MATHEdunesa
Homepage: https://ejournal.unesa.ac.id/index.php/mathedunesa/index
Email: mathedunesa@unesa.ac.id
p-ISSN: 2301-9085; e-ISSN: 2685-7855
Vol.14 No. 3 Tahun 2025
Halaman 780-804

Computational Thinking Abilities of Vocational School Students in Solving Quadratic Function Problems Using Python Programming Language

Misel Rajasyah Hadi Putra[1*], Tatag Yuli Eko Siswono[1] , Novita Vindri Harini[1]

[1]Department of Mathematics, State University of Surabaya, Surabaya, Indonesia

DOI: https://doi.org/10.26740/mathedunesa.v14n3.p780-804

Article History:
Received: 14 May 2025
Revised: 25 August 2025
Accepted: 5 September 2025
Published: 7 November 2025

Keywords:
computational thinking ability, problem solving, quadratic function, python programming language, mathematics education

***Corresponding author:**
misel.23002@mhs.unesa.ac.id

Abstract: The Indonesian Minister of Education and Culture implemented the *Merdeka Belajar Kampus Merdeka* Curriculum in 2019, emphasizing computational thinking (CT) as a crucial 21st-century skill. Despite its importance in mathematics education, most Indonesian mathematics learning has not been oriented toward developing computational thinking abilities. Vocational school students majoring in Software Engineering possess programming knowledge that could potentially enhance their mathematical problem-solving through computational approaches. However, the extent of their CT abilities when applied to mathematical contexts remains unclear. This study aims to describe and analyze the computational thinking abilities of 10th-grade vocational school students majoring in Software Engineering when solving quadratic function problems using Python programming language. This qualitative research employed a case study approach with three purposively selected students representing different proficiency levels (high, moderate, and low) based on standardized programming and mathematics assessment criteria. Data were collected through written tests, structured observations, and semi-structured interviews. The assessment focused on four CT components: decomposition, pattern recognition, abstraction, and algorithmic thinking. The analysis revealed distinct patterns in CT abilities across proficiency levels. High-proficiency students (S1) demonstrated systematic problem decomposition, optimal pattern utilization, effective information filtering, and efficient algorithm development, achieving an average CT score of 91.25. Moderate-proficiency students (S2) showed adequate CT abilities with some limitations in systematic organization and strategic thinking, scoring 78.75 on average. Low-proficiency students (S3) exhibited significant difficulties across all CT components, particularly in problem decomposition and algorithmic thinking, with an average score of 64.25. The findings indicate that students' mathematical foundations significantly influence their CT development when integrated with programming tools. The computational thinking abilities of 10th-grade Software Engineering students vary considerably when solving quadratic function problems with Python assistance. Students with stronger mathematical foundations demonstrate superior CT performance across all components, while those with weaker foundations require substantial scaffolding. These findings highlight the need for differentiated instructional approaches that consider students' varying CT development levels in mathematics education.

INTRODUCTION

Computational thinking represents a fundamental problem-solving approach that involves formulating problems in computational terms and developing systematic solutions, as defined by (Wing, 2006). This cognitive skill extends beyond computer science applications, serving as a critical competency for systematic problem-solving across various disciplines, including mathematics education. According to (Simanjuntak, Armanto, & Dewi, 2023), the integration of computational thinking into mathematics learning supports students in developing structured, logical approaches to complex problem-solving scenarios.

Contemporary educational frameworks emphasize computational thinking as an essential 21st-century skill that enables students to approach problems recursively, establishing pattern regularities and logical calculations that facilitate deeper analysis according to research by (Irawan, Rosjanuardi, & Prabawanto, 2024). This capability aligns closely with mathematics education objectives that prioritize the development of critical thinking and systematic problem-solving abilities, as advocated by the National Council of Teachers of Mathematics and the National Science Teachers Association (Minarni, 2021).

However, according to (Syari, Fatra, & Diwidian, 2024), mathematics education practices in Indonesia remain largely disconnected from computational thinking development. Many educational approaches continue to employ conventional pedagogical methods that inadequately address the four core computational thinking indicators: decomposition, pattern recognition, abstraction, and algorithmic thinking, as identified by researchers such as (Gadanidis, Hughes, Minniti, & White, 2017), (Marcelino, M. J., Pessoa, T., Vieira, C., Salvador, T., & Mendes, 2018), and (Yadav, Gretter, Hambrusch, & Sands, 2017). Traditional teaching methods that emphasize formula memorization and procedural execution , as noted by (Azmi & Yunita, 2022), fail to cultivate the deeper cognitive skills associated with computational thinking development.

Quadratic functions present a particularly suitable mathematical context for computational thinking development due to their conceptual complexity and practical applications. This topic encompasses fundamental mathematical concepts including function relationships, algebraic operations, and graphical representations that connect to real-world problem scenarios , as discussed by (Fadillah, 2019). Nevertheless, research by (Azmi & Yunita, 2022) shows that students frequently encounter difficulties in understanding quadratic function concepts, performing accurate calculations, and connecting prerequisite knowledge to new learning contexts.

