

Developing Computational Thinking in Primary School Students Through a Smart Farming Context Using the Engineering Design Process

Supachai Kongpui¹ and Tussatrin Wannagatesiri^{1,*}

¹Department of Teacher Education, Kasetsart University,
Kamphaeng Saen Campus, Nakhon Pathom, Thailand

*Email: tussatrin.k@ku.th

Received: 17 Jun. 2026 Revised: 13 Aug. 2026 Accepted: 19 Aug. 2026

Abstract. The purpose of this research was to: (1) develop learning activities using the Engineering Design Process (EDP) within a smart farming context to enhance the computational thinking (CT) skills of primary school students, and (2) compare the students' CT skills before and after participating in these activities. The participants were 39 fifth-grade students from a large inclusive primary school in Thailand, selected through purposive sampling. The research employed a pre-experimental design. Research instruments included four EDP-based smart farming lesson plans totaling 12 hours of instruction, a CT concept test, and a performance-based CT rubric. The results indicated that the integration of the six-step EDP, ranging from identifying authentic agricultural problems to iteratively debugging automated sensor prototypes, significantly improved students' CT skills. Quantitative analysis revealed a statistically significant increase in overall CT skill scores, with the post-test mean ($M = 19.46$, $SD = 2.35$) being significantly higher than the pre-test mean ($M = 11.64$, $SD = 2.98$) at $p < .001$. Significant improvements were observed across all specific computational components, with the largest improvement occurring in conditional logic. Furthermore, qualitative data from the performance-based rubric and final smart farming projects confirmed that students effectively applied the four pillars of CT: decomposition, pattern recognition, abstraction, and algorithm design to solve real-world agricultural challenges. This study concludes that anchoring computational tasks in authentic, tangible contexts like smart farming successfully bridges the gap between abstract coding and meaningful physical outcomes.

Keywords: Computational thinking; Engineering Design Process; smart farming; primary education

INTRODUCTION

Computational thinking (CT) is a cognitive process that involves formulating problems and their solutions in a way that can be effectively executed by an information-processing agent (Wing, 2006; Barr & Stephenson, 2011). Quality education plays a pivotal role in equipping learners with CT skills, fostering the creativity necessary to comprehend and transform the world (Wing, 2006). By developing systematic approaches, CT enables learners to achieve efficient outcomes and apply these methods to diverse, everyday challenges (Grover & Pea, 2013). Consequently, CT is widely recognized not merely as a skill for computer scientists, but as a foundational competency for all individuals (Wing, 2006). The core principles of CT, such as decomposition, pattern recognition, algorithmic design, and abstraction, can be utilized to comprehend human behaviors and complex systems (Weintrop et al., 2016). In this study, a clear conceptual distinction is maintained between the two terms: "computational thinking" (CT) refers to the overarching cognitive process and

theoretical framework, whereas “computational thinking skills” refers to the measurable practical competencies—namely, decomposition, pattern recognition, abstraction, and algorithm design—that students develop and apply to solve problems. Furthermore, these principles are highly valuable for navigating unstructured problems in daily life (Yadav et al., 2016). In recent years, the promotion of CT in pre-university education has progressed significantly. Globally, primary and secondary education systems have increasingly emphasized the development of coding, logical reasoning, and problem-solving skills through educational robotics and digital applications (García-Peñalvo & Mendes, 2018; Hsu et al., 2018). Early childhood and primary education are considered optimal periods for introducing CT concepts (Montiel & Gomez-Zermeño, 2021). Consequently, various instructional approaches have been explored, ranging from the use of tangible smart toys (Lin et al., 2020) and simple robotics (Isnaini et al., 2019) to structured curricula introducing basic computer science concepts (Su & Yang, 2023) and novel assessment tools (El-Hamamsy et al., 2022). Despite this growing interest, the literature on the effective implementation of CT specifically in primary schools remains limited, necessitating further research into pedagogical methods that assist teachers in introducing CT concepts effectively (Chalmers, 2018). Recent systematic reviews further emphasize the need for engaging, context-driven CT activities for young learners (Paraskevopoulou-Kollia et al., 2025).

One compelling pedagogical approach to address this gap is the Engineering Design Process (EDP). EDP is a systematic problem-solving framework that operates under resource constraints to yield optimal outcomes (Mentzer et al., 2015). To establish a robust theoretical foundation for this approach, Moore et al. (2014) emphasize in their framework for quality K–12 engineering education that engineering design is a central mechanism for meaningful STEM integration. This framework highlights the critical role of engaging students in authentic problem solving, navigating design constraints, optimizing solutions, and utilizing iterative processes to address real-world applications. Integrating EDP into learning activities encourages students to engage in hands-on, step-by-step problem solving based on real-world situations, thereby fostering creativity and innovation (Kareemee, 2017). This process strongly aligns with constructivist learning theories and the Framework for K–12 Science Education, which emphasizes the integration of science, technology, and engineering practices (National Research Council, 2012; Phutwatthana, 2018). Moreover, integrating CT with EDP creates a robust conceptual framework that significantly enhances students’ problem-solving capabilities (Sonay & Kiray, 2022). Recent studies confirm that STEM-based EDP models effectively improve learning outcomes and critical thinking in elementary settings (Sutisnawati et al., 2025).

Despite these pedagogical advancements, several gaps remain in the current literature. First, there is a knowledge gap regarding how primary students transfer abstract, screen-based coding skills to tangible physical solutions. Second, a methodological gap exists in CT assessment; while prior studies often rely on standardized multiple-choice tests, there is a distinct lack of authentic, performance-based assessments that measure how students apply CT to solve multidimensional problems.

