SCIENTIFIC CULTURE
www.sci-cult.com

SCIENTIFIC CULTURE, Vol. 10, No. 3, (2024), pp. 328-364
Open Access. Online & Print

DOI: 10.5281/zenodo.20240323

THE EFFECT OF NO-CODE DEVELOPMENT INTERFACE PATTERN, BLOCK-BASED VERSUS LOGIC-BASED, ON EDUCATIONAL TECHNOLOGY STUDENTS' EDUCATIONAL APPLICATION DEVELOPMENT SKILLS AND LOGICAL THINKING

Tamer M. Kamel[1]*, Mohamed W. Soliman[2], Nagwa Elshamy Elshamy Mohamed[3]

[1] Faculty of Specific Education, Kafr Elsheikh University, Kafr Elsheikh, Egypt; dr_tamerkamel@spe.kfs.edu.eg
[2] Faculty of Specific Education, Alexandria University, Alexandria, Egypt; mohamed_wahid2007@alexu.edu.eg
[3] Faculty of Specific Education, Kafr Elsheikh University, Kafr Elsheikh, Egypt; nagwa.mohamed@spe.kfs.edu.eg

Received: 28/11/2024
Accepted: 23/12/2024

Corresponding Author: Tamer M. Kamel (dr_tamerkamel@spe.kfs.edu.eg)

ABSTRACT

This study aimed to examine the effect of the no-code development interface pattern, in its two forms, the block-based interface and the logic-based interface, on developing educational application development skills and logical thinking among educational technology students. The study adopted a quasi-experimental design with two equivalent experimental groups. The research sample was randomly assigned to two experimental groups: the first group studied through a block-based no-code development interface using MIT App Inventor, whereas the second group studied through a logic-based no-code development interface using Microsoft Power Apps. Both treatments were implemented within a structured e-learning environment delivered through Moodle. The main research sample consisted of 100 third-year students from the Department of Educational Technology, with 50 students in each group. The experiment lasted for eight weeks during the first semester of the 2022/2023 academic year, beginning on Saturday, October 5, 2023. The study used a set of instruments, including a Cognitive Achievement Test, a Practical Performance Observation Checklist, a Final Product Quality Evaluation Rubric, a Logical Thinking Test, an Attitude Toward the No-Code Development Environment Scale, a Learning Environment Usability Questionnaire for the No-Code Development-Based Learning Environment, a Learning Interaction Analysis Log, and a Preliminary Data and Prior Experience Form. The validity and reliability of the instruments were verified through expert judgment, internal consistency, and reliability coefficients appropriate to the nature of each instrument. The data were analyzed using SPSS through descriptive statistics, the t-test, analysis of covariance (ANCOVA), and effect size calculations. The results showed that the block-based interface outperformed the logic-based interface in cognitive achievement, practical performance, final product quality, attitude toward the no-code development environment, and perceived usability. In contrast, the logic-based interface outperformed the block-based interface in logical

thinking and interaction within the learning environment. The significance of the study lies in providing an experimental model for employing no-code development environments in preparing educational technology students and in identifying the suitability of both block-based and logic-based interfaces for developing the cognitive, practical, and logical aspects associated with educational application development.

KEYWORDS: No-Code Development; Block-Based Interface; Logic-Based Interface; Educational Application Development; Logical Thinking; Educational Technology Students; MIT App Inventor; Microsoft Power Apps.

1. INTRODUCTION

The field of educational technology has witnessed a clear shift in the roles expected of its students. Students in this specialization are no longer required merely to produce digital learning materials or use learning platforms. Rather, they are increasingly expected to understand how to design interactive educational solutions that can be implemented, evaluated, and improved, enabling them to respond more effectively to the needs of contemporary learning.

The rapid development of no-code and low-code development tools has expanded the range of users who are able to produce digital applications. These tools enable users to build applications and digital functions through visual interfaces, configurable drag-and-drop components, or simplified logical formulas, without requiring them to write complex traditional programming code. Recent studies indicate that these environments are no longer limited to institutional or commercial use; they have become a promising area in education, particularly when training non-computer science specialists to develop practical digital solutions (Sońta & Przegalińska, 2023; Tsakalerou & Xenos, 2023).

No-code development environments are particularly important in preparing educational technology students because they help them move from merely using ready-made educational applications to producing small-scale educational applications that address specific instructional goals. Educational technology students need to understand the relationship between the instructional objective, interface design, interaction structure, feedback provision, and learner performance assessment within the application. These components make educational application development a design-based, pedagogical, and technical process at the same time (Corral et al., 2021; Matook et al., 2023).

The block-based interface is one of the prominent patterns in no-code development environments. It represents application logic through visual blocks that correspond to events, commands, conditions, variables, and responses. This visual representation helps novice learners understand the structure of interaction within the application, reduces syntax-related errors, and facilitates experimentation and modification. For this reason, block-based environments have become a common entry point for teaching programming, computational thinking, and problem-solving skills among learners (Hu et al., 2021; Çakiroğlu & Çevik, 2022; Sun et al., 2023).

The logic-based interface represents another pattern of no-code development. In this pattern, users build application behavior through formulas, properties, conditions, workflows, and relationships between user inputs and system responses. This interface does not merely provide visual elements that can be added; rather, it requires students to develop a clearer understanding of the logical relationships within the application. These relationships include when an action occurs, under what condition it occurs, what outcome is expected, and how a score is stored or feedback is displayed. Recent studies suggest that using low-code development platforms may be associated with metacognitive skills, decision-making, and the analysis of application development steps, especially when learners work in the context of producing a functional application (Matook et al., 2023; Lim et al., 2023).

Educational application development skills are connected to an integrated set of processes. These processes begin with analyzing the instructional idea and identifying the objectives and target learners, then move to designing the instructional scenario and application screens, adding components, adjusting their properties, building interaction logic, designing assessment and feedback, testing the application, and addressing errors. Therefore, assessing these skills should not be limited to a cognitive test alone. It requires instruments that measure practical performance and the quality of the final product, in addition to measuring the logical thinking associated with building relationships among events, conditions, and outcomes within the application (AERA et al., 2014; Fraenkel et al., 2023).

Logical thinking is an important variable in the context of educational application development. Building an application is not merely a matter of arranging screens or inserting components; it requires understanding causal and conditional relationships, analyzing errors, organizing data, and making appropriate decisions in response to user actions. These processes intersect with the literature on computational thinking and block-based programming, particularly skills such as abstraction, sequencing, conditional reasoning, debugging, and building simple algorithms (Wing, 2006; Çakiroğlu & Çevik, 2022; Irawan et al., 2023).

The success of a no-code development environment in an educational context does not depend on the technical tool alone. It is also related to the usability of the learning environment through which the treatment is delivered, the clarity of content, activities, instructions, and support, and the level of students' interaction with the environment's

resources. Usability refers to the extent to which users can achieve their goals effectively, efficiently, and satisfactorily within a specific context of use. Tracking students' interaction within learning environments also helps explain learning outcomes and understand patterns of participation, commitment, and return to resources (Brooke, 1996; ISO, 2018; Siemens & Long, 2011).

Previous studies indicate that block-based programming environments support the learning of programming concepts and computational thinking, and that low-code and no-code development platforms provide an important opportunity for teaching non-specialists how to produce practical digital applications. Nevertheless, there remains a need for experimental studies that compare different no-code development interface patterns, especially in the context of educational technology students, whose specialization requires the integration of instructional design, technical understanding, and the production of educational applications that can be implemented and evaluated (Hu et al., 2021; Sońta & Przegalińska, 2023; Tsakalerou & Xenos, 2023).

Accordingly, the present study was conducted to examine the effect of the no-code development interface pattern, block-based versus logic-based, on developing educational application development skills and logical thinking among educational technology students. The study employed a structured experimental treatment comparing two different patterns for representing application-building logic, while controlling for content, duration, activities, final task, and measurement instruments.

The present study is grounded in the assumption that the interface pattern through which no-code development is presented may shape students' learning experiences and outcomes. Although both groups engage with the same content, tasks, duration, Moodle environment, and assessment tools, the way in which application logic is represented differs substantially. The block-based interface represents events, conditions, variables, and responses through visual blocks, whereas the logic-based interface requires students to work with formulas, properties, conditional rules, and response flows. Figure 1 presents the conceptual framework guiding the study.

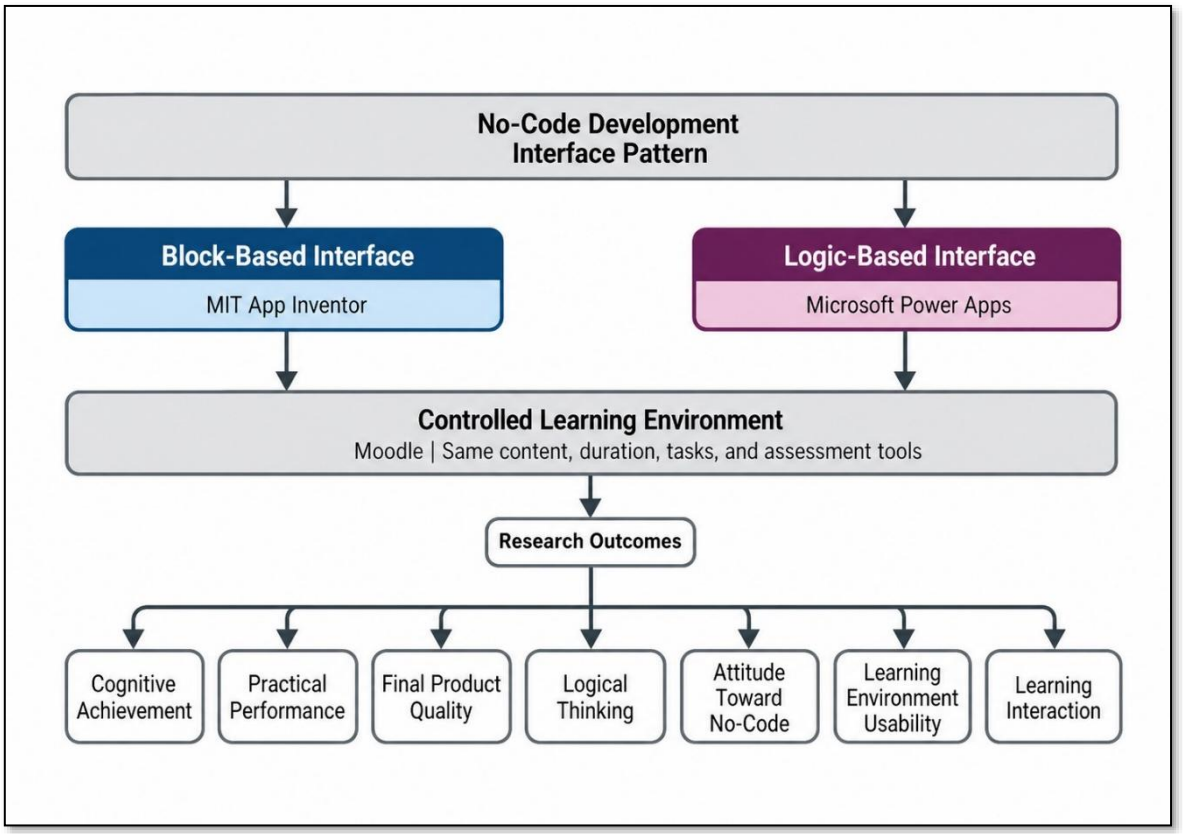

Figure 1. Conceptual Framework of the Effect of No-Code Development Interface Pattern on Research Outcomes.

As shown in Figure 1, the independent variable is the no-code development interface pattern, represented by two levels: the block-based interface using MIT App Inventor and the logic-based interface using Microsoft Power Apps. The learning environment was controlled through Moodle, with

equivalent content, duration, tasks, and assessment tools across the two groups. The dependent variables included cognitive achievement, practical performance, final product quality, logical thinking, attitude toward no-code development, learning environment usability, and learning environment interaction.

2. RESEARCH PROBLEM

Despite the growing use of digital tools in educational technology, students' general experience in using learning platforms or digital resources does not necessarily mean that they are able to develop interactive educational applications. Building an educational application requires skills that go beyond the superficial use of technology. Students need to analyze the instructional idea, design screens, insert components, build interaction logic, provide feedback, and test the final product. The literature on educational measurement and evaluation confirms that judging such complex skills requires a range of instruments that combine tests, performance rubrics, and product evaluation, rather than relying on a single cognitive measure (AERA et al., 2014; Fraenkel et al., 2023).

Before implementing the experimental treatment, the researchers conducted an exploratory study to identify educational technology students' level of experience in using no-code development environments. This was done using the Preliminary Data and Prior Experience Form. The form was designed as a supporting descriptive instrument to collect data on students' experience with e-learning environments, application development, and no-code development tools. It was not intended as a direct instrument for measuring the effect of the experimental treatment. Its results showed that students had general experience in dealing with learning platforms and digital resources, but they did not have direct experience in using no-code development tools or in using block-based or logic-based interfaces to build educational applications.

The exploratory study also revealed a low level of initial skills related to educational application development. Most students had no prior practical ability to build a simple logical condition within an application, calculate a learner's score, test an application and detect its errors, or build educational interaction based on user responses. These skills are central to the construction of an educational application, because an application does not become educational merely by displaying digital content. Rather, it becomes educational through its ability to guide the learner, receive the learner's response, analyze it, provide appropriate feedback, and connect the results to a clear learning path (Brooke, 1996; ISO, 2018).

The pre-measurement results supported this exploratory indication. Students' mean score on the Cognitive Achievement Test was 8.59 out of 30, their mean score on the Logical Thinking Test was 8.59 out of 30, and their mean score on the Practical Performance Observation Checklist was 47.96 out of 135. These values indicate a clear weakness in the cognitive, logical, and performance aspects associated with educational application development before implementing the treatment. This justifies the need to design a practical and structured instructional treatment that helps students gradually acquire development skills.

These indicators are consistent with previous studies showing that novice learners need development environments that reduce entry barriers and help them understand the structure of a program or application through clear visual or logical representations. Studies on block-based programming have shown that representing commands as blocks helps develop programming concepts and computational thinking while reducing the difficulty of textual syntax for novice learners (Hu et al., 2021; Çakiroğlu & Çevik, 2022; Sun et al., 2023). At the same time, studies on low-code and no-code development platforms point to the importance of training non-specialists to use these platforms in a structured way, while examining their effect on learning, performance, and thinking skills, rather than viewing them merely as tools for rapid production (Sońta & Przegalińska, 2023; Matook et al., 2023).

Recent literature recommends expanding experimental studies that examine the educational value of low-code and no-code development platforms, especially in higher education, with attention to learning experience, usability, digital production skills, and differences between interface patterns and tools used (Tsakalerou & Xenos, 2023; Lim et al., 2023). Studies on computational thinking also indicate that the type of programming or development environment can influence learners' behavior, ways of thinking, and level of engagement in building solutions. This makes the comparison between the block-based interface and the logic-based interface in the context of educational application development a research issue of clear importance (Hu et al., 2021; Irawan et al., 2023).