Vocational schools offering Software Engineering programs provide a unique educational context where students acquire programming skills alongside traditional academic subjects. These students develop proficiency in programming languages, including Python, which offers significant potential for mathematical problem-solving applications. Python's accessible syntax and robust visualization capabilities make it particularly suitable for exploring mathematical concepts such as quadratic functions through computational approaches. The integration of programming skills with mathematical learning creates opportunities for enhanced conceptual understanding through interactive exploration and visual representation of abstract mathematical relationships.

The Indonesian Merdeka Belajar curriculum initiative specifically identifies computational thinking as a crucial competency for preparing students to meet future challenges in an increasingly digital society, as outlined by (Nau & Sulistyani, 2023). This educational framework recognizes that traditional mathematics instruction must evolve to incorporate computational approaches that reflect contemporary problem-solving methodologies. However, the effectiveness of this integration depends significantly on understanding how students with existing programming knowledge apply computational thinking skills to mathematical contexts.

Given the unique position of Software Engineering students who possess both mathematical knowledge and programming skills, investigating their computational thinking abilities in mathematical problem-solving contexts becomes particularly relevant. These students represent an important demographic for understanding how programming knowledge can enhance mathematical learning and whether existing computational skills transfer effectively to mathematical applications. Understanding their CT abilities can inform educational strategies for integrating programming into mathematics curricula more broadly.

Therefore, this study aims to describe and analyze the computational thinking abilities of 10th-grade vocational school students majoring in Software Engineering when solving quadratic function problems using Python programming language. This investigation addresses the need for empirical evidence regarding how students apply computational thinking skills in mathematics contexts and provides insights for developing more effective integration strategies between programming and mathematics education.

Computational Thinking Framework

Computational thinking has evolved from its origins in computer science to become a fundamental cognitive framework applicable across multiple disciplines. According to (Wing, 2006) initially defined computational thinking as a problem-solving process that involves formulating problems in computational terms and developing effective solutions through systematic approaches. This definition has been expanded by subsequent researchers who recognize computational thinking as a transferable skill set essential for navigating complex problem-solving scenarios in various academic and professional contexts.

Research by (Angeli & Giannakos, 2020) conceptualize computational thinking as a thinking process that originates from computer science principles but extends its applicability across diverse disciplinary boundaries. Similarly, (Syari et al., 2024) describes computational thinking as a comprehensive problem-solving methodology that begins with problem formulation and proceeds through systematic decomposition into manageable components.

The theoretical framework for computational thinking encompasses four fundamental components that work synergistically to support effective problem-solving. Decomposition involves breaking complex problems into smaller, more manageable parts that can be addressed systematically. Pattern recognition requires identifying similarities, regularities, or recurring elements within problems or solution approaches. Abstraction focuses on identifying and extracting relevant information while filtering out unnecessary details that may complicate the solution process. Algorithmic thinking involves developing clear, sequential steps that lead to systematic problem resolution, as outlined by (Syari et al., 2024) and (Barr & Stephenson, 2011).

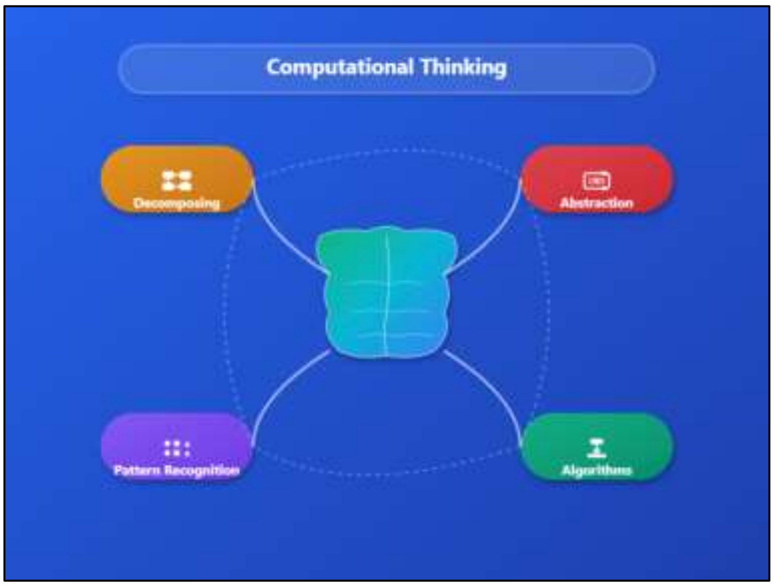


Figure 1. Computational Thinking Components Framework
Source: Made with Figma

Research conducted by (Irawan et al., 2024) expand this framework by emphasizing that computational thinking enables recursive problem-solving approaches, where students can address complex challenges by establishing pattern regularities and implementing logical calculations that facilitate systematic analysis. This recursive capability proves particularly valuable in mathematics education, where students must process and transform information to solve increasingly complex problems.

Mathematical Problem-Solving Theory

Mathematical problem-solving has emerged as a central focus in mathematics education worldwide, reflecting its importance for developing students' analytical and reasoning capabilities. The seminal work by (polya, 1957) seminal work established a four-stage framework for mathematical problem-solving that continues to influence contemporary educational approaches: understanding the problem, devising solution plans, executing the plan, and reviewing results for accuracy and reasonableness.