To address these gaps, this study introduces a contextual innovation by integrating the EDP-CT framework specifically within a smart farming environment. Smart farming is utilized not merely as a generic real-world backdrop, but because its inherent technological characteristics perfectly mirror core computational operations. Specifically, smart farming involves sensor-based data collection, conditional logic, automation, and environmental monitoring. These specific features directly align with the four pillars of CT: identifying complex crop issues requires *decomposition*, analyzing environmental trends utilizes *pattern recognition*, filtering non-essential environmental variables demands *abstraction*, and designing automated irrigation or lighting feedback systems necessitates precise *algorithm design*. Therefore, smart farming provides an unparalleled and highly specific setting that makes the abstract concepts of coding visible and impactful, successfully bridging the gap between theoretical computational models and meaningful physical outcomes.

RESEARCH OBJECTIVES

The specific objectives of this research are as follows:

1. To develop learning activities incorporating the Engineering Design Process (EDP) within a smart farming context to enhance the computational thinking skills of primary school students.
2. To compare the computational thinking skills of primary school students before and after their participation in the developed learning activities.

METHODOLOGY

This study employed a pre-experimental research design, specifically a one-group pre-test–post-test design, to measure changes in students’ computational thinking (CT) skills. Additionally, to capture the dynamic learning atmosphere and provide a deeper understanding of the students’ problem-solving processes and learning behaviors, qualitative data were gathered concurrently. Throughout the intervention, this information was collected via unstructured individual and group interviews, as well as classroom observations.

Participants

The target population comprised all fifth-grade students at a large inclusive primary school, totaling 207 students distributed across 5 classrooms. From this sampling frame, one classroom was selected using purposive sampling. Therefore, the participants consisted of 39 fifth-grade students (aged 10–11 years) enrolled in a science and technology course during the second semester of the 2022 academic year.

Research Instruments

The research instruments utilized in this study comprised instructional and data-collection tools:

EDP-Based Smart Farming Lesson Plans

A set of four Engineering Design Process (EDP)-based smart farming lesson plans, totaling 12 hours of instruction (one hour per week for 12 weeks), was developed for this study. The instructional design was anchored in the six-step EDP framework outlined by the National Research Council (2012): (1) identifying the problem, (2) gathering related information and concepts, (3) designing a solution, (4) planning and executing the solution, (5) testing, evaluating, and improving, and (6) presenting the outcomes. Prior to implementation, these lesson plans were validated by three subject-matter experts to evaluate content validity, alignment with learning objectives, and the appropriateness of the instructional activities. The evaluation results demonstrated an item-objective congruence (IOC) values ranging from .67 to 1.00 for all components, indicating high content validity.

To contextualize the learning experience, the instructor integrated an agricultural context through simulated scenarios across all four lesson plans. This approach provided students with specific, real-world conditions to guide their algorithm design and coding. The progression of the lessons began with “Night-Blooming Purslane,” where students explored the effects of light and temperature. Utilizing teacher-provided data on the light requirements of various plant species, students wrote code to control an artificial lighting system designed to trick a morning-blooming plant into flowering at night. The second lesson, “Little Garden to Care For,” focused on soil moisture. Students analyzed datasets detailing optimal moisture levels for different vegetables, allowing them to define variables and set conditional statements (if/else) to process sensor inputs for an automated watering system. In the third lesson, “Desired pH Levels,” students examined water quality data and developed code to trigger alerts when pH levels deviated from standard criteria necessary for healthy plant growth. Finally, the “Smart Farming” lesson culminated in large-scale farm management by integrating all previously explored parameters: light, moisture, and water quality. In this final stage, students designed a comprehensive, fully integrated sensor system aimed at reducing operational costs and increasing agricultural yields.

Crucially, this pedagogical design ensured that coding was not taught in an abstract or isolated manner. Instead, the teacher acted as a facilitator by supplying an “agricultural database” comprising specific environmental parameters required by the students’ chosen plants to serve as practical input data. Consequently, students were challenged to apply computational thinking skills to decompose these agricultural problems and translate numerical values into logical conditions within their programs. This intentional integration made the process of learning to code highly meaningful, directly demonstrating its applicability in solving authentic, real-world challenges.

Computational Thinking Skills Assessment

To comprehensively evaluate CT skills, two assessment formats were employed:

2.1 CT Concept Test. Adapted from the Computational Thinking Test (CT-test) developed by Román-González et al. (2017), this 28-item multiple-choice instrument comprehensively measures foundational computational concepts. The test addresses seven concepts and conditions: basic directions, loops (repeat times), loops (repeat until), simple conditionals (if), complex conditionals (if/else), while conditionals, and simple functions. Three subject-matter experts validated the adapted instrument, which achieved item-objective congruence (IOC) values ranging from .67 to 1.00 and a Cronbach’s alpha reliability coefficient of .76.

2.2 Performance-Based CT Rubric. The researcher developed a performance-based computational thinking (CT) assessment based on the “Four Pillars of Computational Thinking” framework and informed by Wing (2006) and Barr and Stephenson (2011). The assessment examines students’ proficiency in designing a logical, step-by-step method or flowchart to solve the prioritized problem effectively. To evaluate students’ practical problem-solving capabilities systematically within the prescribed agricultural context, a four-level scoring rubric was established in accordance with performance-assessment guidelines. The rubric yielded high IOC values (.67–1.00) and an excellent reliability coefficient of .96, as demonstrated in the example below.