Accordingly, the research problem of the present study is defined by the need to develop educational application development skills and logical thinking

among educational technology students, in light of their limited prior experience and low pre-measurement scores, while also identifying the effect of the no-code development interface pattern, block-based versus logic-based, on these variables. Therefore, the present study sought to answer the following main question:

What is the effect of the no-code development interface pattern, block-based versus logic-based, on developing educational application development skills and logical thinking among educational technology students?

3. RESEARCH QUESTIONS

The main research question was divided into the following sub-questions:

1. What is the effect of the no-code development interface pattern, block-based versus logic-based, on developing cognitive achievement related to educational application development concepts among educational technology students?
2. What is the effect of the no-code development interface pattern, block-based versus logic-based, on developing the practical performance of educational application development skills among educational technology students?
3. What is the effect of the no-code development interface pattern, block-based versus logic-based, on the quality of the final educational application produced by educational technology students?
4. What is the effect of the no-code development interface pattern, block-based versus logic-based, on developing logical thinking among educational technology students?
5. What is the effect of the no-code development interface pattern, block-based versus logic-based, on educational technology students' attitudes toward the no-code development environment?
6. What is the effect of the no-code development interface pattern, block-based versus logic-based, on the usability of the no-code development-based learning environment from the perspective of educational technology students?
7. What is the effect of the no-code development interface pattern, block-based versus logic-based, on educational technology students' level of interaction within the learning environment?

4. RESEARCH OBJECTIVES

The present study aimed to examine the effect of the no-code development interface pattern, block-based versus logic-based, on developing educational application development skills and logical thinking among educational technology students.

This general objective was further translated into the following objectives:

1. To measure the effect of the no-code development interface pattern on developing cognitive achievement related to educational application development concepts among educational technology students.
2. To determine the effect of the no-code development interface pattern on developing the practical performance of educational application development skills among educational technology students.
3. To examine the effect of the no-code development interface pattern on the quality of the final educational application produced by educational technology students.
4. To measure the effect of the no-code development interface pattern on developing logical thinking in the context of no-code educational application development.
5. To identify the effect of the no-code development interface pattern on educational technology students' attitudes toward the no-code development environment.
6. To measure the effect of the no-code development interface pattern on the usability of the no-code development-based learning environment.
7. To analyze the effect of the no-code development interface pattern on students' level of interaction within the learning environment.

5. SIGNIFICANCE OF THE STUDY

5.1 Theoretical Significance

The theoretical significance of the study lies in its treatment of a recent topic in educational technology: the use of no-code development environments in preparing students in the specialization to produce interactive educational applications. The study also distinguishes between two development interface patterns, namely the block-based interface and the logic-based interface. This distinction is important because each pattern represents the logic of application development in a different way, which may be reflected in students' learning, performance, and logical thinking.

The study also contributes to enriching the literature that connects educational application development with logical thinking. It approaches the educational application as an interactive structure consisting of objectives, content, screens, components, events, conditions, responses, and feedback, rather than as a merely formal digital product. In this way, the study broadens the understanding of educational technology students'

skills to include the ability to build application logic and analyze the relationships between user inputs and interaction outcomes.

5.2 Practical Significance

The practical significance of the study is reflected in providing an instructional treatment that can be employed within the course of Educational Software Design and Production, through training students to develop an interactive educational application without traditional programming. The findings may also help faculty members select the most appropriate interface pattern according to the instructional objective. If the main objective is to support practical performance and the quality of the final product, the block-based interface may be more appropriate. If the objective is to develop logical thinking and analyze conditional relationships, the logic-based interface may be more suitable.

The study also provides a set of measurement instruments that may be used in later studies, including the Cognitive Achievement Test, the Practical Performance Observation Checklist, the Final Product Quality Evaluation Rubric, the Logical Thinking Test, the Attitude Scale, the Learning Environment Usability Questionnaire, and the Interaction Analysis Log.

5.3 Methodological Significance

The methodological significance of the study lies in its use of a quasi-experimental design with two equivalent experimental groups, while controlling for content, duration, activities, final task, and measurement instruments. The study also combines instruments that measure knowledge, performance, product quality, logical thinking, attitudes, usability, and interaction. This allows for a more comprehensive understanding of the effect of the no-code development interface pattern.

6. DELIMITATIONS OF THE STUDY

The present study was delimited as follows:

6.1 Human Delimitations

The study was limited to a sample of 100 third-year students from the Department of Educational Technology, Faculty of Specific Education, Alexandria University. The students were randomly assigned to two equivalent experimental groups, with 50 students in the block-based interface group and 50 students in the logic-based interface group.

6.2 Spatial Delimitations

The study was conducted in the Department of Educational Technology, Faculty of Specific Education, Alexandria University.

6.3 Temporal Delimitations

The research experiment was implemented during the first semester of the 2022/2023 academic year, beginning on Saturday, October 5, 2023, and lasting for eight weeks.

6.4 Subject Delimitations

The study was limited to examining the effect of the no-code development interface pattern in two forms:

- The block-based interface using MIT App Inventor.
- The logic-based interface using Microsoft Power Apps.

The study examined the development of the following dependent variables:

- Cognitive achievement related to educational application development concepts.
- Practical performance of educational application development skills.
- Quality of the final educational application.
- Logical thinking.
- Attitude toward the no-code development environment.
- Usability of the no-code development-based learning environment.
- Interaction within the learning environment.

7. OPERATIONAL DEFINITIONS

7.1 No-Code Development Interface Pattern

The no-code development interface pattern is operationally defined in the present study as the way in which the development environment presents the tools used to build the application and its behavior to educational technology students, either through visual blocks that can be assembled or through logical formulas, properties, and conditions. This pattern has two levels: the block-based interface and the logic-based interface.

7.2 Block-Based Interface

The block-based interface is operationally defined in the present study as a no-code development interface pattern that builds the behavior of the educational application through assembling visual blocks representing events, commands, conditions, variables, and responses. In this study, it was represented by the use of MIT App Inventor by students in the first experimental group to develop an interactive educational application.

7.3 Logic-Based Interface

The logic-based interface is operationally defined in the present study as a no-code development interface pattern that builds the behavior of the educational application through formulas, properties, conditions, workflows, and logical relationships between user inputs and application responses. In this study, it was represented by the use of Microsoft Power Apps by students in the second experimental group to develop an interactive educational application.

7.4 Educational Application Development Skills

Educational application development skills are operationally defined in the present study as the set of knowledge and practical performances that enable educational technology students to analyze the idea of an educational application, design its screens, add its components, adjust its properties, build its internal interaction logic, design assessment and feedback, test it, and submit it in an executable form. In this study, these skills are measured through the Cognitive Achievement Test, the Practical Performance Observation Checklist, and the Final Product Quality Evaluation Rubric.

7.5 Educational Application

An educational application is operationally defined in the present study as a small-scale interactive digital product developed by educational technology students using a no-code development environment. It includes an instructional objective, organized content, an interactive activity, a short assessment, immediate feedback, and a mechanism for calculating the learner's score or displaying the final result.

7.6 Logical Thinking

Logical thinking is operationally defined in the present study as educational technology students' ability to analyze educational application development situations according to organized relationships among the event, condition, action, and outcome. This includes arranging work steps, conditional reasoning, cause-and-effect analysis, detecting logical errors, organizing data and variables, and making appropriate decisions in application development situations. It is measured by the score students obtain on the Logical Thinking Test prepared for this study.

7.7 Attitude Toward the No-Code Development Environment

Attitude toward the no-code development environment is operationally defined in the present study as students' affective and cognitive response to using the no-code development environment in producing educational applications. This includes perceived educational value, perceived ease of use, motivation, confidence in the ability to develop applications, and future intention to use. It is measured by the score students obtain on the Attitude Toward the No-Code Development Environment Scale.

7.8 Usability of the No-Code Development-Based Learning Environment

Usability of the no-code development-based learning environment is operationally defined in the present study as the degree to which students perceive the e-learning environment through which the treatment was delivered as easy, clear, and effective to use. This includes ease of access, content organization, ease of navigation, clarity of task instructions, adequacy of support, and overall satisfaction with the environment. It is measured by the score students obtain on the Learning Environment Learning Environment Usability Questionnaire.

7.9 Interaction Within the Learning Environment

Interaction within the learning environment is operationally defined in the present study as the extent of students' regular access to the Moodle environment and their interaction with instructional content, training activities, support resources, discussion spaces, and educational application development task submissions during the treatment period. It is measured by the score students obtain in the Learning Interaction Analysis Log.

8. THEORETICAL FRAMEWORK

The theoretical framework of the present study addresses five main axes: no-code development in educational technology, the block-based interface in educational application development, the logic-based interface in educational application development, educational application development skills, and logical thinking in the context of no-code educational application development. These axes were developed to align closely with the study variables and to provide a coherent basis for interpreting the possible relationship between the no-code development interface pattern and the targeted cognitive, practical, and logical learning outcomes.

8.1 No-Code Development in Educational Technology

No-code development refers to building applications or digital solutions through interfaces and tools that reduce or eliminate the need to write traditional programming code. Instead, such environments rely on ready-made components, drag-and-drop features, visual blocks, logical formulas, or workflows. This direction has emerged in response to the need to enable non-programmers to participate in producing digital solutions, accelerate the application development cycle, and reduce the gap between an educational or institutional need and the technical ability to produce a suitable functional application (Sońta & Przegalińska, 2023; Tsakalerou & Xenos, 2023).

No-code development should not be viewed as merely a technical shortcut to programming. Rather, it represents a shift in the way application development is learned. It allows learners to focus on solution logic, user experience design, data organization, and the relationship between inputs and outputs, instead of becoming preoccupied with the details of textual programming syntax. Recent studies have indicated that low-code and no-code development platforms can support non-specialists' learning and open opportunities for them to acquire applied digital competencies that go beyond the conventional use of technology (Lim et al., 2023; Matook et al., 2023).

In educational technology, the importance of no-code development becomes particularly clear because it enables students to build small-scale educational applications that respond to specific pedagogical objectives, such as presenting concise content, implementing a training activity, delivering a short quiz, or providing immediate feedback. This differs from merely using ready-made applications, as students become able to transform an instructional idea into a digital product. This process requires the integrated use of instructional design principles, usability considerations, interaction design, and learning assessment (Corral et al., 2021; Matook et al., 2023).

The literature shows that low-code and no-code development platforms should not be treated as neutral technical tools, because the nature of their interfaces and the mechanisms through which they build interaction influence how learners think during development. A student working with a visual block-based interface may construct understanding through assembling visible and direct commands, whereas a student working with an interface based on formulas or logical relationships may need to analyze the condition, action, and outcome before implementing behavior within the application (Hu et al., 2021; Matook et al., 2023; Sun et al., 2023).

Accordingly, using no-code development in preparing educational technology students is not merely a matter of choosing an easy-to-use tool. It requires analyzing the nature of the development interface itself and the extent to which it fits the targeted skill. If the aim is to help students produce a functional educational product, an interface that visually represents relationships may be more supportive of practical performance. If the aim is to deepen students' thinking about conditions and response flows, a logic-based interface may be more closely connected to analysis and reasoning skills (Çakiroğlu & Çevik, 2022; Sońta & Przegalińska, 2023).

8.2 The Block-Based Interface in Educational Application Development

The block-based interface presents programming commands in the form of visual blocks that can be assembled, with each block representing an event, action, condition, variable, or logical operation. This pattern allows novice learners to build application behavior by putting together suitable blocks instead of writing textual commands that require prior knowledge of programming language rules. Meta-analyses and experimental studies have shown that block-based environments can support the learning of programming concepts, develop computational thinking, and improve learners' attitudes toward programming (Hu et al., 2021; Sun et al., 2023).

The idea of the block-based interface is consistent with the principles of visual learning and learning by doing. Students can see the relationships among parts of the programming behavior in a relatively concrete form. They can recognize that an event leads to an action, that a condition determines the response path, and that a variable can store a value that changes during interaction. This representation helps reduce the burden of textual syntax and directs students' attention toward understanding the relationship among the parts of the solution, rather than toward typing errors, punctuation, or the structure of a programming language (Çakiroğlu & Çevik, 2022; Irawan et al., 2023).

In the context of educational application development, the block-based interface can help students build a small-scale application that includes screens, components, buttons, questions, and feedback. For example, a student can connect a "Check" button to an event block, add a condition that verifies the user's answer, and then display an

appropriate message or add a score to a variable. This visible relationship among blocks makes the application logic easier to understand and trace, and it helps students detect errors by examining the order and sequence of blocks (Corral et al., 2021; Ruiz-Rube et al., 2019).

MIT App Inventor is a prominent example of block-based interfaces for mobile application development. It separates interface design through the Designer from application behavior through the Blocks Editor. This separation helps learners distinguish between constructing the visual form of the application and constructing its operating logic. It also allows them to connect visual components with events, responses, and conditions through blocks (Corral et al., 2021; Ruiz-Rube et al., 2019).

Recent studies have shown that block-based environments do not only support procedural learning. They can also contribute to the development of higher-order skills, such as algorithmic thinking, abstraction, problem solving, and debugging, especially when used in instructional activities that require learners to design a digital product or solve a real problem (Çakiroğlu & Çevik, 2022; Hu et al., 2021; Pršala, 2023). However, this effect does not occur automatically simply by using the environment. It depends on the quality of activity design, task clarity, gradual support, and the nature of the product required from students.

Based on this, the present study expects the block-based interface to support practical performance and final product quality to a clear degree, because it provides students with a direct visual way to build interactive behavior within the application, reduces the difficulty of starting, and helps them trace the application pathway while working. However, this does not mean that the block-based interface is superior in all variables. The logic-based interface may be more capable of prompting students to analyze conditional relationships in a more abstract and organized way.

8.3 The Logic-Based Interface in Educational Application Development

In the present study, the logic-based interface refers to a no-code development interface pattern in which application behavior is built through formulas, properties, conditions, workflows, and relationships between user inputs and application responses. This interface does not necessarily rely on assembling visual blocks. Rather, it requires students to think about each component in terms of its properties, what happens when the user interacts with it, and how the state of the application changes when a certain condition is met or not met (Lim et al., 2023; Matook et al., 2023).

The logic-based interface is characterized by bringing students closer to thinking about the application as a system of rules and relationships. When building an educational activity, the student does not merely add a question and a button. The student must determine the condition that makes the answer correct, the formula that updates the score, the action that displays feedback, and the pathway that moves the learner to another screen. This type of work supports cause-and-effect analysis, conditional reasoning, and decision-making based on data or user responses (Matook et al., 2023; Silva et al., 2021).

Microsoft Power Apps represents a low-code/no-code development environment that can embody the logic-based pattern in the present study. It allows the development of Canvas Apps through screens and controls, while the properties and behavior of these controls are adjusted using formulas and logical functions. From an educational perspective, this environment can be used to train students to control the relationship between the element, the property, and the formula, such as determining what happens when a button is pressed, showing a message according to a condition, or updating a score variable (Lim et al., 2023; Matook et al., 2023).

The literature suggests that low-code development platforms may contribute to developing learners' metacognitive skills, because building an application requires planning, monitoring progress, testing the solution, modifying errors, and analyzing reasons for success or failure. This is connected to the nature of work within the logic-based interface, where an error may not always appear as an incorrectly assembled block; it may appear as an inaccurate formula, an unsuitable condition, or an incorrect sequence of actions (Matook et al., 2023; Torres & Kapralos, 2023).