According to the National Council of Teachers of Mathematics, they emphasize that problem-solving serves dual purposes in mathematics education, functioning both as a learning objective and as a pedagogical method for developing mathematical understanding. Through systematic problem-solving experiences, students develop deeper conceptual comprehension of mathematical ideas while strengthening connections between different mathematical concepts and real-world applications.

Research by (Schoenfeld, 2016) identifies four critical components that contribute to successful mathematical problem-solving: foundational mathematical knowledge, strategic problem-solving approaches, metacognitive regulation and self-monitoring, and positive beliefs and dispositions toward mathematics. These components interact dynamically to influence students' overall problem-solving effectiveness and their willingness to persist through challenging mathematical scenarios.

Studies conducted by (Killpatrick, Swafford, & Findell, 2010) propose five mathematical proficiency strands essential for successful problem-solving: conceptual understanding, procedural fluency, strategic competence, adaptive reasoning, and productive disposition. Strategic competence, which involves the ability to formulate, represent, and solve mathematical problems effectively, shows particular alignment with computational thinking principles.

In the context of quadratic function learning, mathematical problem-solving involves applying quadratic relationships to diverse situations, including optimization problems, equation solving, and graphical analysis, as discussed by (Fadillah, 2019). The integration of computational thinking approaches into quadratic function problem-solving can enhance students' systematic solution strategies while providing opportunities for deeper conceptual exploration through technological tools.

Quadratic Function Theory in Mathematics

Quadratic functions represent an important topic in secondary school mathematics curricula. A quadratic function is defined as a function with the general form $f(x) = ax^2 + bx + c$, where a, b, dan c are constants and $a \neq 0$, as explained by (Fadillah, 2019). This function has a parabolic graph that can open upward (if $a > 0$) or downward (if $a < 0$).

According to (Fadillah, 2019) state that quadratic function material encompasses several basic concepts and prerequisite materials such as function concepts, algebraic operations in functions, and function graphs. Students need to understand the properties of quadratic functions, including: (1) The vertex of the parabola, (2) The axis of symmetry, (3) The roots of the quadratic equation, (4) The discriminant and its types, and (5) The relationship between coefficients and roots of quadratic equations.

Research by (Azmi & Yunita, 2022) identifies several difficulties faced by students in learning quadratic functions, including: (1) Inability to understand and apply quadratic function concepts to problems, (2) Difficulties in calculating or operating on quadratic function problems, and (3) Inability to recall previously learned material.

These difficulties are often caused by conventional teaching approaches, where students tend to learn procedurally without deep conceptual understanding.

Integration of Programming in Mathematics Education

The integration of programming languages into mathematics education represents an emerging pedagogical approach designed to enhance conceptual understanding and problem-solving capabilities. Python, with its intuitive syntax and powerful mathematical libraries, has gained recognition as an effective tool for mathematics education integration, as noted by (Marcelino, M. J., Pessoa, T., Vieira, C., Salvador, T., & Mendes, 2018).

Research conducted by (Yadav et al., 2017) demonstrates that programming language integration in mathematics education can significantly enhance students' computational thinking development. Programming environments allow students to apply mathematical concepts in authentic contexts while providing opportunities to visualize abstract mathematical relationships through interactive exploration and graphical representation.

For quadratic function learning specifically, Python programming offers several educational advantages. Students can utilize programming to calculate function values for specific inputs, determine equation roots through numerical methods, create dynamic visualizations of function behavior, and analyze function properties through both numerical and graphical approaches. These capabilities provide students with multiple representation systems for understanding quadratic relationships while developing computational skills simultaneously.

According to (Gadanidis et al., 2017) emphasize that programming integration in mathematics education should focus not only on developing technical programming skills but also on deepening mathematical understanding and cultivating higher-order thinking capabilities. This dual focus ensures that programming tools serve mathematical learning objectives rather than becoming isolated technical skills.

Previous Research on Computational Thinking in Mathematics Education

The intersection of computational thinking and mathematics education has emerged as a critical area of investigation, particularly as educational systems worldwide grapple with integrating 21st-century digital competencies into traditional academic subjects. Research in this domain reveals both promising opportunities and significant implementation challenges that warrant careful examination.

The current state of computational thinking integration in mathematics education presents a complex landscape. Indonesian educational contexts, in particular, face substantial hurdles in developing students' computational thinking capabilities within mathematical learning environments. Research consistently demonstrates that traditional mathematics pedagogical approaches inadequately support the development of computational thinking skills, creating a significant gap between educational goals and classroom realities, as found by (Syari et al., 2024) and (Maifi, Anwar, & Ahmad, 2021). This disconnect is particularly concerning given the Indonesian government's emphasis on computational thinking as a crucial 21st-century competency through the Merdeka Belajar curriculum initiative.

