think that I attain/will attain more successful results because I am working in a group
		I like solving problems related to a group project together with my friends in cooperative learning
Critical thinking	4	I am good at preparing regular plans regarding the solution of complex problems
		I am willing to learn challenging things
Problem solving	6	I have problems in demonstrating the solution to a problem in my mind
		I cannot produce so many options while thinking of the ways of the possible solution to a problem

Table 2 Items in the survey of attitudes toward programming

Attitude	*N*	Items
Enjoyment of programming	3	Programming is Fun
		I like programming
		I am excited about this course
Confidence in programming ability	2	I am good at programming
		I will perform/performed well in this course
Interest in future CS	2	I will find a job related to programming in the future
		I will take more programming courses after this course

Table 3 Research analytical framework

Data source	Analytical method	Research question
Code4all log files	Descriptive statistics/Clustering analysis (CA)	RQ 1
CT skills scale	*T*-test/ANOVA with Bonferroni correction	RQ 2
Attitudinal Survey	Descriptive statistics/Wilcoxon Rank- Sum test/ANOVA with Bonferroni correction	RQ 3

Table 4 Definition of different behaviors during programming

Behavior	Definition
Average number of code-changes (AnC)	The average number of line changes within the code between two clicks on debug button
Number of irrelevant behaviors (NoIB)	Irrelevant behaviors refer to actions or behaviors that do not contribute to the completion of the programming task, such as browsing the internet, talking with classmates about non-task-related topics, or working on unrelated assignments
Number of debugs (NoD)	The number of clicks on debug button made by a student
Average time between two debugs (AtD)	The average time spent between two clicks on debug button
Number of errors (NoE)	The number of debugging that have errors (i.e., syntactical errors)

Thirdly, since the samples were independent and the underlying data is ordinal and non-parametric (Fay & Proschan, 2010), we performed a Wilcoxon Rank Sum test (reported as a U statistic) to compare the two modalities. Additionally, we used Analysis of Variance (ANOVA) to investigate the difference in learners' attitudes toward programming among different clusters. Taken together, mixed methods were used to examine learners' programming behaviors, CT skills, and attitudes toward programming from the summative and process-oriented perspectives.

Additionally, to control for the increased likelihood of making a Type I error when conducting multiple tests, we adjusted the significance level (alpha level) for each test using the Bonferroni correction, which divides the overall alpha level of 0.05 by the number of tests conducted (Emerson, 2020).

Results

Learners' programming behaviors in two modalities

To answer RQ 1 (*What were the differences in learners' behaviors in learning* via *TPM versus BPM?*), learners' total programming behaviors in two modalities were summarized (see Table 5).

The results showed that some obvious differences in learners' programming behaviors between the two modalities. In particular, learners in TPM encountered more errors (NoE: M = 8.81, SD = 20.17) than those in BPM (NoE: M = 4.97, SD = 5.31). Learners' average time spent between two clicks of debug button in TPM (AtD: M = 84.74, SD = 83.35) was almost twice as long as that in BPM (AtD: M = 46.97, SD = 32.19). In addition, learners' average amount of code changing in BPM (AnC: M = 25.80, SD = 15.93) was slightly higher than that in TPM (AnC: M = 24.81, SD = 39.33). The number of clicks on debug button made by learners in BPM (NoD: M = 37.73, SD = 20.25) was higher than that in TPM (NoD: M = 32.03, SD = 43.06). Learners in BPM (NoIB: M = 35.20, SD = 24.06) had more irrelevant behaviors than those in TPM (NoIB: M = 25.53, SD = 17.26).

To further explore the differences in programming behaviors between the two modalities, learners' programming behaviors were modeled by using the features presented in Table 5. Previous studies found that it was possible to draw patterns using fine-grained data from one programming course (Estey & Coady, 2016; Munson & Zitovsky, 2018; Pereira et al., 2019). We inspected the k-means clusters, and the convergence of k-means was achieved in the 10th iteration with k = 5 as the best value with the highest value of mean silhouette coefficient (0.57). 23.44% of the learners were assigned to Cluster 1, 20.31% to Cluster 2, 3.12% to Cluster 3, 6.25% to Cluster 4 and 46.88% to Cluster 5 (see Table 6). Figure 4 depicts the programming profile of learners for each cluster in two modalities.

Comparing the features among five clusters, Cluster 1 was identified as the *Code Changer*, they had the second highest frequency of code changing (AnC: M = 44.53, SD = 10.69) and the number of clicks on debugs (NoD: M = 60.73, SD = 11.28), the second lowest number of irrelevant behaviors (NoIB: M = 23.73, SD = 12.09), and the moderate frequency of the number of errors (NoE: M = 10.67, SD = 7.89) amongst the five clusters. Based on the clustering analysis, we found that five students from TPM and ten students from BPM were assigned to this cluster. Cluster 2 was identified as the *Minimal Debugger* where students had the longest time interval between two clicks of debug (AtD: M = 174.43, SD = 72.95), and the lowest frequency of code changing (AnC: M = 4.00, SD = 2.45), number of clicks on debugs (NoD: M = 7.69, SD = 3.73), and the number of errors (NoE: M = 0.38, SD = 0.62). According to the cluster analysis results, it was determined that this particular cluster comprised ten students from TPM and three students from BPM. Two learners from TPM were clustered into Cluster 3 which was identified