From the perspective of logical thinking, the logic-based interface can provide a rich environment for developing conditional reasoning because it repeatedly places students in front of questions such as: If this condition is met, what is the appropriate action? If it is not met, what is the alternative? How can the result be stored? When should feedback appear? Such questions make students more engaged in constructing abstract mental relationships among application components, rather than merely executing visible steps (Çakiroğlu & Çevik, 2022; Wing, 2006).

Nevertheless, the logic-based interface may be more challenging for novice students because it requires understanding formulas, properties, and the

sequence of procedures. It may increase cognitive load in the early stages of learning if it is not accompanied by gradual support and clear examples. Usability studies of low-code development platforms indicate that interface complexity or ambiguity in the relationships among components and properties may affect user experience, especially when learners have limited programming backgrounds (Silva et al., 2021; Torres & Kapralos, 2023).

The present study therefore expects the logic-based interface to be more closely associated with developing logical thinking and analytical interaction within the learning environment, whereas the block-based interface may be more supportive of practical performance, product quality, and ease of use. This expectation is not based on privileging one tool over another, but on the different ways in which each interface pattern represents application logic.

8.4 Educational Application Development Skills

Educational application development skills consist of an interconnected set of knowledge and performances. These begin with identifying an instructional problem or training need, formulating objectives, analyzing learner characteristics, selecting content, designing interaction, and determining assessment and feedback methods. Therefore, developing an educational application cannot be reduced to the technical aspect alone; it is also linked to instructional design principles, user experience, content quality, and interaction quality (AERA et al., 2014; Fraenkel et al., 2023).

The first stage in developing an educational application involves analyzing and planning the instructional idea. This means that the student identifies the application objective, target group, nature of the content, type of activity, and expected navigation path. This stage is essential because an application that does not begin from a clear objective may become a visually attractive digital interface but remain pedagogically weak (Fraenkel et al., 2023; ISO, 2018).

The second stage concerns the design of the educational user interface. Screens should be clear and organized, buttons should have understandable functional meanings, visual clutter should be minimized, and the application should maintain consistency in colors, fonts, and element distribution. This stage is connected to usability principles, which emphasize effectiveness, efficiency, and satisfaction in the context of use, as well as the importance of enabling users to achieve their goals without confusion or unnecessary errors (Brooke, 1996; ISO, 2018; Nielsen, 2023).

The third stage involves building the application components and functions, such as text, buttons, images, input fields, answer options, media components, and result-display elements. These components are not educational in themselves. They become educational when they are employed to serve a specific objective, such as presenting a concept, guiding the learner, receiving the learner's answer, providing an example, or supporting an interactive activity (Corral et al., 2021; Ruiz-Rube et al., 2019).

The fourth stage is building application logic and interaction, which is one of the most important stages of educational application development because it determines what happens when the user interacts with the application. This stage includes creating events, building conditions, distinguishing between correct and incorrect responses, updating the score, displaying feedback, and organizing navigation between screens. It is closely related to logical and computational thinking skills because it requires understanding the relationships among input, processing, and output (Çakiroğlu & Çevik, 2022; Wing, 2006).

The fifth stage relates to assessment and feedback. The educational application should include an activity or a short test connected to the objectives, and it should provide immediate and understandable feedback that does not merely rely on general statements such as "correct" or "incorrect." Rather, feedback should help learners understand why their response is correct or incorrect. Feedback is a central element in educational applications because it transforms digital interaction into a learning experience that can be improved and revised (Hattie & Timperley, 2007; Shute, 2008).

The sixth stage involves testing the application and fixing errors before submission. At this stage, students test the application from beginning to end, checking buttons, navigation, conditions, score calculation, and feedback display. The literature on programming education and computational thinking emphasizes that debugging and correction are fundamental aspects of learning to build digital solutions, because learners learn by comparing the expected outcome with the actual outcome (Çakiroğlu & Çevik, 2022; Wing, 2006).

Given the diversity of these skills, the present study measured educational application development skills through three integrated indicators: cognitive achievement, practical

performance during development, and final product quality. This multi-dimensional measurement is based on the idea that possessing knowledge does not necessarily mean mastering performance, and that the quality of the final application alone does not reveal all the details of the practical performance students carried out while building the application (AERA et al., 2014; Fraenkel et al., 2023).

Educational application development skills in this study were conceptualized as a multidimensional construct rather than a single technical skill. Developing an educational application requires students to analyze the instructional idea, plan the learning flow, design the user interface, build screens and components, develop interaction logic, design assessment and feedback, test and debug the application, and prepare the final output for submission. Figure 2 summarizes the main dimensions of educational application development skills assessed in the study.

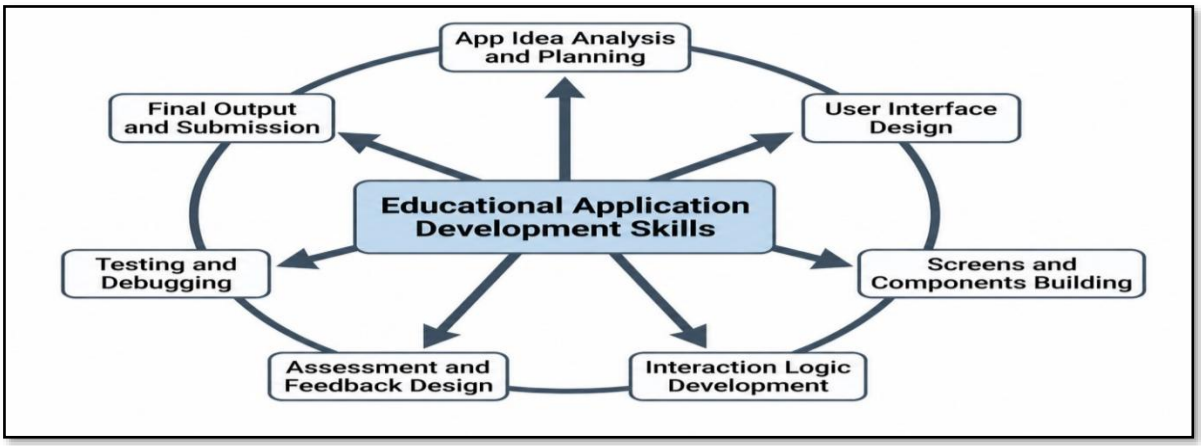

Figure 2. Dimensions of Educational Application Development Skills Assessed in the Study.

Figure 2 indicates that educational application development requires the integration of instructional, technical, and evaluative skills. These dimensions informed the construction of the Practical Performance Observation Checklist and the Final Product Quality Evaluation Rubric. Therefore, students' application development skills were assessed not only through cognitive achievement, but also through observed practical performance and the quality of the final educational application.

8.5 Logical Thinking in the Context of No-Code Educational Application Development

Logical thinking is one of the essential requirements for developing educational applications because application development depends on arranging procedures, analyzing conditions, inferring outcomes, detecting errors, organizing data, and making appropriate design decisions. This type of thinking is closely connected to the computational thinking literature, particularly concepts such as analysis, abstraction, algorithm construction, debugging, and understanding the relationship between a problem and its solution (Wing, 2006; Irawan et al., 2023).

In no-code development environments, logical thinking does not necessarily appear through writing textual code. Rather, it appears in the way students build the application pathway. When a student determines that pressing a certain button should lead to a specific screen, that a particular answer should add a score, or that a low score should direct the learner to a remedial activity, the student is practicing logical thinking based on the relationship among event, condition, action, and outcome (Çakiroğlu & Çevik, 2022; Matook et al., 2023).

Logical thinking in the present study includes six main dimensions: logical sequencing of procedures, conditional reasoning, cause-and-effect analysis, detection of logical errors, organization of data and variables, and logical decision-making in application development situations. These dimensions are consistent with the nature of educational application development, as students need to arrange development steps, understand when a response operates, analyze the cause of an error, and choose an appropriate method for storing a score or displaying a result (Hu et al., 2021; Wing, 2006).

Logical sequencing of procedures appears when students are able to arrange the stages of application development, from identifying the objective to design, construction, and testing, or when they arrange the learner's pathway within the application from the start screen to the content, then to the activity, and finally to the result. Studies on block-based programming emphasize that such sequencing

is fundamental to understanding the structure of a digital solution, and that visual environments can help learners perceive the sequence of commands more clearly (Hu et al., 2021; Sun et al., 2023).

Conditional reasoning appears when students understand the relationship between "if" and "then." For example, if the answer is correct, a score is added; if it is incorrect, feedback is displayed. This relationship is essential in building interactive educational applications because it enables the application to respond to differences in learner performance, rather than presenting a fixed pathway that is not affected by user input (Çakiroğlu & Çevik, 2022; Matook et al., 2023).

Cause-and-effect analysis appears when students are able to explain why a malfunction occurs within the application, such as a button not working, an incorrect message appearing, or a score being calculated inaccurately. This is closely related to debugging, which is a fundamental skill in application development. Students do not simply build functions; they test them and compare actual behavior with expected behavior (Irawan et al., 2023; Wing, 2006).

The organization of data and variables is connected to students' understanding of the function of variables, fields, or values within the application, such as storing the learner's score, the number of attempts, or the answer status. This skill represents a bridge between logical thinking and instructional design because data are not collected for their own sake; they are used to make decisions within the application, such as displaying a result, presenting a remedial activity, or allowing another attempt (Matook et al., 2023; Silva et al., 2021).

The present study expects that the block-based and logic-based interfaces may differ in their effects on logical thinking. The block-based interface makes procedural relationships visible and easier to trace, which may help students understand logic through visual representation. The logic-based interface, however, pushes students to think more abstractly about formulas, conditions, and response flows. Therefore, the comparison between the two patterns does not aim to identify one tool as absolutely superior, but to understand which pattern is more appropriate for each aspect of learning, performance, and thinking.

8.6 Usability of the Learning Environment and Interaction Within It

The effect of the no-code development interface pattern cannot be interpreted separately from the learning environment through which the treatment is delivered. The clarity of content, ease of access to activities, adequacy of support, and organization of assignments are all factors that may influence students' experience and achievement. The usability literature indicates that a good system enables users to achieve their goals effectively, efficiently, and satisfactorily within a specific context of use. This makes usability an important variable when evaluating digital learning environments (Brooke, 1996; ISO, 2018).

In the present study, Moodle was used as the host environment for the instructional treatment. The content was organized into weekly units, and activities, assignments, and support resources were made available. Interaction logs were used to track students' access, resource views, activity completion, and application development task submissions. Such logs provide supporting indicators for interpreting learning outcomes because students' final performance cannot be separated from the regularity of their use of the environment and their interaction with its resources (Gašević et al., 2015; Siemens & Long, 2011).

The learning analytics literature emphasizes that interaction data within e-learning environments can help researchers understand participation patterns, monitor learning behaviors, and interpret differences among students, especially when these data are analyzed alongside test results, performance, and final products. However, caution is needed when interpreting frequent logins or high numbers of clicks as direct evidence of learning quality. Such indicators may reflect actual engagement, but they may also indicate difficulty, hesitation, or repeated searching for support (Gašević et al., 2015; Siemens & Long, 2011).

In the context of the present study, the Interaction Analysis Log was not used as a substitute for the achievement test, performance checklist, or logical thinking test. Rather, it was used as a supporting instrument to help understand how students dealt with the learning environment during the treatment period. Higher interaction in one group, for example, may be related to a greater need to review resources, increased engagement with support activities, or the nature of the interface, which may require more attempts to understand formulas or conditions (Lim et al., 2023; Siemens & Long, 2011).

Therefore, including usability and interaction within the learning environment in the present study broadens the understanding of the effect of the interface pattern. The study does not only measure what students learned or produced; it also attempts to explain how they interacted with the learning

environment, how easy they found it to use, and how regularly they completed activities and returned to resources. This aligns with contemporary approaches to evaluating digital learning environments, which combine performance, attitudes, usability, and interaction data (Brooke, 1996; Gašević et al., 2015; ISO, 2018).

8.7 Summary of the Theoretical Framework

The preceding discussion shows that no-code development represents an appropriate approach for preparing educational technology students to produce interactive educational applications. It reduces barriers associated with textual programming and allows students to focus more on instructional design and interaction logic. It also shows that the interface pattern may play an important role in shaping the nature of learning. The block-based interface presents application logic in a visual form that may support practical performance, final product quality, and ease of tracing. By contrast, the logic-based interface requires deeper analysis of conditional relationships, formulas, and response flows, which may support logical thinking.

Accordingly, the present study is based on the assumption that differences in the no-code development interface pattern may lead to differences in cognitive achievement, practical performance, final product quality, logical thinking, attitude, usability, and interaction within the learning environment.

9. PREVIOUS STUDIES AND COMMENTARY

The previous studies were organized into axes aligned with the variables of the present study, rather than being presented as a separate narrative review. This organization aimed to clarify the position of the present study within the literature and to identify the research gap it addresses, particularly because the topic combines no-code development, development interface pattern, educational application development skills, logical thinking, usability, and interaction within the learning environment.

9.1 Studies on No-Code and Low-Code Development in Education

Sońta and Przegalińska (2023) examined the teaching of no-code and low-code development competencies to non-IT specialists. Their study emphasized that these platforms can help reduce barriers to entering digital solution development, especially when learning experiences are designed around real-world problems and applied tasks. Educational technology students, even if they are not specialists in textual programming, can be trained to produce educational applications through no-code tools, provided that the instructional treatment is carefully structured to match their technical and pedagogical background.

Tsakalerou and Xenos (2023) provided a comparative analysis of student success factors in low-code development platforms. They indicated that the educational value of these platforms does not lie only in the ease of building a product, but also in the learning experience accompanying development, including task clarity, support, activity progression, and students' understanding of the relationships among components and functions. This study supports the direction of the present research in measuring the effect of interface pattern across more than one variable, rather than limiting evaluation to the quality of the final product alone.

Lim et al. (2023) addressed learning experience and accessibility in low-code development platforms. Their study highlighted the importance of examining students' experiences while using these platforms in higher education, particularly in terms of ease of learning, interface clarity, and students' perception of the value of the platform in building functional applications.

Matook et al. (2023) investigated the development of metacognitive skills through low-code application development in a work-integrated learning context. They showed that application development requires learners to plan, monitor performance, analyze errors, and make continuous development decisions. These findings are closely related to the logic of the present study because building an educational application without programming is not limited to performing superficial steps; rather, it requires understanding the relationships among the objective, component, event, condition, and response.

Taken together, these studies show that no-code development environments represent a promising direction in education. At the same time, they reveal that the literature still needs experimental studies that compare interface patterns within these environments. Most previous studies focused on the platform or the general learning experience, whereas the present study addresses the development interface pattern as a specific independent variable by comparing the block-based interface with the logic-based interface among educational technology students.

9.2 Studies on Block-Based Interfaces and Visual Programming

Hu et al. (2021) conducted a meta-analysis on the

effectiveness of block-based visual programming in student learning. The results showed that this pattern can contribute to improving the learning of programming concepts, supporting computational thinking, and developing positive attitudes toward programming.