Empirical investigations into students' computational thinking abilities reveal troubling patterns in skill development. Studies examining Indonesian students' performance across various mathematical contexts consistently identify weaknesses in fundamental computational thinking components, including problem decomposition, pattern recognition, abstraction, and algorithmic reasoning, as reported by (Kamil, Imami, & Abadi, 2021). These deficiencies appear to stem from educational approaches that prioritize procedural knowledge acquisition over the development of systematic thinking processes that characterize effective computational problem-solving.

However, emerging research suggests that strategic integration of programming activities within mathematics education can significantly enhance students' computational thinking development. According to (Harangus & Kátai, 2020) demonstrated that students who engage in programming-integrated mathematics instruction show measurable improvements in computational thinking capabilities compared to peers receiving traditional instruction. This finding supports the theoretical proposition that programming tools can serve as cognitive amplifiers for mathematical thinking when properly integrated into pedagogical frameworks.

The relationship between computational thinking and broader cognitive abilities presents another crucial dimension of this research domain. Educational researchers have established meaningful connections between computational thinking skills and critical thinking, problem-solving, and mathematical reasoning capabilities that extend beyond specific technological applications, as demonstrated by (Cahdriyana & Richardo, 2020). These connections suggest that computational thinking development may have cascading effects on students' overall mathematical proficiency and analytical capabilities.

Specific mathematical topics, such as quadratic functions, have proven particularly suitable for computational thinking integration research. According to (Mubarokah, Pambudi, Lestari, Kurniati, & Jatmiko, 2023) found that students who developed stronger computational thinking skills through programming-enhanced instruction demonstrated superior performance in understanding and applying quadratic function concepts. Their research revealed that computational approaches help students overcome traditional conceptual barriers associated with quadratic function learning while building stronger connections between abstract mathematical relationships and practical problem-solving applications.

Despite these promising findings, significant research gaps remain in understanding how computational thinking manifests across different student populations and mathematical contexts. Most existing studies focus on general student populations without considering how prior programming experience might influence computational thinking development in mathematical contexts. Additionally, limited research examines how computational thinking abilities vary among students with different mathematical proficiency levels, particularly in specialized educational contexts such as vocational programs where students possess existing technical skills.

The research landscape also reveals insufficient attention to the specific mechanisms through which programming tools enhance mathematical understanding. While studies demonstrate positive correlations between programming integration and computational thinking development, fewer investigations examine the detailed cognitive processes through which students translate mathematical concepts into computational solutions. This gap is particularly significant for educators seeking evidence-based strategies for implementing computational thinking instruction in mathematics curricula.

Furthermore, most existing research employs broad assessment approaches that may not capture the nuanced ways computational thinking manifests in specific mathematical problem-solving contexts. There remains a need for more detailed, qualitative investigations that examine how students apply computational thinking components in real-time problem-solving scenarios, particularly when using programming tools to explore mathematical concepts.

These research limitations highlight the importance of investigating computational thinking abilities among specific student populations, such as vocational students with existing programming knowledge, who represent unique cases for understanding how technical skills transfer to mathematical contexts. Such investigations can provide crucial insights for developing more effective integration strategies that leverage students' existing capabilities while addressing identified weaknesses in computational thinking development.

Research Framework

Based on the literature review, this research operates within a framework that views computational thinking as a bridge between programming knowledge and mathematical problem-solving. The study examines how Software Engineering students apply their existing programming skills to mathematical contexts, specifically focusing on how the four computational thinking components manifest in quadratic function problem-solving scenarios using Python programming tools.

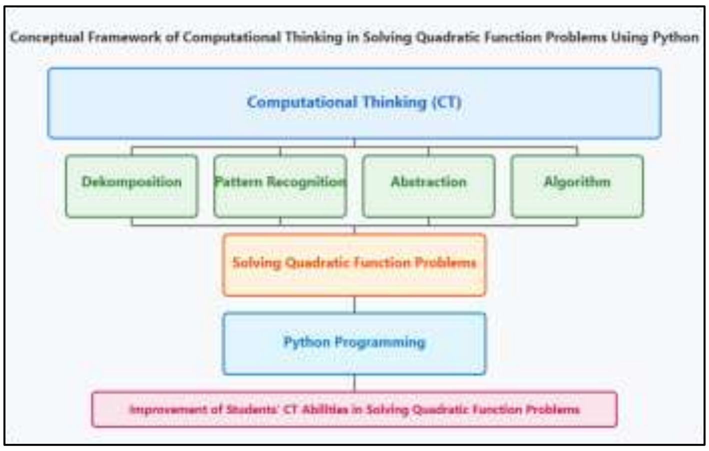

Conceptual Framework of Computational Thinking in Solving Quadratic Function Problems Using Python
Computational Thinking (CT)
Dekomposition
Pattern Recognition
Abstraction
Algorithm
Solving Quadratic Function Problems
Python Programming
Improvement of Students' CT Abilities in Solving Quadratic Function Problems

Figure 2. Conceptual Research Framework
Source: Made with Figma

METHOD

Research Design and Approach

This research employs a qualitative case study approach to provide in-depth understanding of computational thinking abilities among vocational school students. According to Creswell that described by (Mackiewicz, 2018), qualitative research focuses on detailed exploration and comprehensive understanding of specific phenomena within their natural contexts. The case study method allows for intensive examination of computational thinking manifestation in mathematical problem-solving scenarios while maintaining the complexity and richness of real educational settings.