Scenario: *In a small condo in the heart of the city, a young woman named “May” is deeply passionate about nature. She dreams of having a beautiful garden like in the movies. However, due to her hectic work schedule, she spends most of her time tied to a computer screen, making it difficult to care for plants. One day, May decided to buy small plants to decorate her room. She chose easy-to-care-for varieties recommended by the shop. Unfortunately, it wasn’t long before the leaves began turning yellow and falling off one by one. May felt very disappointed. She tried searching the internet for care instructions but still could not solve the problem. **Q1:** Based on the given scenario, break down the problem into clear sub-problems in a bulleted list. (Decomposition: 3 points) **Q2:** Group the identified sub-problems to show which ones are identical, similar, or fall into the same pattern. (Pattern Recognition: 3 points) **Q3:** Identify which sub-problems are essential and which are non-essential. Determine the most critical problem that should be considered first, and provide supporting reasons. (Abstraction: 3 points) **Q4:** Design a step-by-step flowchart to solve the problem based on the scenario. (Algorithm Design: 3 points)*

To clarify the evaluation process, each of the four questions (representing the four CT pillars) was scored on a four-level scale ranging from 0 to 3 points. The performance levels were defined as follows: 0 points (Needs Improvement), 1 point (Fair), 2 points (Moderate), and 3 points (Good). A summary of the scoring criteria is presented in Table 1. Consequently, the mean scores presented in Figure 2 represent the average points (out of a maximum of 3.00) achieved by the 39 students for each specific CT pillar across the learning activities. The qualitative performance levels reported in the results section directly correspond to the established score ranges: 0.00–0.49 (Needs Improvement), 0.50–1.49 (Fair), 1.50–2.49 (Moderate), and 2.50–3.00 (Good).

Data Collection

The intervention commenced with a pre-test using the CT skills assessments. Following the pre-test, the students participated in the 12-hour EDP-based smart farming learning activities over a 12-week period. Throughout the intervention, qualitative data were collected via unstructured individual

and group interviews, as well as observations. Upon completion of all lesson plans, a post-test was administered using the identical CT skills assessments.

Table 1: Summary of the scoring criteria for the performance-based CT rubric

CT Pillar	Level 0: Needs Improvement (0 pts)	Level 1: Fair (1 pt)	Level 2: Moderate (2 pts)	Level 3: Good (3 pts)
Decomposition	Cannot identify any relevant sub-problems from the scenario.	Identifies only one or two minor sub-problems; lacks clarity.	Identifies most sub-problems correctly but misses some key details.	Breaks down the problem into a clear, comprehensive, and accurate bulleted list.
Pattern Recognition	Cannot group sub-problems or identify any patterns.	Attempts to group problems but connections are illogical.	Groups most problems correctly based on similarities or patterns.	Accurately categorizes all identified sub-problems into highly logical patterns.
Abstraction	Cannot distinguish between essential and non-essential problems.	Identifies a critical problem but fails to provide a supporting reason.	Identifies the most critical problem with basic, somewhat vague reasoning.	Accurately prioritizes the most critical problem and provides strong, logical reasons.
Algorithm Design	Fails to provide a step-by-step solution or flowchart.	Designs a basic workflow but lacks clear sequence or standard symbols.	Designs a functional flowchart with minor sequence or symbolic errors.	Designs a highly logical, step-by-step flowchart using correct standard symbols.

Data Analysis

Quantitative data obtained from the pre-test and post-test were analyzed using descriptive statistics, including means and standard deviations. To determine whether CT skills differed significantly before and after the intervention, a paired-samples t-test was conducted. To ensure the reliability and consistency of the performance-based assessment, the students' assignments and final smart farming projects were evaluated by two independent raters. Prior to the formal evaluation, the raters participated in a calibration session using a small sample of student work to establish a shared understanding of the four-level rubric criteria. Interrater reliability was determined using the intraclass correlation coefficient (ICC). The resulting ICC of .87 indicated excellent interrater reliability. Any minor discrepancies in scoring were resolved through discussion to reach a final consensus. The qualitative data derived from interviews and task observations were analyzed using content analysis to support and triangulate the quantitative findings.

RESULTS AND DISCUSSION

Development of EDP-Based Smart Farming Learning Activities

The first objective of this study was to develop learning activities using the Engineering Design Process (EDP) within a smart farming context to enhance the computational thinking skills of primary school students. The instructional design engaged students in a hands-on, problem-solving environment. The learning progression and student activities across the six EDP stages are detailed as follows:

Step 1: Identifying the Problem (Decomposition). Students were presented with an authentic smart farming scenario, such as the story of a young farmer named “Ton” whose vegetables were wilting due to improper soil moisture. Observation notes revealed that students initially struggled to look

past the superficial details of the story. During an unstructured interview, one student remarked, “At first, our group thought the problem was just that Ton was lazy and didn’t have time to water the plants. But when we broke the problem down together, we realized the main issue was actually ‘improper soil moisture.’ So, we had to find a way to measure the moisture for him.” However, after teacher scaffolding, students successfully shifted to computational decomposition. Students analyzed the narrative to identify the core problems (e.g., wilting and stunted growth) and their root causes (e.g., incorrect watering and inadequate soil moisture). By breaking down the scenario, students practiced problem decomposition and proposed initial technological solutions, such as building a simple soil moisture meter. *Although most students successfully identified the relevant issues, a few struggled to align their identified causes with the specific scenario, highlighting the need for guided decomposition.*

Step 2: Gathering Information and Concepts (Pattern Recognition). Students studied supplementary materials and completed activity sheets to discover concepts relevant to the identified problem. Observation data indicated that students could effectively link gathered data and identify underlying trends (patterns) to design scalable solutions. Demonstrating this, one student noted, “I looked at the data in the table, and all these vegetables need similar moisture levels. If we use a soil moisture sensor, we can probably write the same pattern of code to control the watering for all the vegetable beds.” Through group discussions, each group shared data and brainstormed at least two potential solutions. The groups then evaluated and selected the optimal design, such as an automated watering system utilizing soil moisture sensors and an LCD data display.