Çakiroğlu and Çevik (2022) examined abstraction as a sub-skill of computational thinking in block-based programming environments. They emphasized that these environments can help students build mental and procedural representations of solutions, especially when tasks are designed in a way that requires problem analysis and the construction of a logical sequence for the solution.

Sun et al. (2023) compared learning through block-based programming and text-based programming in terms of programming behaviors, computational thinking skills, and attitudes toward programming. Their findings indicated that the programming representation pattern can influence learners' behavior and interaction with the task. This supports the premise of the present study that the interface pattern is not merely an external form, but a factor that can change how students construct a solution.

Corral et al. (2021) presented a view of using block-based programming to enable students to acquire and transfer knowledge within an approach close to no-code development. Ruiz-Rube et al. (2019) also discussed the development of mobile learning experiences based on block-based programming. They showed that building educational applications through block-based environments can support the design of learning experiences connected to practical situations and mobile devices. Although this study is relatively older than the most recent literature, it is relevant to the present research because it is directly connected to building mobile educational applications using block-based environments.

These studies agree that the block-based interface offers learners a suitable visual entry point for learning how to build programming behavior. However, most of them did not compare a block-based interface with a logic-based interface in the context of educational application development among educational technology students. The present study therefore extends this direction by examining the effect of the block-based interface compared with the logic-based interface on achievement, performance, product quality, and logical thinking.

9.3 Studies on Computational and Logical Thinking in Development Environments

Irawan et al. (2023) examined trends, tools, and methods for teaching computational thinking through a systematic review of a large body of studies. They indicated that programming, especially visual and block-based environments, is frequently used to develop skills such as problem solving, sequencing, abstraction, and algorithmic thinking.

Yeni et al. (2023) addressed the integration of computational thinking and education in interdisciplinary contexts. They showed that tools such as Scratch and App Inventor appeared among the tools used in teaching computational thinking. This study supports the inclusion of MIT App Inventor in the present research as an appropriate tool for the block-based interface. It also supports viewing application development as an educational context that can promote thinking, rather than as merely a technical skill.

Lu et al. (2022) conducted a scoping review of computational thinking assessment in higher education. They reported that some studies used application design and development tasks, including mobile application tasks, as appropriate contexts for assessing computational thinking skills.

Weng et al. (2023) examined the integration of artificial intelligence and computational thinking in educational contexts. They indicated that having learners build applications or digital models can support multiple cognitive and practical outcomes, provided that the activities are designed in a way that makes students think about the operating logic of the system.

Aytekin and Topcu (2023) investigated the effect of plugged and unplugged computational thinking approaches on students' creative problem solving. They emphasized the importance of combining conceptual understanding with practical application. Although their sample and context differ from those of the present study, their findings support the idea that thinking is not developed through theoretical explanation alone. It requires performance situations in which learners analyze a problem, build a solution, and test it.

These studies inform the present research by supporting logical thinking as a main dependent variable. Students who develop an educational application are not merely practicing a technical skill; they are analyzing situations, arranging procedures, constructing conditions, inferring outcomes, and detecting errors. Nevertheless, the literature still needs studies that compare no-code development interface patterns in terms of their effect on this type

of thinking, which is the gap addressed by the present study.

9.4 Studies on Usability and Attitudes Toward Development Environments

Usability is an important factor in interpreting students' learning within development environments because interface difficulty or unclear instructions may limit students' ability to focus on the instructional task. Brooke (1996) introduced the System Usability Scale (SUS) as a brief and widely used instrument for estimating system usability from the user's perspective. ISO 9241-11 also emphasizes that usability is related to users' ability to achieve their goals effectively, efficiently, and satisfactorily within a specified context of use (Brooke, 1996; ISO, 2018).

Silva et al. (2021) examined the prediction of usability problems in low-code development platforms. They indicated that users may encounter cognitive difficulties when the relationship among components, properties, and procedures is not clear.

Torres and Kapralos (2023) examined the usability of a serious game authoring platform. Their study highlighted the importance of evaluating authoring and development tools from the user's perspective, especially when the aim is to enable non-specialists to produce educational resources or applications.

From the perspective of attitudes toward technology, the Technology Acceptance Model explains that perceived usefulness and perceived ease of use influence users' attitudes and their future intention to use technology (Davis, 1989). Similarly, the Unified Theory of Acceptance and Use of Technology emphasizes that performance expectancy, effort expectancy, facilitating conditions, and behavioral intention represent important dimensions of technology acceptance (Venkatesh et al., 2003). These models inform the construction of the Attitude Toward the No-Code Development Environment Scale in the present study, particularly its dimensions related to educational value, ease of use, motivation, confidence, and future intention.

These studies and models show that usability and attitude are not marginal variables in the present research. Rather, they help explain students' acceptance of and engagement with the development environment. A student may achieve good performance but may not develop a positive attitude toward the environment if it is perceived as complex or exhausting. Conversely, interaction may increase when students feel that the environment is useful, easy, and applicable to their future projects.

9.5 Studies on Learning Analytics and Interaction Within Learning Environments

Siemens and Long (2011) indicated that learning analytics represents an important approach for understanding what happens within digital learning environments, as it allows researchers to track learners' behavior, access patterns, and interaction with content and activities. This perspective supports the use of the Interaction Analysis Log within Moodle in the present study, as the study does not measure only final outcomes, but also seeks to understand students' behavior during learning.

Gašević et al. (2015) emphasized that learning analytics should remain connected to learning itself, rather than merely collecting digital data. They argued that activity logs should be interpreted in light of the instructional context, objectives, and nature of the task. This supports the way the Interaction Analysis Log is used in the present study as a supporting instrument for interpreting the findings, not as a substitute for tests or performance rubrics.

Recent studies on evaluating digital learning environments also suggest that interaction data can help researchers understand commitment, return to resources, assignment submission, and participation in activities. However, such data require careful interpretation because frequent access does not always mean better learning. It may sometimes reflect difficulty, hesitation, or repeated searching for help (Gašević et al., 2015; Siemens & Long, 2011).

In the context of the present study, the Interaction Analysis Log helps explain differences between the block-based and logic-based interface groups. Higher interaction in one group may reflect deeper engagement, greater need for support, or the nature of an interface that requires repeated review of resources. Therefore, interaction analysis enriches the interpretation of results, particularly when combined with achievement, practical performance, final product quality, and logical thinking.

9.6 Points of Agreement and Difference Between Previous Studies and the Present Study

The present study agrees with previous studies in emphasizing the importance of no-code and low-code development environments in enabling non-specialists to produce applied digital solutions, and in viewing block-based programming as a suitable entry point for novice learners. It also agrees with studies that linked application building or visual programming with the development of computational and logical thinking, as well as with

studies that emphasized the importance of usability and learning analytics in interpreting learners' experiences within digital environments.

The present study differs from most previous studies in that it does not examine a single tool or platform only, nor does it merely measure computational thinking in a general sense. Rather, it compares two no-code development interface patterns: the block-based interface and the logic-based interface. It also studies this comparison within the specific context of educational application development among educational technology students, a group that needs to combine instructional design, technical understanding, and applied production.

The present study is also distinguished by its use of multiple measurement instruments. It is not limited to an achievement test or an attitude scale; rather, it combines the Cognitive Achievement Test, Practical Performance Observation Checklist, Final Product Quality Evaluation Rubric, Logical Thinking Test, Attitude Scale, Learning Environment Usability Questionnaire, and Interaction Analysis Log. This diversity enables a more integrated understanding of the effect of interface pattern because it measures knowledge, performance, product quality, thinking, attitude, usability experience, and behavior within the learning environment.

The research gap lies in the fact that previous literature has addressed block-based programming, low-code/no-code development, and computational thinking through related but often separate tracks. The present study attempts to integrate these tracks into a single experimental design that compares two different interfaces for representing application-building logic and measures the effect of this difference on cognitive, practical, logical, and behavioral outcomes among educational technology students.

10. RESEARCH HYPOTHESES

In light of the research problem, objectives, theoretical framework, and previous studies, the following hypotheses were formulated:

10.1 First Hypothesis

There are statistically significant differences at the level of **p ≤ .05** between the mean scores of students in the first experimental group, who studied through the block-based interface, and those in the second experimental group, who studied through the logic-based interface, in the post-application of the Cognitive Achievement Test related to educational application development concepts, after controlling for the effect of the pre-measurement, in favor of the first experimental group.

10.2 Second Hypothesis

There are statistically significant differences at the level of **p ≤ .05** between the mean scores of students in the first experimental group and those in the second experimental group in the post-application of the Practical Performance Observation Checklist for educational application development skills, after controlling for the effect of the pre-measurement, in favor of the first experimental group.

10.3 Third Hypothesis

There are statistically significant differences at the level of **p ≤ .05** between the mean scores of students in the first experimental group and those in the second experimental group in the Final Product Quality Evaluation Rubric for the educational application, in favor of the first experimental group.

10.4 Fourth Hypothesis

There are statistically significant differences at the level of **p ≤ .05** between the mean scores of students in the first experimental group and those in the second experimental group in the post-application of the Logical Thinking Test, after controlling for the effect of the pre-measurement, in favor of the second experimental group.

10.5 Fifth Hypothesis

There are statistically significant differences at the level of **p ≤ .05** between the mean scores of students in the first experimental group and those in the second experimental group in the post-application of the Attitude Toward the No-Code Development Environment Scale, after controlling for the effect of the pre-measurement, in favor of the first experimental group.

10.6 Sixth Hypothesis

There are statistically significant differences at the level of **p ≤ .05** between the mean scores of students in the first experimental group and those in the second experimental group in the Learning Environment Usability Questionnaire for the No-Code Development-Based Learning Environment, in favor of the first experimental group.

10.7 Seventh Hypothesis

There are statistically significant differences at the level of **p ≤ .05** between the mean scores of students in the first experimental group and those in the second experimental group in the Learning

Interaction Analysis Log, in favor of the second experimental group.

11. RESEARCH METHODOLOGY AND PROCEDURES

This section presents the research method, experimental design, research population and sample, research variables, experimental treatment materials, measurement instruments, procedures for verifying their validity and reliability, steps for implementing the experiment, and statistical methods used to analyze the data.

11.1 Research Method

The present study adopted the quasi-experimental method because it was appropriate for the main objective of the study, namely examining the effect of the no-code development interface pattern, block-based versus logic-based, on developing educational application development skills and logical thinking among educational technology students. The quasi-experimental method is suitable for educational research in which full control over all variables, as in laboratory experiments, is difficult, while still allowing the researcher to control the independent variable, administer pre- and post-measurement instruments, and compare educational groups that are as equivalent as possible (Fraenkel et al., 2023).

This method was selected because the study did not aim merely to describe students' use of no-code development tools. Rather, it aimed to test the effect of two different development interface patterns on specific cognitive, practical, logical, and behavioral outcomes. Therefore, the nature of the study required the design of a structured instructional treatment, its implementation with two experimental groups, and the measurement of differences between them after the experiment, while controlling for the effect of pre-measurements in the instruments for which such control was required.

11.2 Experimental Design of the Study

The study used a two-equivalent-experimental-groups design with pre- and post-measurements for the instruments that measured growth before and after the treatment. Post-only measurement was used for instruments related to outcomes that could only appear after the end of the experiment, such as final product quality, usability, and the Interaction Analysis Log. This design is appropriate for quantitative research that aims to examine the effect of an independent variable on specific dependent variables within a structured educational context, while using pre- and post-measurements and controlling for covariates as far as possible (Creswell & Creswell, 2023; Fraenkel et al., 2023).

The experimental design consisted of two groups:

- The first experimental group studied through a block-based no-code development interface using MIT App Inventor.
- The second experimental group studied through a logic-based no-code development interface using Microsoft Power Apps.

Content, duration, objectives, activities, assignments, learning environment, and final product criteria were controlled so that the main difference between the two groups was limited to the no-code development interface pattern.

The experimental design can be represented as follows:

To examine the effect of the no-code development interface pattern, the study employed a quasi-experimental design with two equivalent experimental groups. Students were randomly assigned to the two groups before the administration of the pretests. The first group studied through the block-based interface using MIT App Inventor, whereas the second group studied through the logic-based interface using Microsoft Power Apps. Pre- and post-measurements were used for variables that required baseline control, while post-only measures were used for final product quality, usability, and learning environment interaction. The experimental design is illustrated in Figure 3.

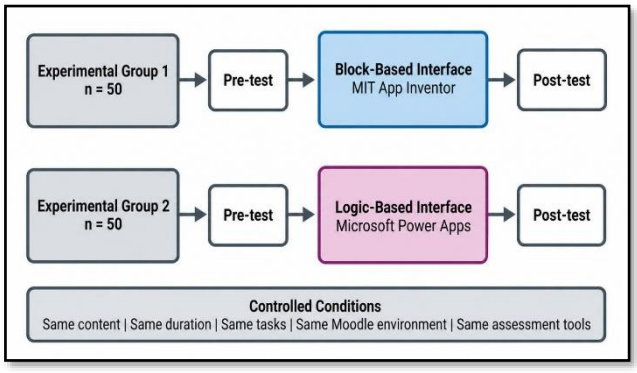

Figure 3. Quasi-Experimental Design of the Study.

Figure 3 shows that the two experimental groups followed the same instructional sequence and were exposed to the same controlled conditions. The only systematic difference between the two groups was the no-code development interface pattern. This design allowed the study to compare the two interface patterns while controlling for content, duration, tasks, Moodle environment, assessment tools, and final product criteria.

11.3 Research Population and Sample

The research population consisted of students in the Department of Educational Technology, Faculty of Specific Education, Alexandria University. The main research sample consisted of 100 third-year students from the Department of Educational Technology. They were randomly assigned, before the pre-application of the instruments, to two equivalent experimental groups, with 50 students in the first experimental group and 50 students in the second experimental group.

Third-year students were selected because they study a course related to educational software design and production, and because they have a general background in using e-learning environments and digital resources, while having limited prior experience in developing educational applications without programming, as indicated by the Preliminary Data and Prior Experience Form. This helped make the sample suitable for studying the effect of interface pattern without attributing the differences to advanced prior experience in no-code development tools.

The study also used a pilot sample of 20 students from outside the main sample. This pilot sample was used to verify the clarity of the measurement instruments, estimate the time required for application, calculate the psychometric properties of the instruments, and make the necessary revisions before the main application.

11.4 Research Variables

The study included the following variables:

11.4.1 Independent Variable

The independent variable was the no-code development interface pattern, with two levels:

1. The block-based interface using MIT App Inventor.
2. The logic-based interface using Microsoft Power Apps.

11.4.2 Dependent Variables

The dependent variables were:

1. Cognitive achievement related to educational application development concepts.
2. Practical performance of educational application development skills.
3. Quality of the final educational application.
4. Logical thinking.
5. Attitude toward the no-code development environment.
6. Usability of the no-code development-based learning environment.
7. Interaction within the learning environment.

11.4.3 Control Variables

The study controlled a number of variables that could affect the results, including:

- The instructional content provided to both groups.
- The duration of the experiment.
- The number of activities and assignments.
- The nature of the required educational product.
- The criteria for evaluating the final application.
- The learning management platform used.
- The measurement instruments.
- Students' academic level.
- Random assignment to the two groups.
- Verification of pre-treatment equivalence between the two groups before implementing the treatment.