Table 5 Mean and standard deviation of the programming behaviors for TPM and BPM

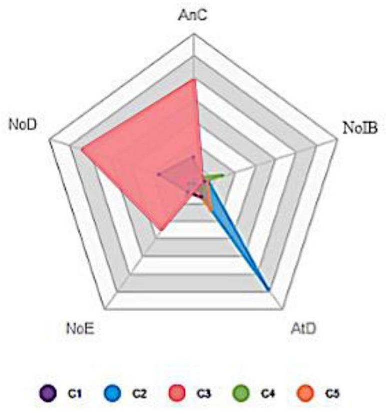
M	N	AnC	NoIB	NoD	AtD	NoE
TPM	32	24.81 (39.33)	25.53 (17.26)	32.03 (43.06)	84.74 (83.35)	8.81 (20.17)
BPM	32	25.80 (15.93)	35.20 (24.06)	37.73 (20.25)	46.97 (32.19)	4.97 (5.31)

TPM text-based modality, *BPM* block-based modality, *AnC* average number of code-changes, *NoIB* number of irrelevant behaviors, *NoD* number of debugs, *AtD* average time between two debugs, *NoE* number of errors

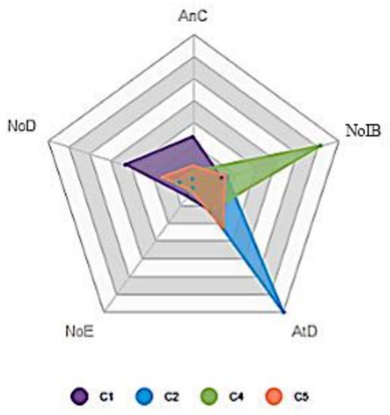
Table 6 Mean and standard deviation of the features for clusters in two modalities

Clu	Mod	*N*	AnC	NoIB	NoD	AtD	NoE
1	Total	15	44.53 (10.69)	23.73 (12.09)	60.73 (11.28)	26.04 (7.42)	10.67 (7.89)
	TPM	5	44.20 (10.11)	18.13 (10.11)	56.40 (9.58)	23.92 (9.58)	13.40 (9.58)
	BPM	10	44.70 (10.97)	26.53 (12.02)	62.90 (11.86)	27.09 (5.77)	9.30 (6.47)
2	Total	13	4.00 (2.45)	27.00 (13.49)	7.69 (3.73)	174.43 (72.95)	0.38 (0.62)
	TPM	10	3.00 (1.73)	25.69 (15.05)	6.10 (2.55)	186.05 (79.30)	0.30 (0.46)
	BPM	3	7.33 (1.25)	31.36 (3.09)	13.00 (1.41)	135.70 (12.18)	0.67 (0.94)
3	Total (TPM)	2	162.00 (35.00)	14.78 (10.78)	179.50 (40.50)	5.91 (2.23)	82.00 (14.00)
4	Total	4	11.39 (10.53)	84.25 (11.09)	10.00 (11.47)	15.11 (14.32)	11.75 (3.74)
	TPM	2	12.26 (0.42)	49.00 (0.77)	3.00 (0.17)	3.00 (0.48)	19.00 (0.82)
	BPM	2	13.50 (12.50)	119.50 (0.50)	17.00 (14.00)	27.23 (12.23)	4.50 (4.50)
5	Total	30	17.44 (7.60)	28.13 (10.06)	26.11 (8.98)	46.49 (16.96)	3.23 (3.12)
	TPM	14	15.07 (6.78)	24.57 (8.66)	21.71 (7.53)	47.90 (19.21)	3.21 (3.30)
	BPM	16	19.39 (7.70)	31.06 (10.18)	29.73 (8.44)	45.32 (14.75)	3.24 (29.60)

The table shows the average value of each behavior



(a) Five clusters in TPM



(b) Four clusters in BPM

Fig. 4 Programming profile of learners for each cluster in two modalities. *TPM* Text-based modality, *BPM* Block-based modality, *AnC* Average number of code-changes, *NoIB* Number of irrelevant behaviors, *NoD* Number of debugs, *AtD* Average time between two debugs, *NoE* Number of errors

as the *Maximal Debugger*. Here, they had the highest frequency in code changing (AnC: M = 162.00, SD = 35.00) and the numbers of debugs (NoD: M = 179.50, SD = 40.50), and they encountered the highest number of errors (NoE: M = 82.00, SD = 14.00), and had the lowest frequency in irrelevant behaviors (NoIB: M = 14.78, SD = 10.78) among five clusters. Cluster 4, which was identified as the *Distracted Coder,* had the highest frequency of irrelevant behaviors (NoIB: M = 84.25, SD = 11.09), the second lowest frequency of the number of debugs (NoD: M = 10.00, SD = 11.47), and the moderate behavior of the number of errors (NoE: M = 11.75, SD = 3.47). This cluster consisted of two students from

TPM and two students from BPM. Cluster 5, which was identified as the *Average Coder*, had the average frequency of code changing, number of debugs, number of errors, irrelevant behaviors, and the time interval between two clicks of debugs among the five clusters.

Learners' CT skills in two modalities

As to RQ 2 (*What were the differences in learners' CT skills in learning* via *TPM versus BPM?*), regarding learners' CT skills, there was no significant difference (creativity: $p=0.286$; algorithm thinking: $p=0.185$; cooperativity: $p=0.217$; critical thinking: $p=0.067$; problem solving: $p=0.095$; overall CT skill: $p=0.941$) before the intervention. The results revealed that, after the intervention, statistically significant differences were identified for algorithm thinking ($t=3.23$, $p=0.002$), cooperativity ($t=-2.11$, $p=0.038$), problem solving ($t=-2.72$, $p=0.008$) and overall CT skills ($t=-2.58$, $p=0.012$). In terms of algorithm thinking, cooperativity, problem solving and overall CT skills, learners in the BPM group outperformed those in the TPM group (see Table 7).

ANOVA analysis with Bonferroni correction results indicated that in TPM (see Table 8), Maximal Debuggers (C3) performed significantly better than Code Changers (C1) in the sub-items of problem-solving ($F=2.39$; $p=0.004$). However, in terms of CT skills, no significant difference was found among students across the four clusters in BPM, although Code Changers (C1) had the highest performance and Distracted Coders (C3) had the lowest performance.