Step 3: Designing a Solution (Abstraction and Algorithm Design). In this collaborative stage, students independently drafted preliminary models for their chosen solutions and selected the necessary materials and equipment. Interview feedback highlighted students’ abilities to filter out irrelevant information (abstraction), as one student explained, “When drawing the flowchart, we don’t need to care about the color of Ton’s plant pots or what kind of watering can he use; that’s not important. The most important part is the if/else condition: if the moisture is below 60, turn on the water pump; if not, turn it off.” They presented their initial sketches to the instructor, receiving constructive feedback to ensure their models practically addressed the scenario’s constraints. Students then dedicated time outside class to refine these models under the teacher’s facilitation.

Step 4: Planning and Executing. Students formulated a concrete workflow to execute their designs, continuously consulting with the teacher and reporting their progress. Observations of group dynamics highlighted strong collaborative execution, with students dividing tasks based on their algorithmic plans: “I’ll be in charge of wiring the sensor to the board and the water pump. You take the flowchart our group drew and put the command blocks together in the computer. That way, we can follow the plan and finish on time.” They translated their initial sketches into detailed blueprints and proceeded to construct the physical prototypes (e.g., the automated sensor system) exactly as designed.

Step 5: Testing, Evaluating, and Improving (Debugging). Students tested their constructed prototypes to determine whether they met the specified conditions and successfully solved the initial agricultural problem. If a prototype failed to operate as intended, students engaged in an iterative debugging process. They analyzed the malfunction, consulted with the instructor regarding material selection or design flaws, and refined the hardware or code until the system functioned perfectly. Teacher observations noted that during the debugging phase, students demonstrated high levels of engagement. When an irrigation pump malfunctioned, a group was observed systematically checking their code line-by-line rather than giving up, directly applying algorithmic thinking as they troubleshooted: “Hey! Why won’t the water pump stop? The water is overflowing the pot... Wait, don’t unplug the wires yet. Let me look at the code on the screen first. Oh! We forgot to add a ‘delay’ command and forgot to loop the sensor check. Let’s go through and fix the code line by line.”

Step 6: Presenting the Outcomes. In the final stage, students conducted live demonstrations to explain the functionality of their prototypes. During these presentations, students successfully articulated the connection between their code and the physical outcomes. One group synthesized their learning by explaining, “Our group’s ‘Smart Watering System’ was built to solve the problem of wilting

vegetables. We used step-by-step thinking to command the sensor to measure the moisture level before deciding to water. This system will save time and can actually do the work for humans.” They comprehensively presented their design concepts, detailing how their computational models and physical inventions effectively resolved the initial smart farming problem.

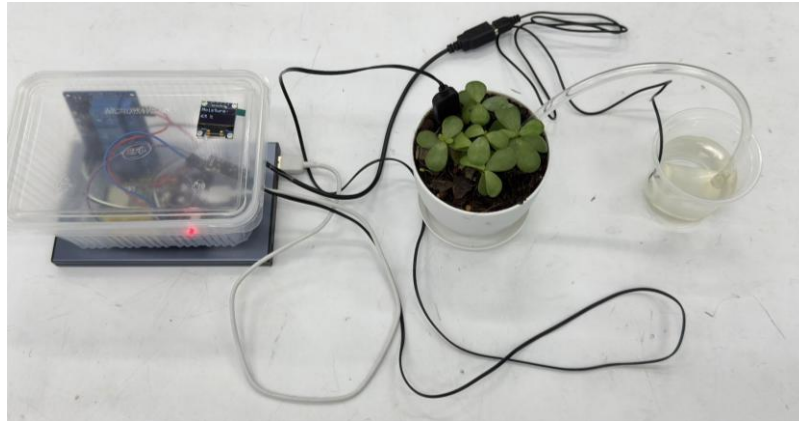


Figure 1: Implementation of the automated irrigation prototype in a simulated smart farming environment.

Comparison of Computational Thinking Skills

To address the second research objective, a paired-samples t-test was conducted to compare students’ computational thinking skills—specifically, their understanding of core computational concepts—before and after participating in the EDP-based smart farming learning activities. The statistical comparison of these pre-test and post-test scores across seven computational concepts is presented in Table 2.

Table 2: Comparison of computational concepts before and after the intervention ($n = 39$)

Computational Concepts	Pre-test <i>M (SD)</i>	Post-test <i>M (SD)</i>	<i>t</i>
Basic directions and sequences	2.90 (1.10)	3.67 (0.58)	4.74*
Loops: repeat times	2.31 (0.83)	3.38 (0.75)	8.68*
Loops: repeat until	2.00 (0.86)	3.15 (0.63)	8.87*
If: simple conditional	0.79 (0.77)	2.18 (1.07)	8.30*
If/else: complex conditional	1.31 (0.92)	2.33 (0.81)	9.58*
While conditional	0.77 (0.71)	2.05 (0.86)	11.06*
Simple functions	1.56 (1.14)	2.67 (0.96)	8.74*
Overall concepts	11.64 (2.98)	19.46 (2.35)	24.60*

Note. * $p < .001$.