11.5 Experimental Treatment Materials

The experimental treatment materials consisted of a structured e-learning environment delivered through Moodle and two equivalent training paths in terms of content and activities. The only difference between the two paths was the no-code development interface pattern used to build the educational application.

11.5.1 Moodle Environment

Moodle was used as the main environment for organizing the experimental treatment. It included weekly learning units, explanation files, training videos, tool links, instructions, staged assignments, task submission areas, and support resources. The environment logs were also used to track students' interaction with content, activities, and resources, which supported the analysis of interaction levels within the learning environment.

The environment was designed to be organized, clear in its instructions, gradual in its presentation of content, and equivalent across the two groups. Accordingly, all students received the same objectives, content, and assignments, while the only difference was the development tool and interface pattern used.

11.5.2 Block-Based Interface Path

Students in the first experimental group used MIT App Inventor as a no-code development environment based on the block-based interface. This path trained students to design application screens, add components, adjust their properties, and build application behavior using blocks that represented events, conditions, variables, and responses.

At the end of this path, each student was required to produce a small-scale interactive educational application that included a start screen, objectives, instructional content, an interactive activity, a short assessment, feedback, and a score or final result.

11.5.3 Logic-Based Interface Path

Students in the second experimental group used Microsoft Power Apps as a no-code development environment based on the logic-based interface. This path trained students to build screens, add controls, adjust properties, and use formulas, conditions, and logic to determine application behavior, such as navigation between screens, response verification, score calculation, and feedback display.

This path adhered to the same instructional objective and the same final task so that the comparison between the two groups would be based on the interface pattern, not on differences in the nature of the required product.

11.6 Experimental Treatment Implementation Plan

The experimental treatment lasted for eight weeks during the first semester of the 2022/2023 academic year, beginning on Saturday, October 5, 2023.

The experimental treatment lasted for eight weeks during the first semester of the 2022/2023 academic year, beginning on October 5, 2023. The treatment was organized progressively, moving from orientation and pretesting to no-code development concepts, application planning, interface design, interaction logic, assessment and feedback design, testing, debugging, final submission, and posttesting. Figure 4 presents the weekly timeline of the experimental treatment.

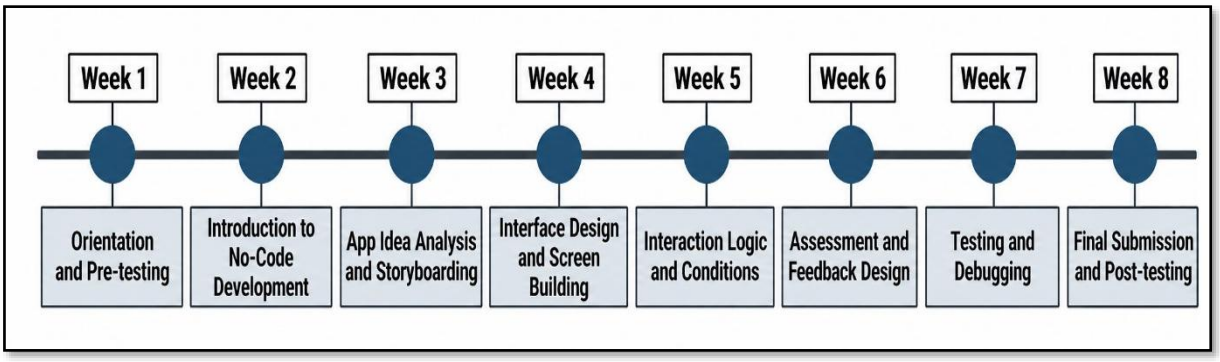

Figure 4. Eight-Week Timeline of the Experimental Treatment.

As presented in Figure 4, the treatment was sequenced to support gradual skill development. Students first became familiar with the Moodle environment and the study requirements, then moved through the stages of educational application development. This progression ensured that both groups completed equivalent learning tasks while using different no-code development interface patterns.

The following table shows the organization of the weekly plan.

Table 1. Weekly Implementation Plan.

Week	Implementation Topic	Interim Learning Outcome
First	Orientation, introduction to the Moodle environment, and administration of the pre-measurement instruments	Students accessed the environment, and the pretests and pre-scales were administered
Second	Introduction to no-code development and educational applications	Identifying the educational application idea and the target group
Third	Analysis of the application idea and design of the instructional scenario	Preparing an initial screen map and learning pathway
Fourth	Designing the application interface and adding components	Building an initial version of the application screens
Fifth	Building interaction and application logic	Implementing an event, condition, or response formula within the application
Sixth	Designing assessment and feedback	Building a short assessment activity and immediate feedback
Seventh	Testing the application and debugging	Preparing a semi-final executable version
Eighth	Submitting the final product and administering the post-measurement instruments	Final application, post-measurement scores, and interaction logs

Both groups followed the same timetable and received the same instructional content, while examples and training activities were presented in accordance with the interface pattern used by each group.

12. RESEARCH INSTRUMENTS

The study used eight instruments: the Preliminary Data and Prior Experience Form, the Cognitive Achievement Test, the Logical Thinking Test, the Practical Performance Observation Checklist, the Final Product Quality Evaluation Rubric, the Attitude Toward the No-Code Development Environment Scale, the Learning Environment Learning Environment Usability Questionnaire, and the Learning Interaction Analysis Log.

12.1 Preliminary Data and Prior Experience Form

The researchers developed the Preliminary Data and Prior Experience Form to describe the characteristics of the research sample and identify students' prior experience in using e-learning environments, developing applications or interactive activities, and using no-code development tools. The form included demographic and academic data, as well as questions about previous experience in using learning platforms, designing applications or interactive activities, and using block-based or logic-based interfaces.

The form included multiple-choice questions, yes/no questions, and self-rating items, with the possibility of coding some responses numerically for analysis.

12.2 Cognitive Achievement Test

The researchers developed a Cognitive Achievement Test to measure educational technology students' possession of the knowledge, concepts, and basic procedures associated with educational application development in no-code development environments. In its final form, the test consisted of 30 multiple-choice items. Each item had four alternatives, with only one correct answer. The maximum score of the test was 30.

The test items were distributed across six dimensions: no-code development and educational application concepts; analysis and planning of the educational application idea; educational user interface design; building application components and functions; developing application logic, interaction, and feedback; and testing, publishing, and evaluating the quality of the application. The test was administered before and after the treatment to measure the effect of the experimental treatment on cognitive achievement.

12.3 Logical Thinking Test

The researchers developed a Logical Thinking Test to measure students' ability to analyze educational application development situations according to logical relationships among the event, condition, action, and outcome. The test consisted of 30 situational items distributed across six dimensions, with a maximum score of 30.

The test dimensions were: logical sequencing of procedures, conditional reasoning, cause-and-effect analysis, detection of logical errors, organization of data and variables, and logical decision-making in application development situations. The test was administered before and after the treatment to examine the effect of the no-code development interface pattern on developing logical thinking.

12.4 Practical Performance Observation Checklist

The researchers developed the Practical Performance Observation Checklist to measure students' performance while carrying out no-code educational application development skills. In its final form, the checklist consisted of 45 indicators distributed across seven dimensions. The maximum score was 135, as each skill was rated using a four-level scale ranging from 0 to 3.

The checklist dimensions included: analyzing and planning the application idea, designing the user interface, building screens and components, developing interaction and application logic, designing assessment and feedback, testing and debugging the application, and adhering to work procedures and task submission requirements. The checklist was used to measure students' practical performance while they were carrying out application development tasks.

12.5 Final Product Quality Evaluation Rubric

The researchers developed the Final Product Quality Evaluation Rubric to judge the quality of the educational application produced by students after the treatment. The rubric consisted of 40 indicators distributed across seven dimensions. The maximum score was 120, as each indicator was rated using a four-level scale ranging from 0 to 3.

The rubric dimensions included: quality of the instructional design of the application, quality and organization of educational content, quality of the user interface and user experience, quality of interaction and application logic, quality of assessment and feedback, technical quality and operability, and quality of final output and submission. The rubric was administered only after the treatment because the final product appears only after the completion of the experimental treatment.

12.6 Attitude Toward the No-Code Development Environment Scale

The researchers developed the Attitude Toward the No-Code Development Environment Scale to identify educational technology students' attitudes toward using the environment in developing educational applications. The scale consisted of 30 statements distributed across five dimensions: perceived educational value, perceived ease of use, motivation and enjoyment of learning, confidence in the ability to develop educational applications, and future intention to use no-code development environments.

The scale statements were answered using a five-point Likert scale, and the maximum score of the scale was 150. The instrument included five negative statements, namely items 6, 12, 18, 24, and 30. Their scores were reversed during scoring, so that a higher score indicated a more positive attitude toward the no-code development environment.

12.7 Learning Environment Learning Environment Usability Questionnaire

The researchers developed the Learning Environment Usability Questionnaire for the No-Code Development-Based Learning Environment to identify the extent to which the learning environment was appropriate from students' perspective after using it to complete application development activities. The questionnaire consisted of 30 statements distributed across six dimensions: ease of access and starting to use the environment, clarity of content and activity organization, ease of navigation within the environment, clarity of application development task instructions, adequacy of support and assistance within the environment, and overall satisfaction with using the environment.

The questionnaire statements were answered using a five-point Likert scale, and the maximum score was 150. The questionnaire included six negative statements, namely items 5, 10, 15, 20, 25, and 30. Their scores were reversed during scoring, so that a higher score indicated greater perceived usability of the learning environment from the students' perspective.

12.8 Learning Interaction Analysis Log

The researchers developed a Learning Interaction Analysis Log to measure students' regularity and interaction with content, activities, support resources, discussion spaces, and task submissions. The log consisted of 30 indicators distributed across six dimensions, with a maximum score of 90.

The log was used after the treatment based on interaction data within Moodle. It was not used as a substitute for the achievement, performance, or thinking instruments, but as a supporting instrument that helped interpret students' level of participation during the treatment.

13. VALIDITY AND RELIABILITY OF THE RESEARCH INSTRUMENTS

13.1 Expert Judgment Validity

The research instruments, in their initial form, were submitted to a group of specialists in educational technology, computer science, and e-learning. This was done to verify content validity, clarity of wording, relevance of items and indicators to the study objectives and variables, and the appropriateness of the application and scoring instructions.

The overall agreement percentages among the reviewers ranged from 89.5% to 98%, which were considered appropriate for approving the instruments after making the suggested revisions. The following table presents the agreement percentages and validation decisions.

Table 2. Expert Agreement Percentages and Validation Decisions for the Research Instruments.

No.	Research Instrument	Overall Agreement Percentage	Validation Decision
1	Cognitive Achievement Test	94.5%	Suitable for application after minor revisions
2	Practical Performance Observation Checklist	92.7%	Suitable for application after merging and revising some indicators
3	Final Product Quality Evaluation Rubric	95.4%	Suitable for application in nearly its final form
4	Logical Thinking Test	91.8%	Suitable for application after replacing two situations to improve clarity
5	Attitude Toward the No-Code Development Environment Scale	96.3%	Suitable for application after rewording one negative statement
6	Learning Environment Usability Questionnaire	95.0%	Suitable for application
7	Student Interaction Analysis Log	89.5%	Suitable for application after clarifying the sources of some evidence
8	Preliminary Data Form	98.0%	Suitable for application

Based on the reviewers' comments, some item wordings were revised, closely related indicators were merged, two situations in the Logical Thinking Test were replaced, the evidence sources in the Interaction Log were clarified, and the negative statements in the scales were reviewed.

13.2 Internal Consistency Validity

The internal consistency validity of the research instruments was verified by administering them to a pilot sample from outside the main sample. Pearson correlation coefficients were then calculated between each item or indicator and the dimension to which it belonged, as well as between each dimension and the total score of the instrument.

For the Cognitive Achievement Test, the item-dimension correlation coefficients ranged from 0.45 to 0.78, while the correlations between the dimensions and the total score ranged from 0.71 to 0.84. All coefficients were statistically significant at the .01 and .05 levels, indicating that the test had an appropriate degree of internal consistency.

For the Logical Thinking Test, the item-dimension correlation coefficients ranged from 0.48 to 0.81, while the correlations between the dimensions and the total score ranged from 0.75 to 0.86. These statistically significant coefficients indicate that the test items were appropriately connected to the dimensions they measured.

As for the Practical Performance Observation Checklist, the correlations between the indicators and their dimensions ranged from 0.52 to 0.85. The correlation coefficients in the Final Product Quality Evaluation Rubric ranged from 0.55 to 0.88, and those in the Interaction Analysis Log ranged from 0.50 to 0.82. All were positive and statistically significant at the .01 level.

For the Attitude Toward the No-Code Development Environment Scale, the correlations between the statements and their dimensions ranged from 0.58 to 0.89. The correlations in the Learning Environment Usability Questionnaire ranged from 0.61 to 0.90 after reversing the scores of the negative statements. These results indicate that the statements were consistent with their respective dimensions.

13.3 Difficulty, Ease, and Discrimination Indices

The ease, difficulty, and discrimination indices were calculated for the items of the Cognitive Achievement Test and the Logical Thinking Test to verify their appropriateness for the students' level.

The ease indices of the Cognitive Achievement Test items ranged from 0.35 to 0.75, with an average of 0.55. The ease indices of the Logical Thinking Test items ranged from 0.30 to 0.70, with an average of 0.52. These values indicate that the items in both tests fell within the acceptable range, as they were neither excessively easy nor excessively difficult.

The discrimination indices in the Cognitive Achievement Test ranged from 0.35 to 0.65, while those in the Logical Thinking Test ranged from 0.40 to 0.75. All indices exceeded the minimum acceptable level, indicating that the items were able to discriminate between students with high performance and those with low performance.

13.4 Reliability of the Research Instruments

The reliability of the research instruments was verified using reliability coefficients appropriate to the nature of each instrument. For the two objective tests, the Kuder–Richardson Formula 20 (KR-20) was used, along with Cronbach's alpha, because the item responses were dichotomously scored. The reliability coefficient of the Cognitive Achievement Test was 0.86, and that of the Logical Thinking Test was 0.88. These values indicate high reliability and support the suitability of the two tests for the main application.

For the rating-based instruments, namely the Practical Performance Observation Checklist, the Final Product Quality Evaluation Rubric, and the Interaction Analysis Log, Cronbach's alpha was calculated. The coefficient reached 0.91 for the observation checklist, 0.93 for the product evaluation rubric, and 0.89 for the interaction analysis log. These values indicate a high level of internal consistency.

For the Attitude Toward the No-Code Development Environment Scale, Cronbach's alpha for the scale as a whole was 0.92, and the split-half reliability coefficient according to Spearman–Brown was 0.90. As for the Learning Environment Usability Questionnaire, Cronbach's alpha was 0.94, and the Spearman–Brown coefficient was 0.91. These high values indicate the stability and internal consistency of both instruments.