Learners' attitudinal changes in two modalities

As to RQ 3 (*What were the differences in learners' perceived attitudes towards programming in learning* via *TPM versus BPM?*), the first attitudinal dimension sought to understand if learners enjoyed programming and, if so, how it differed by modality during the intervention time (see Fig. 5a). A Cronbach's Alpha test was run on these questions and found a sufficient level of correlation (Pre: $\alpha=0.82$, Post: $\alpha=0.86$), which met the

Table 7 Statistical summary of learners' CT skills in the two modalities

CT	Group	*M*	*SD*	*t*	*p*
Creativity	TPM	3.83	0.83	– 1.65	0.103
	BPM	4.16	0.60		
Algorithm thinking	TPM	3.57	0.87	– 3.23**	0.002
	BPM	4.24	0.66		
Cooperativity	TPM	3.85	1.07	– 2.11*	0.038
	BPM	4.35	0.68		
Critical thinking	TPM	3.78	0.94	– 1.05	0.296
	BPM	4.00	0.66		
Problem solving	TPM	3.94	0.67	– 2.72**	0.008
	BPM	4.39	0.63		
Overall	TPM	3.79	0.75	– 2.58*	0.012
	BPM	4.23	0.53		

*$p<.05$; **$p<.01$

Table 8 ANOVA results of learners' CT skills in different clusters

Group	CT	C1 M (SD)	C2 M (SD)	C3 M (SD)	C4 M (SD)	C5 M (SD)	*F*	
TPM	Overall	3.44 (0.61)	3.61 (0.76)	4.59 (0.26)	4.39 (0.19)	3.71 (0.51)	1.43	
	Creativity	3.05 (0.68)	3.75 (0.75)	4.50 (0.25)	4.68 (0.07)	3.25 (0.92)	1.15	
	Algorithm thinking	3.42 (0.75)	3.55 (0.70)	4.50 (0.50)	4.38 (0.37)	3.79 (0.89)	1.15	
	Cooperativity	3.55 (0.97)	3.60 (1.17)	4.00 (0.00)	4.28 (0.23)	4.00 (0.94)	1.05	
	Critical thinking	3.82 (0.85)	3.35 (0.95)	4.63 (0.38)	4.23 (0.02)	3.85 (0.86)	1.36	
	Problem solving	3.03 (0.58)	3.82 (0.68)	4.43 (0.67)	4.32 (0.25)	4.01 (0.48)	2.39**	C3 > C1
BPM	Overall	4.43 (0.46)	4.31 (0.28)	–	4.11 (0.47)	4.12 (0.56)	1.04	
	Creativity	4.40 (0.58)	4.42 (0.31)	–	4.00 (0.25)	4.01 (0.60)	1.61	
	Algorithm thinking	4.43 (0.54)	4.00 (0.54)	–	4.13 (0.63)	4.19 (0.70)	0.50	
	Cooperativity	4.45 (0.60)	4.83 (0.24)	–	4.38 (0.63)	4.22 (0.73)	0.78	
	Critical thinking	4.28 (0.63)	3.83 (0.24)	–	3.75 (0.50)	3.91 (0.69)	1.30	
	Problem solving	4.60 (0.54)	4.44 (0.42)	–	4.42 (0.25)	4.25 (0.69)	0.64	

We applied the Bonferroni correction to account for multiple comparisons and used an adjusted alpha level of 0.005

$^{**}p < 0.01$

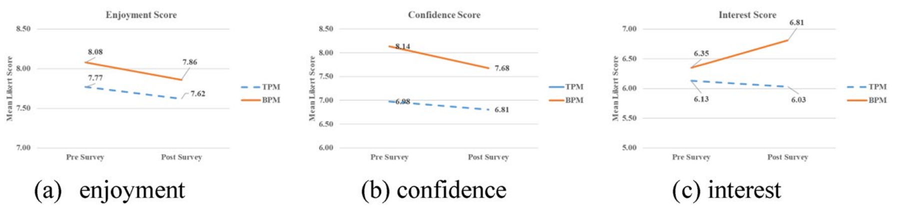


Fig. 5 Scores of learners' enjoyment (**a**), confidence (**b**) and interest (**c**) in programming at two points

0.70 thresholds often cited as the minimum level of acceptability for research purposes (Streiner, 2003). Examining the changes in the pre-and post-test, shown in Fig. 5a, a slight decrease was found in the two modalities. However, there was no statistically significant differences emerging between time periods within the groups or at the same time period across the modalities (Pre: $U = 431.00$, $p = 0.274$; Post: $U = 456.50$, $p = 0.454$). Such lack of difference leads to the conclusion that modality does not affect perceived enjoyment in novice learners, which also suggests that the increased enjoyment of programming found in other studies using block-based tools (e.g., Wilson & Moffat, 2010) may have more to do with the curriculum used or the context in which learners learn programming than the modality itself.

The second attitudinal dimension was learners' perceived confidence in their programming ability. The confidence questions had Cronbach's α scores of 0.85 on the pre-survey and 0.80 on the post-survey. Looking at the two distinct points at which the survey was administered (see Fig. 5b), we found a significant difference in confidence between the modalities in the pre-survey, while no significant difference was found in the post-survey (Pre: $U=333.00$, $p=0.016$; Post: $U=461.50$, $p=0.494$). Despite a significant difference between the conditions at the outset of the study, the difference did not appear in post administrations of the survey. This is echoed by Weintrop and Wilensky (2019) that programming modality alone does not seem to affect students' confidence in their programming ability.