Participant Selection and Characteristics

Three 10th-grade vocational high school students majoring in Software Engineering were selected through purposive sampling based on standardized assessment of their programming and mathematics proficiency levels. The selection process involved administering comprehensive assessments in both domains to ensure representation across different ability levels.

Programming Proficiency Assessment Criteria.

Students were evaluated on Python programming fundamentals including variable manipulation, control structures implementation, function development, and basic data processing capabilities. Assessment items required students to demonstrate understanding of programming logic, syntax accuracy, and problem-solving approaches using Python.

Mathematics Proficiency Assessment Criteria

Students completed assessments covering quadratic function concepts, including function properties identification, equation solving techniques, graphical interpretation skills, and application of quadratic relationships to contextual problems.

Proficiency categories were established using the Merdeka Curriculum assessment framework, as outlined by Directorate General of Higher Education in year of 2022 as in Table 1.

Table 1. Proficiency Level Classification

No.	Score Range	Proficiency Category
1.	85 < score ≤ 100	High
2.	65 < score ≤ 85	Moderate
3.	0 < score ≤ 65	Low

From the assessment results, three students were selected representing each proficiency category: one high-proficiency student (S1), one moderate-proficiency student (S2), and one low-proficiency student (S3). This selection strategy ensures comprehensive representation of computational thinking abilities across different competency levels.

Research Instruments and Procedures

This research utilized five primary instruments to collect comprehensive data regarding students' computational thinking capabilities. Instrument development incorporated indicators of CT capabilities and quadratic function conceptual material. Prior to implementation, all instruments were validated by two experts in mathematics education and informatics to ensure content validity.

The research procedure commenced with a preparation phase encompassing instrument development, validation, and subject selection. Subsequently, during the implementation phase, students completed written tests while being observed by researchers, followed by in-depth interviews. The acquired data were then analyzed through data reduction, data presentation, and conclusion drawing. The following are details and examples of each instrument used.

Python Programming Proficiency Assessment (pre-test)

This pre-assessment evaluated students' foundational programming knowledge through practical coding tasks. This assessment consisted of one basic programming question covering concepts of variables, control structures, functions, and simple data manipulation.

"Create a Python function named 'calculate_average' that accepts a list of numbers as a parameter and returns the average value of the list. Then, test the function with the list [10, 15, 20, 25, 30]."

Assessment criteria included syntax accuracy, algorithm efficiency, and program output correctness.

Mathematics Proficiency Test (pre- test)

This pre-assessment examined students' understanding of quadratic function concepts through analytical and graphical problems This assessment contained one question on quadratic function concepts and applications.

"Determine the vertex point, axis of symmetry, and maximum or minimum value of the quadratic function $f(x) = -2x^2 + 8x - 3$. *Draw the graph of this function."*

Assessment was based on conceptual accuracy, calculation precision, and answer completeness.

Computational Thinking Test in Quadratic Function Problem-Solving (core)

This core assessment presented contextual problems requiring integration of mathematical understanding with programming implementation. The assessment was specifically designed to evaluate all four computational thinking components within authentic problem-solving contexts.

"A manufacturing company discovered that their daily profit (in thousands of rupiah) can be modeled with the function $P(x) = -2x^2 + 120x - 300$, where x is the number of products (in hundreds of units) produced per day.

a) Create a Python program to determine how many products should be produced for maximum profit.

b) Modify the program to calculate the maximum profit obtainable.

c) Create a visualization of the profit function graph using matplotlib.

d) If production costs increase such that the x^2 coefficient changes to -3 how does this affect the optimal production quantity and maximum profit? Modify your program to answer this question."

Assessment encompassed students' capabilities in decomposing problems, identifying patterns, performing abstraction, and developing solution algorithms.

Observation Guidelines

The observation guidelines contained indicators of CT capabilities observed while students completed the problem-solving test. Observation was conducted using structured observation sheets of CT indicator with source from developed based on computational thinking literature and validated by expert review.

Table 2. Computational Thinking Indicators Observation

Decomposition	Pattern Recognition	Abstraction	Algorithm
Unable to break down the problem into smaller parts	Unable to identify patterns in the problem	Unable to identify relevant information	Unable to develop solution steps
Breaks down a small portion of the problem into smaller parts but not systematically	Identifies patterns in a limited way and cannot utilize them	Identifies a small portion of relevant information but remains focused on unnecessary details	Develops solution steps but not systematically and sequentially
Breaks down most of the problem into smaller parts but less systematically	Identifies patterns well but sub-optimally utilizes them	Identifies most relevant information but still considers some unnecessary details	Develops systematic solution steps but less efficiently
Breaks down the problem into smaller parts systematically and comprehensively	Identifies patterns accurately and optimally utilizes them for solutions	Identifies all relevant information and ignores unnecessary details	Develops systematic, sequential, and efficient solution steps

Interview Guidelines

Semi-structured interviews were conducted after students completed the test to explore their thinking processes and strategies more deeply. The interview guidelines contained questions covering all four CT indicators. Interview questions:

Table 3. Interview Questions

Decomposition	Pattern Recognition	Abstraction	Algorithm
How did you begin solving this problem?	Did you see any similarities or specific patterns in this problem with problems you've solved previously?	What information did you ignore because it was considered irrelevant?	Why did you choose that approach?
What parts did you identify from this problem?	How did you use those patterns to assist in the solution?	Explain the steps you used to solve this problem.	How did you implement the mathematical solution into Python code?
	What information did you consider important in this problem?		