Inter-rater reliability was also verified for the Practical Performance Observation Checklist and the Final Product Quality Evaluation Rubric. The researcher and another observer jointly rated the performance of five students from the pilot sample. The agreement percentage, calculated using Cooper's formula, was 88.5% for the observation checklist and 91.2% for the product quality rubric. These values indicate an appropriate level of objectivity in the rating process.

14. RESEARCH IMPLEMENTATION PROCEDURES

The implementation of the study proceeded

through the following steps:

1. Reviewing the literature and previous studies related to no-code development, block-based programming, logic-based interfaces, educational application development, logical thinking, usability, and learning analytics.
2. Identifying the educational application development skills required for educational technology students in light of the nature of the Educational Software Design and Production course and the characteristics of no-code development environments.
3. Developing the experimental treatment in two forms: a path based on the block-based interface using MIT App Inventor, and a path based on the logic-based interface using Microsoft Power Apps.
4. Designing the Moodle environment and organizing the content, activities, assignments, and support resources in an equivalent manner for the two groups.
5. Preparing the research instruments in their initial form, including the achievement test, logical thinking test, practical performance observation checklist, final product quality evaluation rubric, attitude scale, Learning Environment Usability Questionnaire, interaction log, and preliminary data form.
6. Submitting the research instruments to specialists in educational technology, computer science, and e-learning, and making the revisions they recommended.
7. Administering the instruments to the pilot sample to calculate their psychometric properties, estimate the application time, and ensure the clarity of instructions.
8. Randomly assigning students to the two research groups before the pre-application of the instruments.
9. Administering the pre-measurements to the two groups, including the Cognitive Achievement Test, Logical Thinking Test, Practical Performance Observation Checklist, and Attitude Scale.
10. Verifying the pre-treatment equivalence of the two groups in the main variables before implementing the experimental treatment.
11. Implementing the experimental treatment for eight weeks, beginning on Saturday, October 5, 2023.
12. Monitoring students' commitment to the activities and assignments within Moodle, and providing instructional support according to unified guidelines.
13. Administering the post-measurements after the completion of the treatment, including the Cognitive Achievement Test, Logical Thinking Test, Practical Performance Observation Checklist, Attitude Scale, Learning Environment Usability Questionnaire, Final Product Quality Evaluation Rubric, and Interaction Analysis Log.
14. Scoring the instruments according to the approved scoring keys, while reversing the scores of the negative statements in the Attitude Scale and the Learning Environment Usability Questionnaire.
15. Entering the data into SPSS, reviewing the scores, and checking that the data were free from entry or coding errors.
16. Conducting the appropriate statistical analyses and interpreting the results in light of the theoretical framework and previous studies.

15. STATISTICAL METHODS USED

SPSS was used to process the data. The following statistical methods were employed:

1. Means and standard deviations were used to describe students' performance on the research instruments.
2. Frequencies and percentages were used to describe the data obtained from the prior experience form and some sample characteristics.
3. The independent-samples t-test was used to verify the pre-treatment equivalence of the two groups.
4. The independent-samples t-test was used to compare the two groups in the instruments that were administered post-treatment only, such as the Final Product Quality Evaluation Rubric, the Learning Environment Usability Questionnaire, and the Interaction Analysis Log.
5. The paired-samples t-test was used when needed to compare pre- and post-measurements within each group.
6. Analysis of covariance (ANCOVA) was used to compare the two groups in the post-measurements of instruments that had both pre- and post-measurements, while controlling for the effect of the pre-measurement.
7. Effect size was calculated using Eta squared or partial Eta squared, depending on the type of analysis.
8. Cohen's d was calculated in pairwise comparisons to estimate the magnitude of differences between means.
9. Reliability coefficients were calculated using KR-20, Cronbach's alpha, and split-half reliability,

according to the nature of the instrument.

10. The assumptions of statistical analysis were examined, including normality of distribution, homogeneity of variance, and homogeneity of regression slopes when using ANCOVA.

16. ETHICAL CONSIDERATIONS

The research adhered to the ethical principles of educational research. Students were informed that participation in the study was for scientific research purposes only, and that the data collected would be treated confidentially and used in aggregate form. They were also informed that participation in the research instruments would not affect their course grades.

Students' names were replaced with numerical codes when entering and analyzing the data. The study also ensured that both groups received equivalent instructional support, that neither group was deprived of the basic course content or activities, and that all participants had access to the instructional materials and support resources within Moodle.

17. RESULTS

This section presents the results of the study in light of its hypotheses, after the experimental treatment had been implemented with the two research groups, the measurement instruments had been scored according to the approved scoring keys, and the data had been entered into SPSS. The data were analyzed using descriptive statistics, the independent-samples t-test to verify pre-treatment equivalence and to conduct post-treatment comparisons for instruments administered only after the treatment, and analysis of covariance (ANCOVA) for instruments administered before and after the treatment, with effect sizes calculated.

Before conducting ANCOVA, the assumptions of normality, homogeneity of variance, and homogeneity of regression slopes were examined. The results indicated that the assumptions were sufficiently met for the instruments analyzed using ANCOVA, as the group × pre-measurement interaction was non-significant. Accordingly, ANCOVA was considered appropriate for comparing the two groups after controlling for the pre-measurement.

17.1 Verification of Pre-Treatment Equivalence Between the Two Research Groups

Before implementing the experimental treatment, the equivalence of the two groups was verified in the pre-measurements of the instruments administered before the treatment: the Cognitive Achievement Test, the Logical Thinking Test, the Practical Performance Observation Checklist, and the Attitude Toward the No-Code Development Environment Scale. The following table presents the results of the independent-samples t-test.

Table 3. Results of the Independent-Samples t-Test for Pre-Treatment Equivalence Between the Two Groups.

Instrument	Group	N	Pre-Mean	Standard Deviation	t	p
Cognitive Achievement Test	Block-based interface	50	8.56	1.94	-0.16	.876
	Logic-based interface	50	8.62	1.89		
Logical Thinking Test	Block-based interface	50	8.56	1.94	-0.16	.876
	Logic-based interface	50	8.62	1.89		
Practical Performance Observation Checklist	Block-based interface	50	47.92	2.61	-0.15	.878
	Logic-based interface	50	48.00	2.58		
Attitude Toward the No-Code Development Environment Scale	Block-based interface	50	86.92	3.21	0.09	.926
	Logic-based interface	50	86.86	3.23		

The table shows that all p-values for pre-treatment differences were greater than .05. This indicates that there were no statistically significant differences between the two groups before the treatment. Accordingly, the two groups were equivalent in cognitive achievement, logical thinking, practical performance, and attitude toward the no-code development environment before the experimental treatment was implemented.

17.2 Result of the First Hypothesis: The Effect of Interface Pattern on Cognitive Achievement

The first hypothesis stated that there would be statistically significant differences at the level of **p ≤**

.05 between the mean scores of students in the first experimental group, who studied through the block-based interface, and those in the second experimental group, who studied through the logic-based interface, in the post-application of the Cognitive Achievement Test related to educational application development concepts, after controlling for the effect of the pre-measurement, in favor of the first experimental group.

The following table presents the means and standard deviations of students' scores on the Cognitive Achievement Test.

Table 4. Means and Standard Deviations of Students' Scores on the Cognitive Achievement Test.

Group	N	Pre-Mean	Pre-SD	Post-Mean	Post-SD
Block-based interface	50	8.56	1.94	26.04	1.98
Logic-based interface	50	8.62	1.89	23.66	2.33

The table shows that the post-mean score of the group that studied through the block-based interface was 26.04, whereas the post-mean score of the group that studied through the logic-based interface was 23.66. This indicates an apparent difference in favor of the block-based interface group.

To verify the statistical significance of this difference after controlling for the effect of the pre-measurement, ANCOVA was used, as shown in the following table.

Table 5. ANCOVA Results for the Cognitive Achievement Test.

Source of Variance	Sum of Squares	df	Mean Square	F	p	Effect Size Partial η^2
Pre-measurement	368.79	1	368.79	404.91	< .001	–
Interface pattern	148.89	1	148.89	163.47	< .001	.628
Error	88.35	97	0.91	–	–	–

The results indicate a statistically significant difference between the two groups in the post-application of the Cognitive Achievement Test after controlling for the effect of the pre-measurement. The F value for interface pattern was 163.47, which was significant at **$p < .001$**. The effect size value was .628, which is a high value indicating that the no-code development interface pattern had a clear effect on cognitive achievement. Accordingly, the first hypothesis was supported in favor of the group that studied through the block-based interface.

17.3 Result of the Second Hypothesis: The Effect of Interface Pattern on Practical Performance

The second hypothesis stated that there would be statistically significant differences at the level of **$p \leq .05$** between the mean scores of students in the first experimental group and those in the second experimental group in the post-application of the Practical Performance Observation Checklist for educational application development skills, after controlling for the effect of the pre-measurement, in favor of the first experimental group.

The following table presents the means and standard deviations of students' scores on the Practical Performance Observation Checklist.

Table 6. Means and Standard Deviations of Students' Scores on the Practical Performance Observation Checklist.

Group	N	Pre-Mean	Pre-SD	Post-Mean	Post-SD
Block-based interface	50	47.92	2.61	117.46	3.89
Logic-based interface	50	48.00	2.58	104.38	3.70

The table shows that the post-mean score of the block-based interface group in practical performance was 117.46 out of 135, whereas the mean score of the logic-based interface group was 104.38. This indicates an apparent difference in favor of the block-based interface.

To verify the statistical significance of this difference after controlling for the effect of the pre-measurement, ANCOVA was used, as shown below.

Table 7. ANCOVA Results for the Practical Performance Observation Checklist.

Source of Variance	Sum of Squares	df	Mean Square	F	p	Effect Size Partial η^2
Pre-measurement	1327.61	1	1327.61	1522.34	< .001	–
Interface pattern	4350.65	1	4350.65	4988.81	< .001	.981
Error	84.59	97	0.87	–	–	–

The results indicate a statistically significant difference between the two groups in post-treatment practical performance after controlling for the effect of the pre-measurement. The F value for interface pattern was 4988.81, which was significant at **p < .001**. The effect size value was .981, which is very high and indicates a strong effect of interface pattern on practical performance. Accordingly, the second hypothesis was supported in favor of the group that studied through the block-based interface.

17.4 Result of the Third Hypothesis: The Effect of Interface Pattern on Final Product Quality

The third hypothesis stated that there would be statistically significant differences at the level of **p ≤ .05** between the mean scores of students in the first experimental group and those in the second experimental group on the Final Product Quality Evaluation Rubric for the educational application, in favor of the first experimental group.

Because final product quality was not measured before the treatment, the independent-samples t-test was used to compare the two groups in the post-application of the Final Product Quality Evaluation Rubric.

Table 8. Results of the Independent-Samples t-Test for the Final Product Quality Evaluation Rubric.

Group	N	Mean	Standard Deviation	t	df	p	Effect Size η²
Block-based interface	50	105.12	4.74	11.42	98	< .001	.571
Logic-based interface	50	95.36	3.75				

The table shows that the mean score of the block-based interface group in final product quality was 105.12 out of 120, whereas the mean score of the logic-based interface group was 95.36. The t value was 11.42, which was significant at **p < .001**, and the effect size was .571, indicating a high effect size. Accordingly, the third hypothesis was supported in favor of the group that studied through the block-based interface.

17.5 Result of the Fourth Hypothesis: The Effect of Interface Pattern on Logical Thinking

The fourth hypothesis stated that there would be statistically significant differences at the level of **p ≤ .05** between the mean scores of students in the first experimental group and those in the second experimental group in the post-application of the Logical Thinking Test, after controlling for the effect of the pre-measurement, in favor of the second experimental group.

The following table presents the means and standard deviations of students' scores on the Logical Thinking Test.

Table 9. Means and Standard Deviations of Students' Scores on the Logical Thinking Test.

Group	N	Pre-Mean	Pre-SD	Post-Mean	Post-SD
Block-based interface	50	8.56	1.94	23.36	1.78
Logic-based interface	50	8.62	1.89	26.52	1.73

The table shows that the post-mean score of the logic-based interface group on the Logical Thinking Test was 26.52, whereas the post-mean score of the block-based interface group was 23.36. This indicates an apparent difference in favor of the logic-based interface.

To verify the statistical significance of this difference after controlling for the effect of the pre-measurement, ANCOVA was used, as shown below.

Table 10. ANCOVA Results for the Logical Thinking Test.

Source of Variance	Sum of Squares	df	Mean Square	F	p	Effect Size Partial η²
Pre-measurement	268.25	1	268.25	770.96	< .001	—
Interface pattern	241.46	1	241.46	693.98	< .001	.877
Error	33.75	97	0.35	—	—	—

The results indicate a statistically significant difference between the two groups in logical thinking after controlling for the effect of the pre-measurement. The F value for interface pattern was 693.98, which was significant at **p < .001**. The effect size value was .877, which is high and indicates a strong effect of interface pattern on logical thinking. Accordingly, the fourth hypothesis was supported in favor of the group that studied through the logic-based interface.

17.6 Result of the Fifth Hypothesis: The Effect of Interface Pattern on Attitude Toward the No-Code Development Environment

The fifth hypothesis stated that there would be statistically significant differences at the level of **$p \leq .05$** between the mean scores of students in the first experimental group and those in the second experimental group in the post-application of the Attitude Toward the No-Code Development Environment Scale, after controlling for the effect of the pre-measurement, in favor of the first experimental group.

The following table presents the means and standard deviations of students' scores on the Attitude Scale.

Table 11. Means and Standard Deviations of Students' Scores on the Attitude Toward the No-Code Development Environment Scale.

Group	N	Pre-Mean	Pre-SD	Post-Mean	Post-SD
Block-based interface	50	86.92	3.21	137.16	3.57
Logic-based interface	50	86.86	3.23	126.52	2.99

The table shows that the post-mean attitude score of students in the block-based interface group was 137.16 out of 150, whereas the mean score of the logic-based interface group was 126.52. This indicates an apparent difference in favor of the block-based interface.

To verify the statistical significance of this difference after controlling for the effect of the pre-measurement, ANCOVA was used, as shown below.

The results indicate a statistically significant difference between the two groups in attitude toward the no-code development environment after controlling for the effect of the pre-measurement. The F value for interface pattern was 8957.91, which was significant at **$p < .001$**. The effect size was .989, which is a very high effect size. Accordingly, the fifth hypothesis was supported in favor of the group that studied through the block-based interface.

Table 12. ANCOVA Results for the Attitude Toward the No-Code Development Environment Scale.

Source of Variance	Sum of Squares	df	Mean Square	F	p	Effect Size Partial η^2
Pre-measurement	1030.90	1	1030.90	3300.53	< .001	—
Interface pattern	2797.96	1	2797.96	8957.91	< .001	.989
Error	30.30	97	0.31	—	—	—

17.7 Result of the Sixth Hypothesis: The Effect of Interface Pattern on Learning Environment Usability

The sixth hypothesis stated that there would be statistically significant differences at the level of **$p \leq .05$** between the mean scores of students in the first experimental group and those in the second experimental group on the Learning Environment Usability Questionnaire for the No-Code Development-Based Learning Environment, in favor of the first experimental group.

Because the Learning Environment Usability Questionnaire was administered after students had completed their use of the learning environment, the independent-samples t-test was used to compare the two groups post-treatment.