The third attitudinal dimension was learners' interest in pursuing future computer science learning opportunities. A Cronbach's Alpha test was run on these questions and found a sufficient level of correlation (Pre: $\alpha=0.75$, Post: $\alpha=0.70$). Regarding learners' interest in future computer science courses (see Fig. 5c), the study found that learners in the two modalities started with different points (not significant) and went in different directions. BPM class showed an increasing trend, while TPM revealed a slightly decreasing trend in the post-survey. Besides, the study found no significant difference in confidence between the modalities (Pre: $U=462.00$, $p=0.501$; Post: $U=421.00$, $p=0.220$). This pattern matches the findings of researchers identifying the difficulty in text-based programming learning, such as higher demand for reserved knowledge and a higher possibility of grammatical errors (Armoni et al., 2015). Overall, the study's instructional treatments witnessed BPM participants becoming more interested in computer science courses, while learners in TPM became less interested.

ANOVA analysis with Bonferroni correction results revealed that, in terms of learners' attitudes toward programming in different clusters, the homogeneity of variances assumption was checked before the formal analysis and confirmed through Levene's tests for equality of variances (enjoyment ($F=0.009$; $p=0.927>0.05$); confidence ($F=0.448$; $p=0.506>0.05$); interest ($F=0.059$; $p=0.0.809>0.05$). Table 9 revealed that, in TPM, it was found that in the confidence aspect, Cluster 3 significantly outperformed Cluster 1 ($F=2.53$, $p=0.004$). In BPM, Cluster 2 scored significantly higher than Cluster 5 in interest level ($F=1.96$, $p=0.001$).

Table 9 ANOVA results of learners' attitudes toward programming in different clusters

Group	CT	C1		C2		C3		C4		C5		*F*	
		M	SD	M	SD	M	SD	M	SD	M	SD		
TPM	Enjoyment	6.33	2.34	7.20	2.16	9.34	0.67	9.50	0.50	7.13	1.39	1.67	
	Confidence	4.60	1.33	7.30	2.03	8.75	1.25	8.50	0.50	6.07	1.29	2.53**	C3 > C1
	Interest	6.72	1.50	5.90	1.93	9.00	1.00	6.50	0.50	5.62	1.51	2.23	
BPM	Enjoyment	7.73	1.89	8.22	1.85	–		8.00	1.67	7.29	1.87	0.21	
	Confidence	7.25	2.05	8.33	1.70	–		6.75	0.75	7.09	1.54	0.40	
	Interest	7.23	1.33	8.82	1.34	–		7.33	2.00	5.28	1.93	1.96**	C2 > C5

We applied the Bonferroni correction to account for multiple comparisons and used an adjusted alpha level of 0.005

*$p<0.01$

Discussion

Addressing research questions

With the more pervasive development of computer programming education, there is increasing research on how learning occurs under the two typical programming modalities (BPM and TPM). This study collected multi-modal data to analyze students' programming behaviors, CT skills, and attitudes toward programming in two programming modalities. Learners' CT skills under two modalities were compared through pre- and post-test, and changes in learners' attitudes toward programming under two modalities were also analyzed through pre- and post-surveys.

For research question 1, this research revealed that learners in TPM tended to spend more time between two clicks of debug button and encountered more syntactical errors. Students in BPM spent more time on code changing (operating blocks and adjusting parameters), made more attempts at debugging, and had more irrelevant behaviors. The research revealed five clusters based on students' programming behaviors, including *Code Changer* (C1), *Minimal Debugger* (C2), *Maximal Debugger* (C3), *Distracted Coder* (C4), and *Average Coder* (C5).

For research question 2, the results of comparing students' CT skills in the two modalities from the post-test found that learners in BPM outperformed those in the TPM group in terms of their algorithm thinking, cooperativity, and overall CT skills. One possible explanation is that block-based programming environments provide learners with a more intuitive and concrete way of coding, where they can drag and drop blocks to create programs instead of writing lines of text. This approach can help learners better understand fundamental programming concepts, such as sequencing, conditionals, loops, and variables, and develop their algorithmic thinking (Grover, 2021; Mladenović et al., 2018). Furthermore, BPM supports the iterative development of programs, enabling learners to see immediate results as they build their code. BPM also includes tools to promote cooperativity and collaboration, including the sharing of code resources, co-programming, and peer review sessions, which can enhance communication, cooperation, and social skills, all of which are essential components of CT (Grover & Basu, 2017; Jiang et al., 2021; Maloney et al., 2008).

Additionally, in TPM, frequent coding and debugging is associated with higher problem-solving performance, likely due to the expressive nature of languages like Python and their flexibility in designing tailored solutions (Kölling et al., 2015). Conversely, block-based programming languages, such as Scratch or Blockly, rely on pre-built code blocks and may limit the complexity and abstraction that can be achieved (Weintrop & Wilensky, 2019). Active coding and debugging is crucial for developing programming concepts and problem-solving skills, students can develop a deeper understanding of programming concepts and become more proficient at identifying and resolving problems. This skill is likely more effectively developed in text-based programming than block-based programming, as it requires students to work on their own code, rather than simply dragging and dropping pre-built blocks (Weintrop & Wilensky, 2015).

For research question 3, in terms of learners' attitudinal changes toward programming between pre- and post-test, learners in TPM showed a slight decrease in enjoyment, confidence, and interest in future CS. Learners in BPM showed an increase in interest, while they had a decrease in enjoyment and confidence for programming. One possible reason is the scaffolding approach used in block-based programming. Scaffolding allows learners to easily complete tasks without getting bogged down by syntax errors or other technical complications. As a result, learners may have been more motivated to engage in programming and learned to appreciate the satisfaction of programming. However, the decrease in enjoyment and confidence may reflect the limitations of block-based programming. Learners may perceive it as a less challenging, and thus less interesting, medium than text-based programming (Grover & Basu, 2017).