Overall Performance. The findings revealed a statistically significant improvement in the students’ overall computational thinking skills. Out of a maximum score of 28, the students’ post-test mean score ($M = 19.46$, $SD = 2.35$; $range = 13–23$) was significantly higher than their pre-test mean score ($M = 11.64$, $SD = 2.98$; $range = 5–17$), with $t(38) = 24.60$, $p < .001$. This substantial increase confirms that the integration of the Engineering Design Process within an agricultural context effectively enhanced the students’ computational thinking skills.

Analysis by Computational Concepts. When analyzing the seven specific computational components, statistically significant improvements ($p < .001$) were observed across all areas:

- Basic Directions and Sequences: Students demonstrated a solid foundational understanding. The pre-test mean was 2.90 ($SD = 1.10$), which significantly increased to a near-mastery level of 3.67 ($SD = 0.58$) out of 4 points in the post-test.

- Loops (Iterative Patterns): The students' comprehension of iterative processes showed notable growth. The "Loops-repeat times" scores increased from 2.31 to 3.38, while the more complex "Loops-repeat until" scores rose from 2.00 to 3.15. This indicates that the practical activities successfully solidified their understanding of both basic and advanced repetitive patterns.
- Conditional logic (if, if/else, while): The most striking developments occurred in the application of conditional statements, which are crucial for decision-making in programming. (1) *If: simple conditional*: The baseline knowledge was initially low ($M = 0.79$), but it significantly improved post-intervention ($M = 2.18$), demonstrating the effectiveness of teaching basic conditional commands. (2) *If/else: complex conditional*: Despite the complexity, the mean score notably increased from 1.31 to 2.33, demonstrating that most learners benefited from the contextualized activities. (3) *While conditional*: This component recorded the lowest pre-test score ($M = 0.77$) but exhibited the largest improvement ($t = 11.06$), reaching a post-test mean of 2.05. The hands-on smart farming scenarios were especially effective in conveying complex control flow concepts.
- Simple functions: The ability to design modular, functional code increased from a pre-test mean of 1.56 to a post-test mean of 2.67 ($t = 8.74$). This growth highlights the students' improved capacity to manage and organize larger segments of code effectively.

In conclusion, the data demonstrate that anchoring computational concepts in tangible, real-world smart farming problems allows students to successfully grasp and apply complex programming logic, leading to significantly higher achievement in computational thinking skills.

Practical Application of Computational Thinking Skills (Performance-Based Rubric)

To triangulate and support the quantitative findings from the multiple-choice CT Concept Test (Table 2), the students' practical abilities to solve problems were evaluated using the performance-based CT assessment (rubric scoring). This instrument specifically measured the application of the four core CT pillars in authentic scenarios. Figure 2 shows the development of scores for each computational thinking skill across the four learning activities.

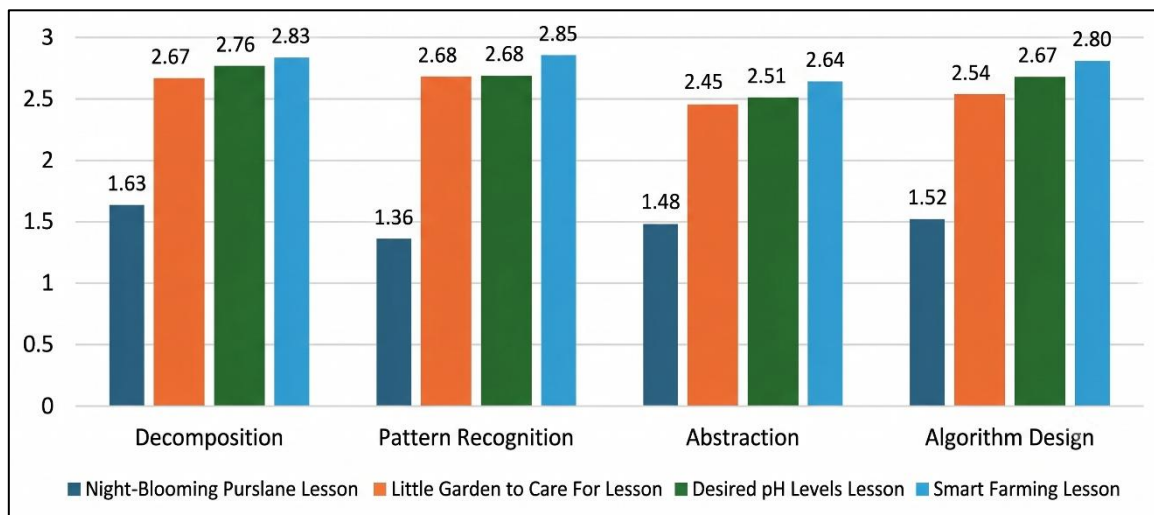


Figure 2: Development of students' computational thinking skills across four EDP-based smart farming learning activities.