Table 13. Results of the Independent-Samples t-Test for the Learning Environment Learning Environment Usability Questionnaire.

Group	N	Mean	Standard Deviation	t	df	p	Effect Size η^2
Block-based interface	50	138.74	3.16	28.86	98	< .001	.895
Logic-based interface	50	117.46	4.15				

The table shows that the mean score of the block-based interface group on the Learning Environment Learning Environment Usability Questionnaire was 138.74, whereas the mean score of the logic-based interface group was 117.46. The t value was 28.86, which was significant at **$p < .001$**, and the effect size was .895, which is very high. Accordingly, the sixth hypothesis was supported in favor of the group that studied through the block-based interface.

17.8 Result of the Seventh Hypothesis: The Effect of Interface Pattern on Interaction Within the Learning Environment

The seventh hypothesis stated that there would be statistically significant differences at the level of **$p \leq .05$** between the mean scores of students in the first experimental group and those in the second experimental group in the Learning Interaction Analysis Log, in favor of the second experimental group.

The independent-samples t-test was used to compare the two groups in their scores on the Learning Interaction Analysis Log, as shown in the following table.

Table 14. Results of the Independent-Samples t-Test for the Learning Interaction Analysis Log.

Group	N	Mean	Standard Deviation	t	df	p	Effect Size η²
Block-based interface	50	73.12	3.31	-6.34	98	< .001	.291
Logic-based interface	50	77.24	3.19				

The table shows that the mean score of the logic-based interface group in the Interaction Log was 77.24 out of 90, whereas the mean score of the block-based interface group was 73.12. The t value was -6.34, which was significant at **p < .001**, and the effect size was .291, indicating a moderate-to-high effect size. Accordingly, the seventh hypothesis was supported in favor of the group that studied through the logic-based interface.

17.9 Summary of Hypothesis Results

The following table summarizes the results of the research hypotheses.

Table 15. Summary of the Research Hypothesis Results.

Hypothesis	Dependent Variable	Direction of the Result	Decision
First	Cognitive achievement	In favor of the block-based interface	Supported
Second	Practical performance	In favor of the block-based interface	Supported
Third	Final product quality	In favor of the block-based interface	Supported
Fourth	Logical thinking	In favor of the logic-based interface	Supported
Fifth	Attitude toward the no-code development environment	In favor of the block-based interface	Supported
Sixth	Learning environment usability	In favor of the block-based interface	Supported
Seventh	Interaction within the learning environment	In favor of the logic-based interface	Supported

The results show that the block-based interface had a stronger effect on cognitive achievement, practical performance, final product quality, attitude toward the development environment, and learning environment usability. In contrast, the logic-based interface outperformed the block-based interface in logical thinking and interaction within the learning environment. These findings indicate that the difference between the two no-code development interface patterns did not produce a single uniform direction of effect. Rather, the effect varied according to the nature of the dependent variable. Variables related to ease of performance and product construction favored the block-based interface, whereas variables related to logical analysis and intensity of interaction favored the logic-based interface.

The high effect sizes can be interpreted in light of the wide differences between the post-measurement means and the relatively low dispersion within each group. They are also consistent with the nature of the treatment, which offered two different experiences in representing application logic: one visual and block-based, and the other logic-based and grounded in formulas, conditions, and response flows.

To provide an overall visual summary of the magnitude and direction of the differences between the two experimental groups, standardized mean differences were calculated using Cohen's d. Positive values indicate that the outcome favored the block-based interface group, whereas negative values indicate that the outcome favored the logic-based interface group. Figure 5 presents the standardized mean differences across the seven dependent variables.

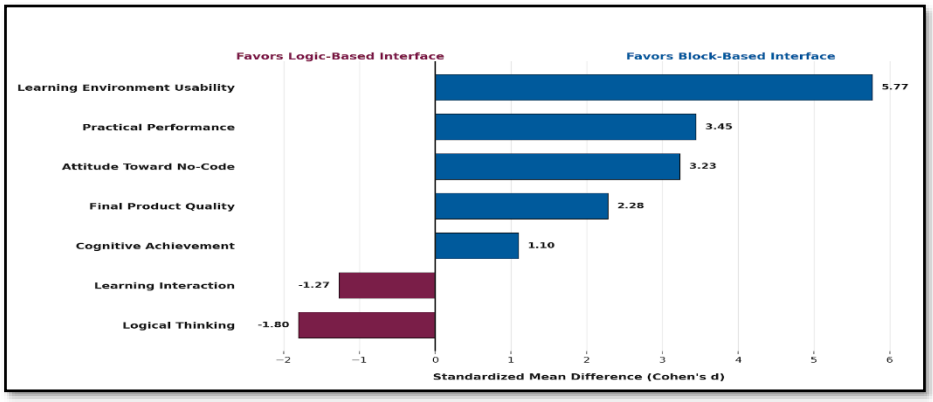

Figure 5. Standardized Mean Differences Between the Block-Based and Logic-Based Interface Groups.
Note. Positive values favor the block-based interface, whereas negative values favor the logic-based interface.

Figure 5 shows that the block-based interface produced larger effects on learning environment usability, practical performance, attitude toward no-code development, final product quality, and cognitive achievement. In contrast, the logic-based interface produced stronger effects on logical thinking and learning environment interaction. This pattern suggests that the block-based interface was more supportive of usability, performance, and product completion, whereas the logic-based interface was more effective in promoting analytical engagement with conditional rules, formulas, and application logic.

The very large effect sizes observed in some outcomes should be interpreted in light of the wide differences between post-measurement means and the relatively low within-group dispersion. They also reflect the clearly differentiated nature of the two experimental treatments in representing application logic. Nevertheless, these values should be interpreted within the limits of the sample, instruments, and quasi-experimental design.

18. DISCUSSION AND INTERPRETATION OF RESULTS

The results of the study showed that the effect of the no-code development interface pattern did not follow a single direction across all variables. The block-based interface outperformed the logic-based interface in cognitive achievement, practical performance, final product quality, attitude toward the development environment, and learning environment usability. In contrast, the logic-based interface outperformed the block-based interface in logical thinking and interaction within the learning environment. This pattern indicates that the interface pattern does not function merely as an external form of the tool; rather, it determines how knowledge, procedures, and logic are represented during educational application development.

18.1 Discussion of the First Hypothesis Related to Cognitive Achievement

The results showed a statistically significant difference between the two groups in the post-application of the Cognitive Achievement Test, after controlling for the effect of the pre-measurement, in favor of the group that studied through the block-based interface. This result can be interpreted in light of the nature of the block-based interface, which presents the basic concepts of application development in a clear visual form. Students see the event, condition, variable, and response as assemblable blocks, which helps them form a direct cognitive representation of the structure and functions of the application.

This result is consistent with studies on block-based programming, which indicate that visual representation of commands reduces the burden associated with code syntax and supports novice learners' understanding of programming concepts. Hu et al. (2021) showed that block-based visual programming can support students' learning of programming concepts, while Sun et al. (2023) indicated that the programming pattern used affects learners' behavior and understanding of programming tasks. Recent reviews also support this direction, showing that visual and block-based programming tools are widely used to develop computational thinking and concepts associated with building digital solutions (Irawan et al., 2023).

The superiority of the block-based interface in cognitive achievement can also be explained by the fact that the educational technology students in the present study had no prior experience with no-code development tools. They therefore needed an interface pattern that reduced the difficulty of getting started and transformed abstract concepts into visible components. The block-based interface helped students connect the concept directly to application practice: the concept of an "event" became associated with a block triggered by pressing a button, the concept of a "condition" became associated with an if/then block, and the concept of a "variable" became associated with storing a score or the number of attempts.

This result does not mean that the logic-based interface has lower cognitive value. Rather, it may require a higher level of abstraction at the beginning because students deal with formulas, properties, and logical relationships in a less direct way than with visual blocks. Therefore, the logic-based interface may be more appropriate after students have acquired an initial cognitive foundation about application structure, whereas the block-based interface appears more suitable in the early stages of concept acquisition.

18.2 Discussion of the Second Hypothesis Related to Practical Performance

The results showed a statistically significant difference between the two groups in the Practical Performance Observation Checklist, after controlling for the effect of the pre-measurement, in favor of the group that studied through the block-based interface. This result indicates that the block-based interface helped students perform educational application development steps more efficiently during work,

particularly in the skills of designing screens, adding components, building interaction, designing assessment and feedback, and testing the application.

This can be explained by the fact that the block-based interface makes practical performance more observable and modifiable. Students can see the sequence of blocks, change their order, replace one block with another, and directly test the effect of these changes within the application. This is consistent with studies on block-based programming, which have shown that visual environments help learners focus on solution logic rather than becoming preoccupied with textual syntax errors. They also allow learners to experiment and revise repeatedly in a less threatening way than traditional programming (Çakiroğlu & Çevik, 2022; Hu et al., 2021).

This result is also connected to the nature of the sample. Educational technology students need to transform instructional design concepts into an applied product, but they are not necessarily professional programmers. The block-based interface therefore provided them with a practical pathway for building the application step by step, while maintaining clarity in the relationship between interface elements and their behavior. This helped raise practical performance compared with the logic-based interface, which may require students to understand properties and formulas before reaching the practical outcome.

This result is further supported by Corral et al. (2021), who showed the ability of block-based programming to enable students to acquire and transfer knowledge in development contexts close to no-code development. In the block-based interface, students do not only learn an abstract concept; they use it directly to build a functioning application. This explains the higher practical performance scores achieved by this group.

18.3 Discussion of the Third Hypothesis Related to Final Product Quality

The results showed a statistically significant difference between the two groups in the Final Product Quality Evaluation Rubric in favor of the block-based interface. This result can be explained by the clarity of the block-based interface and the ease of tracing relationships among components and events, which helped students produce more complete applications in terms of design, interaction, assessment, feedback, and operability.

The final product in this study was not merely a visual interface. It was a small-scale educational application that included an objective, content, an activity, assessment, feedback, and a result or score. These elements require clear organization between the design aspect and the functional aspect. The block-based interface helped students connect visual components with operational logic with relative ease, which was reflected in the quality of the final product.

This result is consistent with studies showing that block-based environments help learners build executable digital products by reducing technical barriers and allowing them to focus on constructing and modifying the solution. Ruiz-Rube et al. (2019) showed that block-based development of mobile learning experiences can support the construction of educational applications linked to practical learning contexts. Hu et al. (2021) also indicated the effectiveness of block-based visual programming in improving students' learning outcomes.

The fact that the logic-based interface did not outperform the block-based interface in final product quality, despite its superiority in logical thinking, can be explained by the idea that logical thinking does not necessarily guarantee the production of a more complete product in terms of design or usability. A student may be able to analyze conditions and formulas, but may need more time and effort to adjust the interface, properties, and relationships within the application. Thus, the logic-based interface may be deeper analytically, but it is also more demanding for novice students when they are required to produce a fully integrated final product.

18.4 Discussion of the Fourth Hypothesis Related to Logical Thinking

The results showed a statistically significant difference between the two groups in the Logical Thinking Test, after controlling for the effect of the pre-measurement, in favor of the group that studied through the logic-based interface. This is one of the central findings of the study because it reveals that the superiority of the block-based interface in performance and product quality does not mean superiority in all aspects of learning. The logic-based interface may have a stronger effect on developing thinking based on analyzing relationships, conditions, and outcomes.

This result can be explained by the fact that the logic-based interface requires students to deal directly with formulas, properties, conditions, and response flows. Students do not merely assemble ready-made blocks; they need to think about the appropriate formula, determine the condition, adjust the property, and analyze the effect of each decision on the behavior of the application. This type of work

supports skills such as conditional reasoning, cause-and-effect analysis, detection of logical errors, and organization of data and variables.

This result is consistent with Matook et al. (2023), who indicated that application development in low-code environments can support metacognitive skills, including planning, monitoring, analysis, and decision-making. It is also consistent with Wing's (2006) view of computational thinking as being linked to problem analysis and systematic solution construction, and with Çakiroğlu and Çevik (2022), who showed that development environments can support sub-skills such as abstraction and analysis when tasks are designed to require them.

The superiority of the logic-based interface in logical thinking can also be explained by the fact that students had to review the underlying reason behind each response within the application. When building a question inside an educational application, the student had to determine when the answer was correct, how the score would be calculated, when feedback would appear, and what the next action should be. These operations represent repeated logical thinking situations, not merely technical steps.

This result points to the importance of not relying only on the easier interface when designing educational application development courses. The block-based interface may be suitable at the beginning of learning, but it may not be sufficient on its own to develop logical thinking in depth. A gradual integration of the block-based and logic-based interfaces may therefore be a more effective approach in preparing educational technology students.

18.5 Discussion of the Fifth Hypothesis Related to Attitude Toward the No-Code Development Environment

The results showed a statistically significant difference between the two groups in the Attitude Toward the No-Code Development Environment Scale in favor of the block-based interface. This result can be explained by the fact that students who used the block-based interface experienced a stronger sense of control and accomplishment while building the application. This was reflected positively in their perception of the value of the environment, its ease of use, their motivation, and their confidence in their ability to develop educational applications in the future.

This result is consistent with the Technology Acceptance Model, which explains that perceived ease of use and perceived usefulness influence users' attitudes toward technology (Davis, 1989). It is also consistent with the UTAUT model, which emphasizes the importance of performance expectancy and effort expectancy in technology acceptance (Venkatesh et al., 2003). The block-based interface reduced students' perceived effort while at the same time enabling them to produce a clear educational application, which strengthened their positive attitude toward the development environment.

This result can also be interpreted in light of students' limited prior experience in no-code development. Novice learners may build their attitude toward a tool through their first experience with it. If they feel capable of achievement, their motivation and confidence increase; if they experience ambiguity and complexity, their attitudes may decline despite the value of the tool. The effect of the block-based interface on attitude was therefore clear because it provided a more reassuring and clearer learning experience.

However, the relatively lower mean attitude score in the logic-based interface group should not be interpreted as rejection of the tool. Rather, it reflects the more challenging nature of the interface. Students recognized the value of the logic-based interface through interaction and thinking, but it required greater effort in formulas and conditions, which affected perceived ease of use and overall attitude.

18.6 Discussion of the Sixth Hypothesis Related to Learning Environment Usability

The results showed a statistically significant difference between the two groups in the Learning Environment Learning Environment Usability Questionnaire in favor of the block-based interface. This result can be explained by the fact that students were not evaluating Moodle alone; rather, they were evaluating their overall experience within the no-code development-based learning environment, including task clarity, ease of completing activities, adequacy of support, and smooth transition from explanation to application.

This result is consistent with the definition of usability in ISO 9241-11, which links usability to users' ability to achieve their goals effectively, efficiently, and satisfactorily within a specific context of use (ISO, 2018). Students in the block-based interface group were able to achieve their practical goals more successfully and with relatively less effort, which raised their perception of the usability of the environment.

This result also supports Brooke's (1996) view of the importance of self-reported usability in

understanding user experience, as well as Silva et al.'s (2021) observation that users in low-code development platforms may encounter cognitive difficulties when the relationships among components and properties are unclear. Accordingly, the logic-based interface may have required students to interpret formulas and properties more frequently, which lowered their relative evaluation of usability despite their higher logical thinking and interaction levels.