Data Collection Techniques

Data in this research were collected through three techniques: observation, written tests, and interviews. Observation was conducted to monitor students' processes in solving quadratic function problems using Python, focusing on the application of four CT capability indicators. Written tests were used to assess students' capabilities in solving quadratic function problems with Python assistance, and interviews were conducted to obtain deeper information about students' thinking processes and strategies.

Data Analysis Techniques

Data analysis was performed following Miles and Huberman's model, as described by (Dull & Reinhardt, 2014), which includes three stages: data reduction, data presentation, and conclusion drawing. To ensure data validity, researchers conducted method triangulation by comparing data obtained from problem-solving test results with interview results. Additionally, researchers also performed member checking by confirming analysis results with research subjects.

RESULTS AND DISCUSSION

Results

This research aimed to describe the computational thinking capabilities of tenth-grade vocational high school students majoring in Software Engineering in solving quadratic function problems with Python programming language assistance. Data collection was conducted through written tests, observation, and interviews with three research subjects selected based on programming and mathematics proficiency criteria.

Research Subject Profiles

Based on Python programming proficiency and mathematics proficiency test results, three research subjects coded S1, S2, and S3 were selected, representing high, moderate, and low proficiency categories. The profiles of the three research subjects are presented in Table 4.

Table 4. Research Subject Background and Proficiency Levels

Subject Code	Programming Proficiency Score	Mathematics Proficiency Score	Proficiency Category	Key Characteristics
S1	92	88	High	Strong algorithmic thinking, solid mathematical reasoning
S2	78	72	Moderate	Good basic skills with some conceptual gaps
S3	63	58	Low	Limited programming experience, struggles with mathematical abstractions

Computational Thinking Assessment Results

The core assessment revealed significant variations in computational thinking abilities across the four key components. Each student's performance was evaluated using the structured rubric, yielding comprehensive profiles of their computational thinking development.

Table 5. Computational Thinking Capability Assessment Results

Subject	Decomposition	Pattern Recognition	Abstraction	Algorithm	Average	Category
S1	93	90	87	95	91.25	High
S2	82	75	78	80	78.75	Moderate
S3	65	60	70	62	64.25	Low

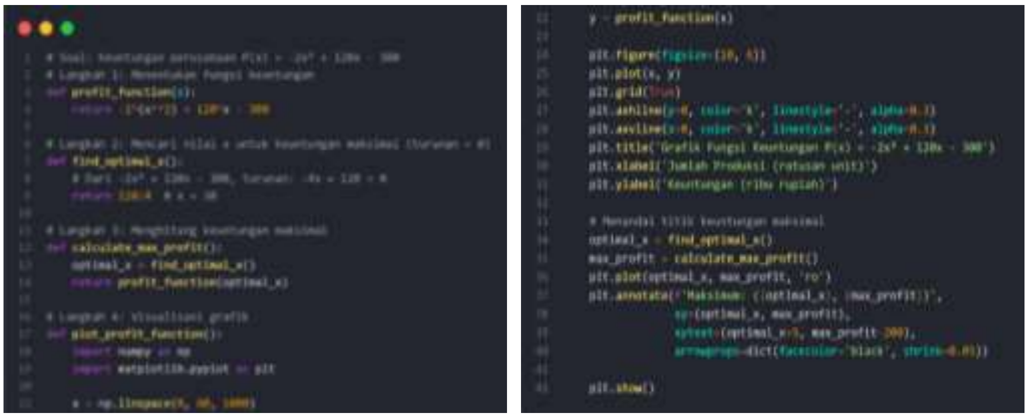
These results demonstrate clear differentiation in computational thinking abilities that align with students' foundational proficiency levels while revealing specific patterns within each component.

Analysis of Subject S1's Computational Thinking Capabilities

Subject S1 demonstrated high computational thinking capabilities in solving quadratic function problems with Python assistance. The following is an analysis of subject S1's capabilities based on four indicators.