Development Across CT Components

Continuous assessment during the four learning activities—Night-Blooming Purslane, Little Garden to Care For, Desired pH Levels, and Smart Farming—revealed clear progression and distinct developmental trajectories across each computational thinking component:

- **Problem Decomposition:** Students demonstrated strong, consistent development from the first activity through to the final lesson. The overall assessment indicates a high level of ability to break down complex scenarios into specific, manageable issues ($M = 2.46$, $SD = 0.56$). This is reflected in the post-test results, where 41.03% of students achieved a “Good” level and 56.41% performed at a “Moderate” level. Students successfully demonstrated this by providing precise answers to pinpoint underlying issues, such as identifying “too little or too much watering” or “yellowing plant leaves.”
- **Algorithm Design:** This component showed strong and consistent development similar to Problem Decomposition, with students continuously improving their procedural planning skills across all activities ($M = 2.38$, $SD = 0.59$). Interestingly, the post-test evaluation revealed that Algorithm Design was the component in which students demonstrated the highest proficiency. A total of 53.85% of students successfully achieved a “Good” level. This was evidenced by their ability to successfully translate their logical thinking into clear, systematic flowcharts to solve the given problems.
- **Pattern Recognition:** The developmental trajectory for this component was unique and highly distinctive. Although initial scores during the Night-Blooming Purslane lesson were noticeably low, students demonstrated the most significant adaptation and growth over time, eventually achieving the highest mastery in the final Smart Farming activity ($M = 2.38$, $SD = 0.70$). Based on the post-test responses, students effectively identified trends and grouped similar issues (28.21% achieved a “Good” level and 71.79% achieved a “Moderate” level). For instance, they effectively categorized answers like “incorrect watering” and “improper care” under the same environmental pattern.
- **Abstraction:** Students showed increasing development in filtering out irrelevant narrative details to accurately pinpoint core problems. Consequently, in the post-test evaluation, 46.15% of students achieved a “Good” level in this component. A clear example of this progression is their ability to prioritize issues, stating priorities such as “resolving the water issue,” or analyzing the context to abstract the necessity for an automated solution by identifying that the protagonist “has no time.”

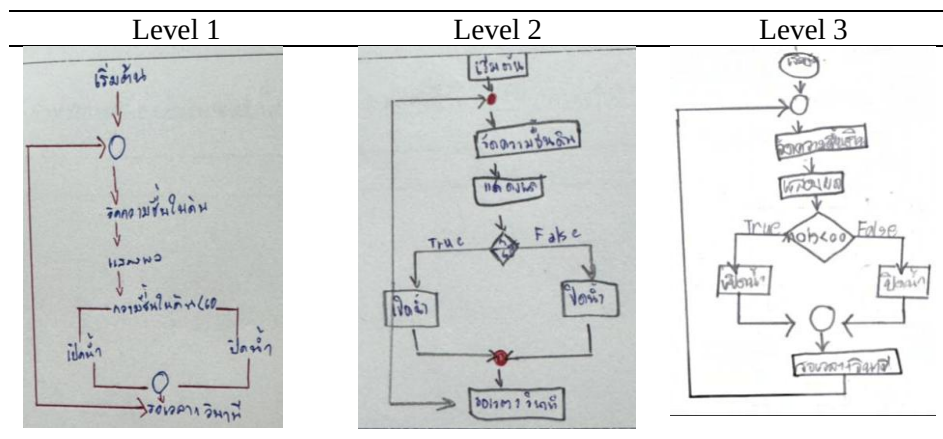


Figure 3: Examples of student flowcharts demonstrating varying levels of proficiency in algorithm design.

Based on the Algorithm Design rubric, the three student flowcharts (from left to right) illustrate distinct levels of proficiency. The left flowchart (Level 1: Fair) shows a basic understanding of the operational sequence but predominantly lacks standard flowchart symbols, omitting process rectangles and decision diamonds. It relies heavily on plain text and un-arrowed lines at decision branches, resulting in unclear communication. The middle flowchart (Level 2: Moderate) demonstrates improved accuracy by incorporating standard symbols and clearly defining “True/False” conditional paths. However, it contains minor symbolic errors, most notably using a

process rectangle instead of a standard terminator (oval) for the “Start” state. Finally, the right flowchart (Level 3: Good) fully meets the rubric’s highest criteria. The student accurately designed the algorithm with clear, step-by-step details, employing the correct standard symbols throughout—terminators for the start state, rectangles for processes (e.g., measuring moisture, displaying data), and a decision diamond with well-defined conditional paths and a neatly organized feedback loop.

Application in Smart Farming Projects

The practical application of these CT skills was evident in the students’ final smart farming projects. Across all six collaborative groups (with project titles such as “Automated Moisture,” “Smart Plant Sensor,” and “Good Soil, Perfect Water”), the application of the four CT pillars was highly consistent. As observed in the project data, students effectively decomposed environmental factors (identifying inconsistent watering and improper humidity as sub-problems), categorized these into broader water management patterns, abstracted the core requirement (the need for automated alerts to save labor), and designed functional algorithms (e.g., *Start → Read Sensor Value → Compare with Standard → Trigger Alert/Open Water Pump → End*). This robust alignment between the rubric assessment and the final prototypes confirms that the EDP-based smart farming activities successfully enhanced both theoretical understanding and practical computational thinking skills. The implementation of these EDP-based activities yielded a significant improvement in students’ computational thinking skill scores. This finding aligns with Román-González et al. (2017) and Jatuponthianchai et al. (2022), who established that problem-solving abilities significantly impact CT development.