Furthermore, the Maximal Debugger (C3) in text-based programming demonstrated significantly higher confidence levels compared to the Code Changer (C1), indicating a correlation between text-based programming and increased self-efficacy (Lin & Weintrop, 2021). Furthermore, the Minimal Debugger (C2) in block-based programming exhibited significantly higher interest levels, potentially due to the immediate visual feedback and error highlighting offered by block-based programming (Tempel, 2013). The Minimal Debugger's additional features, such as step-by-step debugging, is likely to contribute to a positive learning experience in block-based programming.

Pedagogical implications

Firstly, instructors could consider integrating block-based programming into introductory programming courses to improve novice learners' CT skills. Programming can encourage learners to observe time-varying phenomena, support certain types of causal explanations, and segment the world into processes, which can help cultivate CT skills (Sherin, 2001). Our study found that learners in BPM achieved a higher level of overall CT skills and sub-items compared to those in TPM. This supports the effectiveness of BPM in facilitating novice learners' programming learning (Grover et al., 2015). In contrast, learners in TPM experienced a slight decrease in overall CT skills and the five sub-items. This may be due to the challenge of grammatical errors and the need for additional learning materials in the text-based modality (Armoni et al., 2015; Mladenović et al., 2018; Price & Barnes, 2015), as well as the time period of receiving instruction. Weintrop and Wilensky (2019) found that students in the text-based programming condition achieved fewer gains in computer science concepts during the introductory period but had incremental improvement in the following 15-week study.

Secondly, instructors may adopt different strategies in TPM and BPM to promote the learner's engagement during the programming process. Our results showed that, with its flexibility and convenience, the BPM led to fewer syntax mistakes and more debugging and irrelevant behaviors from learners. In addition, learners in BPM experienced increased interest in programming. Therefore, instructors should provide learners with more intervention when noticing learners' irrelevant behaviors in BPM. As for TPM, learners wrote longer lines of code, had longer time intervals between two clicks of debugs and encountered more syntax errors. Eid and Millham (2012) suggested that TPM required learners to spell and type in all the codes, which provided learners with a critical step for mastering a new programming language. Hence, in TPM, instructors may encourage learners to have more attempts in debugging and also provide learners with more scaffolding for error correction, such as an explanation list of some common debugging errors or a summary form of the most frequent syntactical errors from earlier programming practices for each learner.

Thirdly, to improve student performance, instructors should consider students' characteristics and encourage them to engage in both coding and debugging activities, regardless of the modality. Students who frequently engage in these activities tend to display higher problem-solving skills and interest in programming. Through coding and debugging, learners can deepen their understanding of programming concepts and become better problem solvers. Instructors can also assist students in identifying and resolving debugging errors by providing debugging information and sharing common errors among students in the class (Giannakos et al., 2013; Sun et al., 2021). By targeting debugging errors, instructors can help students become more proficient in debugging and improve their overall performance.

Taken together, the findings of this study are consistent with earlier research demonstrating that different representational forms have unique affordances that can affect learners' conceptualization of the material (Sherin, 2001; Weintrop & Wilensky, 2017). While programming instruction may vary by country and culture, and instructors may choose different modalities based on students' grade levels, the results of this study provide valuable insights for designing programming instruction in secondary and post-secondary education.

Conclusion and future directions

Identifying the fine-grained difference in programming modalities is crucial in promoting computer programming education and cultivating computational literacy. This quasi-experimental study in a secondary school compares learners' behavior patterns, CT skills, and attitudes toward programming in BPM and TPM, revealing differences between modalities. However, this study has three limitations. Firstly, our study is limited by the specific context in which it was conducted, as the sample used may not be representative of other populations. These limitations emphasize the need for cautious interpretation and replication with larger, more diverse samples. Secondly, this study collected survey data and actual behavioral data in an experimental study, future research could use performance-based tests or observation to gain a deeper understanding of CT skills and consider mixed methods with qualitative interviews to explore the learners' experiences and how BMP and TPM could complement each other. Thirdly, following the trend of multimodal learning analytics (Ochoa, 2017), this study collected multi-modal data from log files and survey data. Future research could consider collecting more diverse sources of data to reveal learners' behavioral, cognitive, metacognitive, and social activities during programming learning.

Overall, it is critical to cultivate young learners' CT to prepare them for future learning and jobs. In this context, this study that probes the differences in learning via block-based versus text-based programming modalities contributes to understanding effective approaches for implementing K-12 programming education and cultivating relevant CT skills.

Acknowledgements This work was supported by the National Natural Science Foundation of China (NSFC) [grant number 62307011]; the National Natural Science Foundation of China (NSFC) [grant number 61977057]; Guangdong Basic and Applied Basic Research Foundation, China [Grant No. 2021A1515110081]; Guangdong Planning Office of Philosophy and Social Science, China [Grant No. GD22XJY12]; Shenzhen Science, Technology and Innovation Commission, China [Grant No. 20220810115236001].

Funding This work was funded by National Natural Science Foundation of China (Grant No. 62307011), Dan Sun. National Natural Science Foundation of China (NSFC) (Grant No. 61977057), Yan Li. Guangdong Basic and Applied Basic Research Foundation (Grant No. 2021A1515110081), Miaoting Cheng. Guangdong Planning Office of Philosophy and Social Science (Grant No. GD22XJY12), Miaoting Cheng. Science, Technology and Innovation Commission of Shenzhen Municipality (Grant No. 20220810115236001), Miaoting Cheng.

Data availability The data can be made available upon request from the corresponding authors.

Declarations

Conflict of interest We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service and/or company that could be construed as influencing the position presented in, or the review of, the manuscript entitled.

Informed consent Informed consent was obtained from all individual participants included in the study.