Decomposition Capability

Subject S1 was able to break down quadratic function problems into smaller parts systematically and comprehensively. This was evident from S1's work on the manufacturing company profit model question shown in Figure 3.



```python
# Soal: keuntungan perusahaan P(x) = -2x² + 120x - 300
# Langkah 1: Menentukan fungsi keuntungan
def profit_function(x):
    return -2*(x**2) + 120*x - 300

# Langkah 2: Mencari nilai x untuk keuntungan maksimal (turunan = 0)
def find_optimal_x():
    # Dari -2x² + 120x - 300, turunan: -4x + 120 = 0
    return 120/4  # x = 30

# Langkah 3: Menghitung keuntungan maksimal
def calculate_max_profit():
    optimal_x = find_optimal_x()
    return profit_function(optimal_x)

# Langkah 4: Visualisasi grafik
def plot_profit_function():
    import numpy as np
    import matplotlib.pyplot as plt

    x = np.linspace(0, 60, 1000)
    y = profit_function(x)

    plt.figure(figsize=(10, 6))
    plt.plot(x, y)
    plt.grid(True)
    plt.axhline(y=0, color='k', linestyle='-', alpha=0.3)
    plt.axvline(x=0, color='k', linestyle='-', alpha=0.3)
    plt.title('Grafik Fungsi Keuntungan P(x) = -2x² + 120x - 300')
    plt.xlabel('Jumlah Produksi (ratusan unit)')
    plt.ylabel('Keuntungan (ribu rupiah)')

    # Menandai titik keuntungan maksimal
    optimal_x = find_optimal_x()
    max_profit = calculate_max_profit()
    plt.plot(optimal_x, max_profit, 'ro')
    plt.annotate(f'Maksimum: ({optimal_x}, {max_profit})',
                 xy=(optimal_x, max_profit),
                 xytext=(optimal_x+5, max_profit-200),
                 arrowprops=dict(facecolor='black', shrink=0.05))

    plt.show()
```

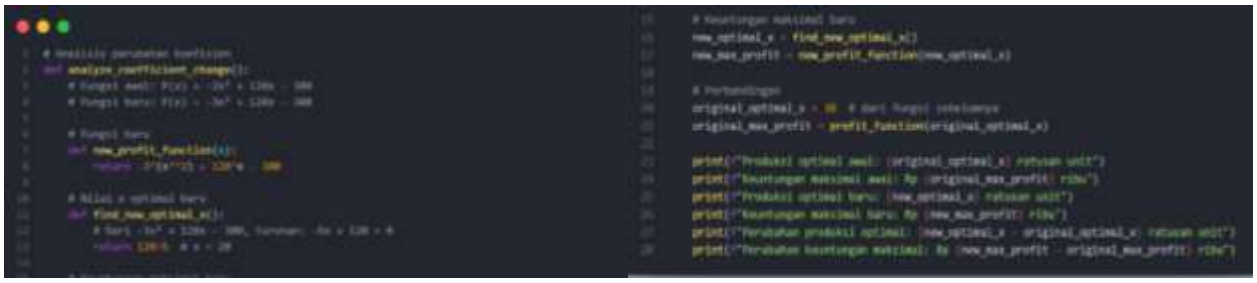
Figure 3. Problem Decomposition by Subject S1
Source: VSCode Text Editor

During the interview, subject S1 explained their decomposition process:

> *"I broke the problem down into several steps. First, defining the profit function. Second, using calculus to find the optimal x value from the first derivative equals zero. Third, calculating the maximum profit by inputting the optimal x value into the profit function. Finally, visualizing the graph to verify the results."*

Pattern Recognition Capability

Subject S1 was able to identify patterns accurately and utilize them optimally for solutions. In the profit function modification question, S1 quickly recognized the pattern of changes occurring and adapted their solution as shown in Figure 4.



```python
# Analisis perubahan koefisien
def analyze_coefficient_change():
    # Fungsi awal: P(x) = -2x² + 120x - 300
    # Fungsi baru: P(x) = -3x² + 120x - 300

    # Fungsi baru
    def new_profit_function(x):
        return -3*(x**2) + 120*x - 300

    # Nilai x optimal baru
    def find_new_optimal_x():
        # Dari -3x² + 120x - 300, turunan: -6x + 120 = 0
        return 120/6  # x = 20

    # Keuntungan maksimal baru
    new_optimal_x = find_new_optimal_x()
    new_max_profit = new_profit_function(new_optimal_x)

    # Perbandingan
    original_optimal_x = 30  # dari fungsi sebelumnya
    original_max_profit = profit_function(original_optimal_x)

    print(f"Produksi optimal awal: {original_optimal_x} ratusan unit")
    print(f"Keuntungan maksimal awal: Rp {original_max_profit} ribu")
    print(f"Produksi optimal baru: {new_optimal_x} ratusan unit")
    print(f"Keuntungan maksimal baru: Rp {new_max_profit} ribu")
    print(f"Perubahan produksi optimal: {new_optimal_x - original_optimal_x} ratusan unit")
    print(f"Perubahan keuntungan maksimal: Rp {new_max_profit - original_max_profit} ribu")
```

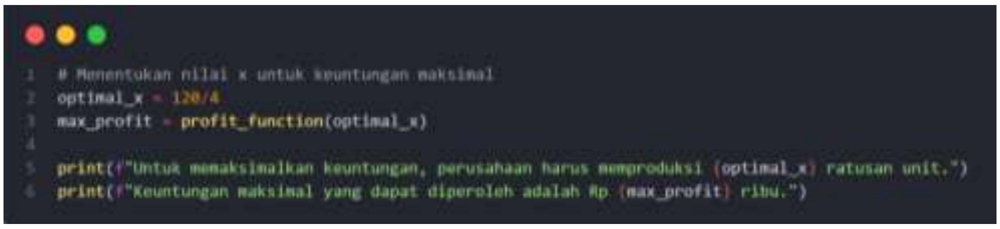
Figure 4. Pattern Recognition by Subject S1
Source: VSCode Text Editor