The success of this intervention stems from the six-step Engineering Design Process, which inherently operationalizes the four pillars of CT; from decomposing problems (Step 1) and recognizing patterns in data (Step 2), to abstracting constraints for design (Step 3) and iteratively debugging algorithms (Steps 4–6). By replacing abstract coding instruction with hands-on, student-centered prototyping, learners successfully constructed new knowledge. This approach is consistent with Samad et al. (2023), who demonstrated that integrating CT with the EDP effectively transforms students into self-reliant, reflective problem solvers capable of navigating complex STEM challenges. Furthermore, Galanti and Holincheck (2024) corroborate that aligning instructional design with EDP phases effectively builds elementary students’ CT dispositions, particularly their persistence in open-ended problem solving.

Crucially, applying the EDP within an authentic smart farming context challenged students with relatable, real-life variables (e.g., soil moisture, light). Recent research by Thipphayasaeng et al. (2024) strongly supports this conclusion, showing that utilizing smart agriculture environments to teach programming significantly enhances learners’ CT skills by grounding logical tasks in tangible, physical outcomes. Ultimately, systematic engineering challenges within agricultural contexts act as highly effective catalysts for translating theoretical computational concepts into practical 21st-century skills.

Beyond empirical consistency with prior studies, the success of this intervention can be deeply understood through multiple theoretical lenses, which explain the underlying mechanisms of how EDP promotes computational thinking. First, through the lens of constructivism, the EDP framework required students to actively construct knowledge through iterative problem solving rather than passively receiving information. When prototypes failed during the testing phase, students engaged in continuous cognitive restructuring to find solutions. Second, constructionism played a vital role; by designing and creating tangible artifacts—such as the automated irrigation systems—students materialized abstract coding syntax into physical, functional models, making the learning process highly meaningful.

Furthermore, the use of the smart farming context aligns directly with situated learning theory. Students did not learn computational thinking in a vacuum; rather, they acquired these skills within an authentic, relatable agricultural context, demonstrating how computational concepts are inextricably linked to solving real-world challenges. Finally, the instructional process effectively utilized cognitive apprenticeship. The teacher acted as a facilitator who provided initial scaffolding,

modeling, and coaching during the early stages of problem decomposition and algorithm design. As the lessons progressed, there was a gradual transfer of responsibility, culminating in the students autonomously debugging their prototypes and presenting their final smart farming outcomes. Together, these theoretical mechanisms illustrate why the integration of EDP and smart farming is highly effective in cultivating higher-order computational thinking.

CONCLUSION AND IMPLICATIONS

This study concludes that integrating the Engineering Design Process (EDP) within a smart farming context significantly enhances the computational thinking (CT) skills of fifth-grade students. The developed instructional activities successfully transformed abstract programming concepts into tangible, real-world applications. By actively engaging in the six-step EDP, ranging from identifying authentic agricultural problems, such as improper soil moisture, to designing, testing, and debugging automated sensor prototypes, students systematically practiced and mastered the four core pillars of CT: decomposition, pattern recognition, abstraction, and algorithm design. The quantitative results corroborated these gains, revealing a statistically significant improvement in both the students' theoretical understanding of computational concepts and their practical problem-solving proficiencies post-intervention.

The overarching impact of this research lies in its pedagogical contribution to primary computing and STEM education. By anchoring computational tasks in authentic agricultural scenarios, the instructional framework successfully bridged the gap between isolated coding exercises and meaningful physical outcomes. Students no longer perceived programming as a detached, screen-based activity; instead, they recognized it as a powerful, applicable tool for innovation and agricultural sustainability. Ultimately, this context-driven EDP model provides educators with a highly effective, constructivist approach to foster computational thinking, preparing young learners to confidently navigate and solve complex, multidisciplinary challenges in the future.

Despite the significant improvements observed, this study acknowledges certain methodological limitations inherent in the one-group pre-test–post-test design. Specifically, the absence of a control group exposes the findings to potential threats to internal validity, such as maturation (students' natural cognitive development over the 12-week intervention), history effects, and testing effects (such as memory bias from using identical pre- and post-test instruments). However, it should be noted that the 12-week interval between test administrations helps mitigate immediate recall bias, and the quantitative test results are strongly triangulated by the students' consistent growth in the qualitative performance-based assessments and final physical projects. While the qualitative evidence strongly links the students' progress directly to their engagement in the EDP-based smart farming activities, it is possible that these external factors partially contributed to the overall score increases. Therefore, future research should employ a controlled experimental or quasi-experimental design to more robustly isolate the effects of the instructional intervention.

Furthermore, the findings revealed distinct developmental trajectories among specific CT components; notably, pattern recognition required extended time to develop, whereas students' understanding of while conditionals improved dramatically. Future studies should investigate the cognitive factors underlying these learning curves. A clearer understanding of these processes would allow educators to design more targeted instructional scaffolding for each computational concept.

USE OF ARTIFICIAL INTELLIGENCE TOOLS

In the preparation of this manuscript, the authors utilized Gemini to assist with spelling and grammar checking, language refinement, reference formatting, and text extraction from large documents. The authors have thoroughly reviewed all AI-assisted outputs and take full responsibility for the accuracy, completeness, and intellectual integrity of the final content.