Research involving human and animal rights All procedures performed in studies involving human participants were in accordance with the ethical standards of the institutional and/or national research committee.

References

Allsop, Y. (2019). Assessing computational thinking process using a multiple evaluation approach. *International Journal of Child-Computer Interaction, 19*, 30–55. https://doi.org/10.1016/j.ijcci.2018.10.004

Alshaigy, B., Kamal, S., Mitchell, F., Martin, C., & Aldea, A. (2015). Pilet: an interactive learning tool to teach python. In Judith, G. E., Sue, S., & Jan, V. (Eds.), *WiPSCE'15*: *Proceedings of the workshop in primary and secondary computing education* (pp. 76–79). ACM.

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 computational thinking curriculum framework: Implications for teacher knowledge. *Educational Technology & Society, 19*(3), 47–57.

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to "real" programming. *ACM Transactions on Computing Education, 14*(25), 1–15. https://doi.org/10.1145/2677087

Bai, X. M., & Gu, X. Q. (2019). Research on the construction and application of the computational thinking instrument in K12. *China Educational Technology, 10*, 83–90.

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming: Blocks and beyond. *Communications of the ACM, 60*, 72–80. https://doi.org/10.1145/3015455

Bey, A., Pérez-Sanagustín, M., & Broisin, J. (2019). Unsupervised automatic detection of learners' programming behavior. In Scheffel, M., Broisin, J., Pammer-Schindler, V., Ioannou, A., & Schneider, J. (Eds.), *EC-TEL 2021: Proceedings of the European conference on technology enhanced learning* (pp. 69–82). Springer.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In Arnetha, F. B., & Cynthia, A. T. (Eds.), *AERA' 12: Proceedings of the 2012 annual meeting of the American educational research association* (pp. 1–25). AERA.

Dijkstra, E. (1982). How do we tell truths that might hurt? *ACM SIGPLAN Notices, 17*(5), 13–15. https://doi.org/10.1145/947923.947924

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

Duncan, C., Bell, T., & Tanimoto, S. (2014). Should your 8-year-old learn coding? In Carsten, S., Michael, E. C., & Judith, G. E. (Chairs), *WiPSE'13*: *Proceedings of the 9th workshop in primary and secondary computing education* (pp.60–69). ACM.

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

Dutt, A., Aghabozrgi, S., Ismail, M. A. B., & Mahroeian, H. (2015). Clustering algorithms applied in educational data mining. *International Journal of Information and Electronics Engineering, 5*(2), 112. https://doi.org/10.7763/IJIEE.2015.V5.513

Eid, C., & Millham, R. (2012). Which introductory programming approach is most suitable for learners: Procedural or visual programming. *American Journal of Business Education, 5*(2), 173–178. https://doi.org/10.19030/ajbe.v5i2.6819

Estey, A., & Coady, Y. (2016). Can interaction patterns with supplemental study tools predict outcomes in CS1? In Alison, C., Ernesto, C. V. (Chairs), *ITiCSE'16*: *Proceedings of the 2016 ACM conference on innovation and technology in computer science education* (pp. 236–241). ACM. https://doi.org/10.1145/2899415.2899428

Falloon, G. (2016). An analysis of young learners' thinking when completing basic coding tasks using Scratch Jnr. on the iPad. *Journal of Computer Assisted Learning, 32*(6), 576–593. https://doi.org/10.1111/jcal.12155

Fay, M. P., & Proschan, M. A. (2010). Wilcoxon-mann-whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics Surveys, 4*, 1–39. https://doi.org/10.1214/09-SS051

García, D., Harvey, B., & Barnes, T. (2015). The beauty and joy of computing. *ACM Inroads, 6*, 71–79. https://doi.org/10.1145/2835184

Garneli, V., & Chorianopoulos, K. (2018). Programming video games and simulations in science education: Exploring computational thinking through code analysis. *Interactive Learning Environments, 26*(3), 386–401. https://doi.org/10.1080/10494820.2017.1337036

Giannakos, M. N., Koilias, C., Vlamos, P., & Doukakis, S. (2013). Measuring students' acceptance and confidence in algorithms and programming: The impact of engagement with CS on greek secondary education. *Informatics in Education, 12*(2), 207–219.

Grover, S. (2021). Teaching and assessing for transfer from block-to-text programming in middle school computer science. In C. Hohensee & J. Lobato (Eds.), *Transfer of learning. Research in mathematics education.* Springer.

Grover, S., & Basu, S. (2017). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. In Michael, E. C., & Stephen, H. E. (Eds.), *SIGCSE'17*: *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education* (pp. 267–272). ACM.

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school learners. *Computer Science Education, 25*(2), 199–237. https://doi.org/10.1080/08993408.2015.1033142

HelloGitHub. (2023, July 24). Programming Language Rankings in June 2023. https://hellogithub.com/report/tiobe/

Howland, K., & Good, J. (2014). Learning to communicate computationally with flip: a bi-modal programming language for game creation. *Computers & Education, 80*, 224–240. https://doi.org/10.1016/j.compedu.2014.08.014

ISTE. (2015). *CT leadership toolkit.* Retrieved 2015, http://www.iste.org/docs/ctdocuments/ct-leadershipt-toolkit.pdf?sfvrsn¼4.

Jiang, B., Zhao, W., Gu, X., & Yin, C. (2021). frame: a case study from scratch online community. *Educational Technology Research and Development, 69*(5), 2399–2421. https://doi.org/10.1007/s11423-021-10021-8

Jocius, R., O'Byrne, W. I., Albert, J., Joshi, D., Robinson, R., & Andrews, A. (2021). Infusing computational thinking into STEM teaching: From professional development to classroom practice. *Educational Technology & Society, 24*(4), 166–179.