This finding suggests that usability in educational development environments is not related only to the quality of the learning platform organization. It is also connected to the nature of the development tool itself. The environment may be well organized, but the tool used within it may make the experience easier or more complex. This justifies the inclusion of usability among the variables of the present study.

18.7 Discussion of the Seventh Hypothesis Related to Interaction Within the Learning Environment

The results showed a statistically significant difference between the two groups in the Learning Interaction Analysis Log in favor of the logic-based interface. At first glance, this result may seem different from the results related to attitude and usability. However, it is logical when viewed in relation to the nature of the logic-based interface, which required students to review instructional resources, return to examples, repeat attempts, seek support, and follow instructions more frequently.

This result is consistent with the learning analytics literature, which emphasizes that interaction data should not be interpreted as direct evidence of ease or difficulty alone, but as an indicator that requires interpretation in light of the task and instructional context (Gašević et al., 2015; Siemens & Long, 2011). Higher interaction in the logic-based interface group may indicate deeper cognitive engagement, a greater need for review and support, or the nature of tasks requiring multiple attempts.

This result may also be explained by the fact that the logic-based interface pushed students to return repeatedly to the content and support resources in order to understand formulas, conditions, and workflows. This increased interaction does not necessarily mean that the interface was easier; rather, it may mean that it stimulated processes of searching, checking, and reviewing, which are closely related to logical thinking and problem solving.

This finding highlights the importance of using interaction logs with caution. The block-based interface may be higher in usability and attitude, but it does not require students to interact intensively with resources in the same way as the logic-based interface. Therefore, the higher interaction in the logic-based group enriches the interpretation of its superiority in logical thinking because it suggests that students engaged in more extended processes of learning and review while building the application.

18.8 General Discussion of the Results

Taken together, the findings reveal that the no-code development interface pattern affects learning outcomes in different ways. The block-based interface was more supportive of aspects requiring visual clarity and ease of implementation, such as cognitive achievement, practical performance, final product quality, attitude, and usability. This can be explained by the fact that it is well suited to novice students because it reduces syntax-related difficulties, makes application logic visible, and helps them build a functioning educational product within an appropriate time frame.

By contrast, the logic-based interface had a stronger effect on logical thinking and interaction within the learning environment. This is because it places students in analytical tasks that require understanding formulas, identifying conditions, adjusting properties, and testing relationships between inputs and outputs. This type of work may not be the easiest, but it supports deeper thinking about application logic and pushes students to review resources and interact with the environment more intensively.

These findings are consistent with recent trends that do not view no-code development environments merely as easy tools, but as learning environments capable of developing different levels of knowledge, performance, and thinking, depending on their design, interface pattern, and context of use (Matook et al., 2023; Sońta & Przegalińska, 2023; Tsakalerou & Xenos, 2023). The findings also support the importance of distinguishing between "ease of completion" and "depth of thinking." The easier interface may be more suitable for initial production, while the more challenging interface may be more suitable for logical thinking and analysis.

From the perspective of preparing educational technology students, the results indicate that training students in no-code development should not be limited to a single interface. It may be more appropriate to begin with the block-based interface to build confidence, practical understanding, and product development skills, and then move gradually to the logic-based interface to deepen conditional and analytical thinking. In this way, a

balance can be achieved between ease of learning and product quality on the one hand, and the development of logical thinking on the other.

19. CONCLUSIONS

In light of the results of the study and their discussion, the following conclusions can be drawn:

1. The no-code development interface pattern affects the learning outcomes of educational technology students. Therefore, no-code development interfaces should not be viewed as educationally equivalent.
2. The block-based interface supports the development of cognitive achievement related to educational application development concepts because it presents concepts and procedures in a visual and traceable form.
3. The block-based interface contributes more strongly to developing practical performance and final product quality among novice students because it reduces the difficulty of getting started and helps them build the application step by step.
4. The logic-based interface is more supportive of logical thinking because it requires students to work directly with formulas, conditions, properties, and response flows.
5. The block-based interface is associated with more positive attitudes and higher perceived usability among students, particularly in the early stages of learning no-code development.
6. The logic-based interface leads to a higher level of interaction within the learning environment, most likely because it requires more frequent review of resources, examples, and support materials.
7. A gradual integration of the block-based interface and the logic-based interface represents a promising direction for preparing educational technology students to develop educational applications that combine product quality with depth of thinking.

20. RECOMMENDATIONS

In light of the findings, the researchers recommend the following:

1. No-code development environments should be incorporated into Educational Software Design and Production courses in educational technology preparation programs.
2. Novice students should begin their training with block-based interfaces when educational application development concepts are introduced for the first time, because these interfaces provide visual clarity and facilitate product construction.
3. Logic-based interfaces should be employed in later stages of training, especially when the instructional goal is to develop logical thinking, conditional reasoning, and analysis of workflows within the application.
4. Educational application development activities should be designed to combine instructional analysis with technical implementation, rather than being limited to the construction of superficial interfaces.
5. Students should be trained to build educational applications that include content, an interactive activity, assessment, feedback, and a score or final result.
6. Practical Performance Observation Checklists and Final Product Quality Evaluation Rubrics should be used when assessing educational application development skills, rather than relying solely on cognitive tests.
7. Interaction logs within learning platforms such as Moodle should be used as a supporting source for understanding students' behavior during learning, provided that these logs are interpreted in light of the nature of the task and not in isolation.
8. The usability of learning environments should be given careful attention when presenting application development tasks, because students' perceptions of clarity and ease of use influence their attitudes and persistence.
9. Training guides should be prepared for faculty members on how to design no-code development-based learning pathways that take into account the differences between block-based and logic-based interfaces.
10. Students should be encouraged to test the educational applications they develop, analyze errors, and improve the product before final submission.

21. SUGGESTIONS FOR FUTURE RESEARCH

The present study suggests that future research may address the following topics:

1. The effect of sequential integration between the block-based interface and the logic-based interface on developing educational application development skills and computational thinking among educational technology students.
2. The effect of no-code development interface pattern on developing design thinking among educational technology students.
3. The interaction between no-code development interface pattern and prior technical experience level in developing the quality of digital

educational products.
4. The effect of using no-code development environments in producing AI-based educational applications among educational technology students.
5. A comparison of the effects of block-based, logic-based, and text-based interfaces on developing educational application development skills among university students.
6. A qualitative study of educational technology students' experiences while using logic-based interfaces to build educational applications.
7. The effect of employing collaborative work activities within no-code development environments on application quality and logical thinking.
8. A proposed training model for teaching no-code educational application development based on progression from the block-based interface to the logic-based interface.
9. The effect of the usability of no-code development tools on motivation and attitudes toward producing educational applications.
10. An analysis of interaction patterns within Moodle and their relationship to final product quality in educational software production courses.

22. LIMITATIONS

The findings of the study should be interpreted in light of the following limitations:

The study was limited to third-year students in the Department of Educational Technology, Faculty of Specific Education, Alexandria University. Therefore, caution should be exercised when generalizing the findings to other specializations or educational levels.

The study compared a block-based interface represented by MIT App Inventor and a logic-based interface represented by Microsoft Power Apps. The results may differ when using other tools that belong to the same interface pattern.

The treatment lasted for eight weeks. A longer application period or more extended training may lead to different results, particularly for the logic-based interface, which may require more time to master.

Logical thinking was measured through a situational test connected to the context of educational application development. Therefore, the results reflect logical thinking within this context and not all general forms of logical thinking.

The Interaction Analysis Log relied on indicators extracted from the Moodle environment. These indicators support the interpretation of behavior within the environment, but they are not sufficient by themselves to judge the quality of learning.

The final product required from students was limited to a small-scale interactive educational application. The results may differ if students are asked to develop larger or more complex applications.

23. DECLARATIONS

23.1 Data Availability

The data used in this study can be made available in aggregate form upon reasonable request, without disclosing students' identities or any individual data that may compromise confidentiality.

23.2 Funding

The research did not receive any external funding.

23.3 Conflicts of Interest

The researchers declare that there are no conflicts of interest related to this study.

23.4 Authors' Contributions

The researchers jointly contributed to designing the experimental treatment, preparing the research instruments, supervising implementation, analyzing the data, interpreting the results, and writing the final version of the research.

REFERENCES

1. American Educational Research Association, American Psychological Association, & National Council on Measurement in Education. (2014). *Standards for educational and psychological testing*. American Educational Research Association. https://www.testingstandards.net/uploads/7/6/6/4/76643089/standards_2014edition.pdf
2. Aytekin, A., & Topcu, M. S. (2023). Improving 6th grade students' creative problem solving skills through plugged and unplugged computational thinking approaches. *Journal of Science Education and Technology*. https://doi.org/10.1007/s10956-024-10130-y
3. Brooke, J. (1996). SUS: A quick and dirty usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, & I. L. McClelland (Eds.), *Usability evaluation in industry*. Taylor & Francis. https://digital.ahrq.gov/sites/default/files/docs/survey/systemusabilityscale%2528sus%2529_comp%255B1%255D.pdf
4. Çakiroğlu, Ü., & Çevik, İ. (2022). A framework for measuring abstraction as a sub-skill of computational

thinking in block-based programming environments. *Education and Information Technologies*. https://doi.org/10.1007/s10639-022-11019-2
5. Corral, L., Fronza, I., & Pahl, C. (2021). Block-based programming enabling students to gain and transfer knowledge with a no-code approach. In *Proceedings of the 22nd Annual Conference on Information Technology Education*. Association for Computing Machinery. https://doi.org/10.1145/3450329.3478314
6. Creswell, J. W., & Creswell, J. D. (2023). *Research design: Qualitative, quantitative, and mixed methods approaches* (6th ed.). SAGE.
7. Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly, 13*(3), 319–340. https://doi.org/10.2307/249008
8. Fraenkel, J. R., Wallen, N. E., & Hyun, H. H. (2023). *How to design and evaluate research in education* (11th ed.). McGraw Hill.
9. Gašević, D., Dawson, S., & Siemens, G. (2015). Let's not forget: Learning analytics are about learning. *TechTrends, 59*, 64–71. https://doi.org/10.1007/s11528-014-0822-x
10. Hattie, J., & Timperley, H. (2007). The power of feedback. *Review of Educational Research, 77*(1), 81–112. https://doi.org/10.3102/003465430298487
11. Hu, Y., Chen, C. H., & Su, C. Y. (2021). Exploring the effectiveness and moderators of block-based visual programming on student learning: A meta-analysis. *Journal of Educational Computing Research, 58*(8), 1467–1493. https://doi.org/10.1177/0735633120945935
12. International Organization for Standardization. (2018). *ISO 9241-11:2018: Ergonomics of human-system interaction – Part 11: Usability: Definitions and concepts*. https://www.iso.org/obp/ui/
13. Irawan, E., Rosjanuardi, R., & Prabawanto, S. (2023). Promoting computational thinking through programming trends, tools, and educational approaches: A systematic review. *JTAM: Jurnal Teori dan Aplikasi Matematika*. https://pdfs.semanticscholar.org/2e88/551f42072ca3ba8d2b771a96274183de12f6.pdf
14. Lim, J. T., Lee, C. K., Lim, E. H., Wong, P. V., Yew, K. T., & Ooi, C. Y. (2023). Preliminary study on the accessibility and learning experience of low-code development platforms. In *2023 5th International Conference on Artificial Intelligence and Emerging Technology*. IEEE. https://ieeexplore.ieee.org/document/10730306/
15. Lu, C., Macdonald, R., Odell, B., Kokhan, V., Demmans Epp, C., & Cutumisu, M. (2022). A scoping review of computational thinking assessments in higher education. *Journal of Computing in Higher Education*. https://doi.org/10.1007/s12528-021-09305-y
16. Matook, S., Wang, Y. M., Koeppel, N., & Guerin, S. (2023). Metacognitive skills in low-code app development: Work-integrated learning in information systems development. *Journal of Information Technology*. https://doi.org/10.1177/02683962231170238
17. Nielsen, J. (2023). *10 usability heuristics for user interface design*. Nielsen Norman Group. https://www.nngroup.com/articles/ten-usability-heuristics/
18. Pršala, J. (2023). The effect of block-based programming activities on the computational thinking skills of pre-service primary school teachers. *Journal of Pedagogical Sociology and Psychology*. https://www.j-psp.com/article/the-effect-of-block-based-programming-activities-on-the-computational-thinking-skills-of-pre-service-14644
19. Ruiz-Rube, I., Mota, J. M., Person, T., Corral, J. M. R., & Dodero, J. M. (2019). Block-based development of mobile learning experiences for the internet of things. *Sensors, 19*(24), Article 5467. https://doi.org/10.3390/s19245467
20. Shute, V. J. (2008). Focus on formative feedback. *Review of Educational Research, 78*(1), 153–189. https://doi.org/10.3102/0034654307313795
21. Siemens, G., & Long, P. (2011). Penetrating the fog: Analytics in learning and education. *EDUCAUSE Review, 46*(5), 30–40. https://er.educause.edu/articles/2011/9/penetrating-the-fog-analytics-in-learning-and-education
22. Silva, C., Vieira, J., Campos, J. C., Couto, R., & Ribeiro, A. N. (2021). Development and validation of a descriptive cognitive model for predicting usability issues in a low-code development platform. *Human Factors, 63*(6), 1012–1032. https://doi.org/10.1177/0018720820920429
23. Sońta, M., & Przegalińska, A. (2023). Say "yes" to "no-code" solutions: How to teach low-code and no-code competencies to non-IT students. In *Handbook of social computing*. Edward Elgar Publishing. https://www.elgaronline.com/edcollchap/book/9781803921259/book-part-9781803921259-30.xml
24. Sun, D., Looi, C. K., Li, Y., Zhu, C., Zhu, C., & Cheng, M. (2023). Block-based versus text-based

programming: A comparison of learners' programming behaviors, computational thinking skills and attitudes toward programming. *Educational Technology Research and Development*. https://www.hznu.edu.cn/upload/resources/file/2023/01/18/7815323.pdf

25. Torres, A., & Kapralos, B. (2023). Examining the usability of the Moirai serious game authoring platform. *IEEE Transactions on Games*. https://ieeexplore.ieee.org/document/10417811
26. Tsakalerou, M., & Xenos, M. N. (2023). Unlocking the secrets of student success in low-code platforms: An in-depth comparative analysis. *2023 ASEE Annual Conference & Exposition*. https://peer.asee.org/48206.pdf
27. Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D. (2003). User acceptance of information technology: Toward a unified view. *MIS Quarterly, 27*(3), 425–478. https://doi.org/10.2307/30036540
28. Weng, X., Ye, H., Dai, Y., & Ng, O. (2023). Integrating artificial intelligence and computational thinking in educational contexts: A systematic review of instructional design and student learning outcomes. *Journal of Educational Computing Research*. https://doi.org/10.1177/07356331241248686
29. Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35. https://doi.org/10.1145/1118178.1118215
30. Yeni, S., Grgurina, N., Saeli, M., & Hermans, F. (2023). Interdisciplinary integration of computational thinking in K-12 education: A systematic review. *Informatics in Education*. https://files.eric.ed.gov/fulltext/EJ1428807.pdf