REFERENCES

- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K–12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1), 48–54. <https://doi.org/10.1145/1929887.1929905>
- Chalmers, C. (2018). Robotics and computational thinking in primary school. *International Journal of Child-Computer Interaction*, 17, 93–100. <https://doi.org/10.1016/j.ijcci.2018.06.005>
- El-Hamamsy, L., Zapata-Cáceres, M., Martín-Barroso, E., Mondada, F., Dehler Zufferey, J., & Bruno, B. (2022). The competent computational thinking test: Development and validation of an unplugged computational thinking test for upper primary school. *Journal of Educational Computing Research*, 60(7), 1818–1866. <https://doi.org/10.1177/07356331221081753>
- Galanti, T. M., & Holincheck, N. M. (2024). Integrating computational thinking in elementary STEM using the engineering design process. *School Science and Mathematics*, 124(4), 279–286. <https://doi.org/10.1111/ssm.12638>
- García-Peñalvo, F. J., & Mendes, A. J. (2018). Exploring the computational thinking effects in pre-university education. *Computers in Human Behavior*, 80, 407–411. <https://doi.org/10.1016/j.chb.2017.12.005>
- Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, 42(1), 38–43. <https://doi.org/10.3102/0013189X12463051>
- Hsu, T. C., Chang, S. C., & Hung, Y. T. (2018). How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education*, 126, 296–310. <https://doi.org/10.1016/j.compedu.2018.07.004>
- Isnaini, R., Budiyanto, C., & Widiastuti, I. (2019). Robotics-based learning to support computational thinking skills in early childhood. *AIP Conference Proceedings*, 2194(1), 020044. <https://doi.org/10.1063/1.5139776>
- Jatuponthianchai, R., Samahito, C., & Butkatanyoo, O. (2022). The effects of computational thinking learning experiences provision on problem solving ability for early childhood students. *Journal of Educational Innovation and Research*, 6(2), 366–380.
- Kareemee, S. (2017). Using the engineering design process to foster creativity and problem-solving skills (Part 1). *IPST Magazine*, 46(209), 23–27.
- Lin, S.Y., Chien, S.Y., Hsiao, C.L., Hsia, C.H., & Chao, K.M. (2020). Enhancing computational thinking capability of preschool children by game-based smart toys. *Electronic Commerce Research and Applications*, 44, Article 101011. <https://doi.org/10.1016/j.elerap.2020.101011>
- Mentzer, N., Becker, K., & Sutton, M. (2015). Engineering design thinking: High school students' performance and knowledge. *Journal of Engineering Education*, 104(4), 417–432. <https://doi.org/10.1002/jee.20105>
- Montiel, H., & Gomez-Zermeño, M. G. (2021). Educational challenges for computational thinking in K–12 education: A systematic literature review of “Scratch” as an innovative programming tool. *Computers*, 10(6), Article 69. <https://doi.org/10.3390/computers10060069>
- Moore, T. J., Glancy, A. W., Tank, K. M., Kersten, J. A., Smith, K. A., & Stohlmann, M. S. (2014). A framework for quality K–12 engineering education: Research and development. *Journal of Pre-College Engineering Education Research*, 4(1), Article 2. <https://doi.org/10.7771/2157-9288.1069>
- National Research Council. (2012). *A framework for K–12 science education: Practices, crosscutting concepts, and core ideas*. The National Academies Press. <https://doi.org/10.17226/13165>
- Paraskevopoulou-Kollia, E. A., Michalakopoulos, C. A., Zygouris, N. C., & Bagos, P. G. (2025). Computational thinking in primary and pre-school children: A systematic review of the literature. *Education Sciences*, 15(8), 985. <https://doi.org/10.3390/educsci15080985>

- Phutwatthana, N. (2018). Engineering design and biomimicry in STEM education. *STOU Education Journal*, 11(2), 31–42. https://so05.tci-thaijo.org/index.php/edjour_stou/article/view/162004
- Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior*, 72, 678–691. <https://doi.org/10.1016/j.chb.2016.08.047>
- Samad, N. A., Osman, K., & Nayan, N. A. (2023). Computational thinking through the engineering design process in chemistry education. *International Journal of Educational Methodology*, 9(4), 771–785. <https://doi.org/10.12973/ijem.9.4.771>
- Sonay, S., & Kiray, C. (2022). Engineering design process-based computational thinking abilities among brilliant and talented students. *International Journal of Advanced Science and Computer Applications*, 1(2). <https://doi.org/10.47679/ijasca.v1i2.14>
- Su, J., & Yang, W. (2023). A systematic review of integrating computational thinking in early childhood education. *Computers and Education Open*, 4, Article 100122. <https://doi.org/10.1016/j.caeo.2023.100122>
- Sutisnawati, A., Rahmawati, Y., Sumantri, M. S., & Lestari, I. (2025). A review of integration of Engineering Design Process (EDP) learning model through STEM approach in elementary schools. *EduBasic Journal: Jurnal Pendidikan Dasar*, 7(1), 129–145. <https://doi.org/10.17509/ebj.v7i1.82001>
- Thipphayasaeng, P., Piriyasurawong, P., & Phanichsiti, S. (2024). Digital twins-based cognitive apprenticeship model in smart agriculture. *International Journal of Interactive Mobile Technologies*, 18(12), 72–84. <https://doi.org/10.3991/ijim.v18i12.46847>
- Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, 25(1), 127–147. <https://doi.org/10.1007/s10956-015-9581-5>
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. <https://doi.org/10.1145/1118178.1118215>
- Yadav, A., Hong, H., & Stephenson, C. (2016). Computational thinking for all: Pedagogical approaches to embedding 21st century problem solving in K–12 classrooms. *TechTrends*, 60(6), 565–568. <https://doi.org/10.1007/s11528-016-0087-7>