Katai, Z. (2015). The challenge of promoting algorithmic thinking of both sciences-and humanities-oriented learners. *Journal of Computer Assisted Learning, 31*(4), 287–299. https://doi.org/10.1111/jcal.12070

Kölling, M., Brown, N., & Altadmri, A. (2015). *Frame-based editing: Easing the transition from blocks to text-based programming.* In Judith, G. E., Sue, S., & Jan, V. (Eds.), *WiPSCE '15: Proceedings of the workshop in primary and secondary computing education* (pp. 29–38). ACM.

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.

Lin, Y., & Weintrop, D. (2021). The landscape of Block-based programming: Characteristics of block-based environments and how they support the transition to text-based programming. *Journal of Computer Languages*, *67*, 101075. https://doi.org/10.1016/j.cola.2021.101075

Ma, H., Zhao, M., Wang, H., Wan, X., Cavanaugh, T. W., & Liu, J. (2021). Promoting pupils' computational thinking skills and self-efficacy: A problem-solving instructional approach. *Educational Technology Research and Development, 69*(3), 1599–1616. https://doi.org/10.1007/s11423-021-10016-5

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In Lucien, M. L. C., & Jerzy, N. (Eds.), *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (pp. 281–297). Berkeley.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with scratch. *SIGCSE Bulletin, 40*(1), 367–371. https://doi.org/10.1145/1352322.1352260

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., et al. (2014). Computational thinking in K-9 education. *Proceedings of the working group reports of the 2014 on innovation & technology in computer science education conference, ITiCSE-WGR 2014* (1–29). ACM.

Ministry of Education. (2017). General high school information technology curriculum standard (2017 Edition). *Ministry of Education of the People's Republic of China website*. http://www.moe.gov.cn/jyb_xxgk/xxgk_jyta/jyta_kjs/202002/.html

Mladenović, M., Boljat, I., & Žanko, Ž. (2018). Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. *Education and Information Technologies, 23*(4), 1483–1500. https://doi.org/10.1007/s10639-017-9673-3

Munson, J. P., & Zitovsky, J. P. (2018). Models for early identification of struggling novice programmers. In Tiffany, B., & Daniel, G. (Chairs), *SIGCSE'18*: *Proceedings of the 49th ACM technical symposium on computer science education* (pp. 699–704). ACM. https://doi.org/10.1145/3159450.3159476

Nolan, K., & Bergin, S. (2016). The role of anxiety when learning to program: A systematic review of the literature. In Judy, S., & Calkin, S. M. (Eds.), *Koli Calling'16*: *Proceedings of the 16th koli calling international conference on computing education research* (pp. 61–70). ACM.

Noone, M., & Mooney, A. (2018). Visual and textual programming languages: A systematic review of the literature. *Journal of Computers in Education, 5*(2), 149–174. https://doi.org/10.1007/s40692-018-0101-5

Ochoa, X. (2017). Chapter 11: multimodal learning analytics. In C. Lang, G. Siemens, A. Wise, & D. Gašević (Eds.), *Handbook of learning analytics* (1st ed., pp. 143–150). Creative Commons License.

Pellas, N., & Vosinakis, S. (2018). The effect of simulation games on learning computer programming: A comparative study on high school students' learning performance by assessing computational problem-solving strategies. *Education and Information Technologies, 23*(6), 2423–2452. https://doi.org/10.1007/s10639018-9724-4

Pereira, F. D., Oliveira, E., Cristea, A., Fernandes, D., Silva, L., Aguiar, G., Alamri, A., & Alshehri, M. (2019). Early dropout prediction for programming courses supported by online judges. In I. Seiji, M. A. O. Eva, H. Peter, M. Bruce, & L. Rose (Eds.), *Artificial intelligence in education* (pp. 67–72). Springer. https://doi.org/10.1007/978-3-030-23207-8_13

Pereira, F. D., Oliveira, E. H. T., Oliveira, D. B. F., Cristea, A. I., Carvalho, L. S. G., Fonseca, S. C., Toda, A., & Isotani, S. (2020). Using learning analytics in the Amazonas: Understanding students' behaviour in introductory programming. *British Journal of Educational Technology, 51*(4), 955–972. https://doi.org/10.1111/bjet.12953

Price, T.W., & Barnes, T. (2015) Comparing textual and block interfaces in a novice programming environment. In Brian, D. (Eds.), *ICER'15*: *Proceedings of the eleventh annual international conference on international computing education research* (pp.91–99). ACM.

Riley, D. D., & Hunt, K. A. (2014). *Computational thinking for the modern problem Solver*. CRC Press.

Rojas-López, A., & García-Peñalvo, F. J. (2018). Learning scenarios for the subject methodology of programming from evaluating the computational thinking of new students. *Revista Iberoamericana De Tecnologias Del Aprendizaje, 13*(1), 30–36. https://doi.org/10.1109/RITA.2018.2809941

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the computational thinking test. *Computers in Human Behavior, 72*, 678–691. https://doi.org/10.1016/j.chb.2016.08.047

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics, 20*, 53–65.

Scherer, R., Siddiq, F., & Sánchez Viveros, B. (2020). A meta-analysis of teaching and learning computer programming: Effective instructional approaches and conditions. *Computers in Human Behavior, 109*, 106349. https://doi.org/10.1016/j.MoB.2020.106349

Sherin, B. L. (2001). A comparison of programming languages and algebraic notation as expressive languages for physics. *International Journal of Computers for Mathematical Learning, 6*(1), 1–61. https://doi.org/10.1023/A:1011434026437

Shi, L., & Cristea, A. I. (2018). In-depth exploration of engagement patterns in MOOCs. In Hakim, H., Wojciech, C., Hua, M., Hye-Young, P., & Rui, Zhou. (Eds.), *ICWE 2019*: *Proceedings of the 19th international conference on web information systems engineering* (pp. 395–409). Springer.

Shi, L., Cristea, A. I., Toda, A. M., & Oliveira, W. (2019). Revealing the hidden patterns: A comparative study on profiling subpopulations of MOOC learners. In Siarheyeva. A., Barry. C., Lang, M., Linger, H., & Schneider, C. (Eds.), *Proceedings of the 28th international conference on information systems development*. Springer.

Streiner, D. L. (2003). Starting at the beginning: An introduction to coefficient Alpha and internal consistency. *Journal of Personality Assessment, 80*, 99–103. https://doi.org/10.1207/S15327752JPA8001_18

Sun, D., Ouyang, F., Li, Y., & Chen, H. (2021). Three contrasting pairs' collaborative programming processes in China's secondary education. *Journal of Educational Computing Research, 1*(8), 54.

Taub, R., Armoni, M., & Ben-Ari, M. (2012). CS unplugged and middle-school learners' views, attitudes, and intentions regarding CS. *ACM Transactions on Computing Education, 12*(2), 1–29. https://doi.org/10.1145/2160547.2160551

Tempel, M. (2013). *Blocks programming*. CSTA Voice.

TIOBE. (2023). *TIOBE Index for August 2023*. https://www.tiobe.com/tiobe-index/

Tsai, M. J., Liang, J. C., & Hsu, C.-Y. (2021). The computational thinking scale for computer literacy education. *Journal of Educational Computing Research, 59*(4), 579–602. https://doi.org/10.1177/0735633120972356

Weintrop, D. (2015). Comparing text-based, blocks-based, and hybrid blocks/text programming tools. Brian, D. (Ed.), *Proceedings of the eleventh annual international conference on international computing education research* (pp. 283–284). ACM.

Weintrop, D., & Wilensky, U. (2015). To block or not to block, that is the question: Learners' perceptions of blocks-based programming. In Marina, U. B., & Glenda, R. (Eds.), *Proceedings of the 14th international conference on interaction design and children* (pp. 199–208). ACM.

Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education, 18*(1), 1–25. https://doi.org/10.1145/3089799

Weintrop, D., & Wilensky, U. (2019). Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. *Computers & Education, 142*(103646), 1–17. https://doi.org/10.1016/j.compedu.2019.103646

Wilson, A., & Moffat, D. C. (2010). Evaluating Scratch to introduce younger schoolchildren to programming. *SCRATCHED website*. https://scratched.gse.harvard.edu/resources/evaluating-scratch-introduce-younger-schoolchildren-programming.html

Wing, J. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A, 366*, 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Wing, J. (2011). *Computational thinking—what and why?* Carnegie Mellon.

Wing, J. (2014). Computational thinking benefits society. *Social issues in computing website*. http://socialissues.cs.toronto.edu/index.html%3Fp=279.html.

Wu, B., Hu, Y., Ruis, A. R., & Wang, M. (2019). Analysing computational thinking in collaborative programming: A quantitative ethnography approach. *Journal of Computer Assisted Learning, 35*(3), 421–434. https://doi.org/10.1111/jcal.12348

Yucer, Y., & Rizvanoglu, K. (2019). Battling gender stereotypes: A user study of a code-learning game, "Code combat", with middle school children. *Computers in Human Behavior, 99*, 352–365. https://doi.org/10.1016/j.MoB.2019.05.029

Dan Sun is currently an assistant professor in Jing Hengyi School of Education, Chinese Education Modernization Research Institute of Hangzhou Normal University. Her research interests include AI in education, programming education, learning analytics, and computational thinking.

Chee-Kit Looi is currently Research Chair Professor of Learning Sciences in the Education University of Hong Kong, and Emeritus Professor of National Institute of Education, Nanyang Technological University.

His research focuses on learning sciences, computer-supported collaborative learning, mobile learning, AI in education, and computational thinking.

Yan Li is currently a professor at Zhejiang University, China, where she is also the director of the Research Center for AI in Education and the Director of the Department of Curriculum and Learning Sciences, Zhejiang University. Her research interests include distance education, ICT education, media education, AI in education, diffusion of educational innovations, etc.

Chengcong Zhu is currently an information technology teacher of Xiaoshan High School. His research interests include programming education, and artificial intelligence in education.

Caifeng Zhu is currently an information technology teacher at Hangzhou Chu Kochen Honors School. Her research interests include youth programming, and artificial intelligence in education.

Miaoting Cheng is an assistant professor in the Faculty of Education, Shenzhen University. Her research focuses on the social, cultural and psychological use of ICT, AI education, programming education, and technology acceptance. She has authored many publications in the aforementioned areas.

Authors and Affiliations

Dan Sun[1] · Chee-Kit Looi[2] · Yan Li[3] · Chengcong Zhu[4] · Caifeng Zhu[5] · Miaoting Cheng[6]

✉ Yan Li
yanli@zju.edu.cn

1 Jing Hengyi School of Education, Chinese Education Modernization Research Institute of Hangzhou Normal University, Yu Hang Tang Rd. #2318, Hangzhou 310058, Zhejiang, China

2 Department of Curriculum and Instruction, Education University of Hong Kong, New Territories, 10 Lo Ping Rd, Hong Kong SAR, China

3 College of Education, Zhejiang University, Yu Hang Tang Rd. #866, Hangzhou 310058, Zhejiang, China

4 Xiaoshan High School, Gongxiu Rd. #538, Hangzhou 311201, Zhejiang, China

5 Hangzhou Chu Kochen Honors School, Xi Pu Rd. #118, Hangzhou 310053, Zhejiang, China

6 Faculty of Education, Shenzhen University, Nanhai Rd. #3688, Shenzhen 518060, Guangdong, China