Based on interview results, subject S1 explained:

> "*I saw the pattern that if the x^2 coefficient changes from -2 to -3, then the optimal point calculation will change. I used the same pattern as before, finding the first derivative and setting it equal to zero. If the x^2 coefficient is -3, then its derivative is $-6x + 120$, so the optimal $x = 20$. This shows a pattern that the more negative the x^2 coefficient, the smaller the optimal x value, which logically means the company must reduce production when production costs increase.*"

Abstraction Capability

Subject S1 was able to identify and extract relevant information and ignore unnecessary details well. This was evident from how S1 modeled the quadratic function problem in computational form as shown in Figure 5.



```python
# Menentukan nilai x untuk keuntungan maksimal
optimal_x = 120/4
max_profit = profit_function(optimal_x)

print(f"Untuk memaksimalkan keuntungan, perusahaan harus memproduksi {optimal_x} ratusan unit.")
print(f"Keuntungan maksimal yang dapat diperoleh adalah Rp {max_profit} ribu.")
```

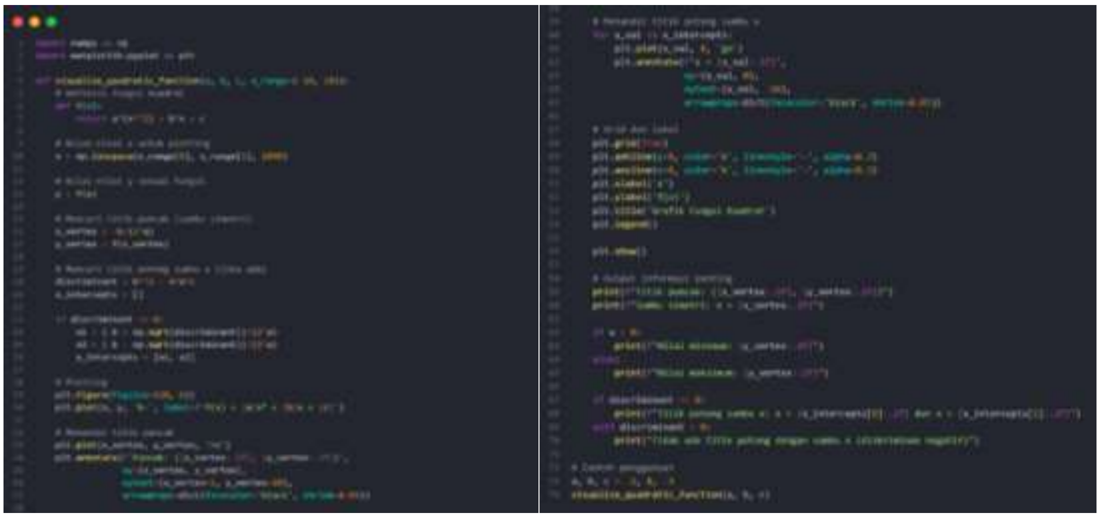
Figure 5. Abstraction by Subject S1
Source: VSCode Text Editor

From the interview results, subject S1 explained their abstraction process:

"In this problem, what's relevant are the quadratic function coefficients and the formula for finding the optimal x value. I ignored details about how the company produces goods or its marketing process because that's irrelevant to the mathematical solution. I also ignored other values on the graph and only focused on the maximum point."

Algorithm Capability

Subject S1 was able to develop ordered steps to solve problems very well. The algorithm developed by S1 to visualize quadratic functions is shown in Figure 6.



```
[illegible]
```

Figure 6. Algorithm Development by Subject S1
Source: VSCode Text Editor

In the interview, subject S1 explained:

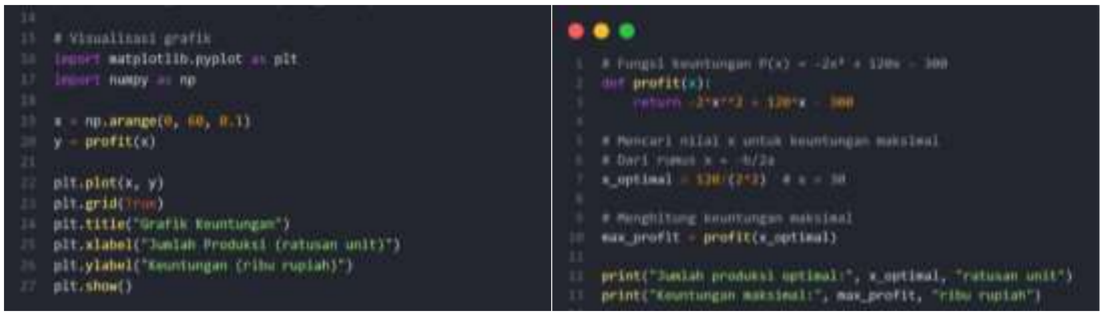
"I created a general function for visualizing quadratic functions that can be used for various coefficient values a, b, and c. This algorithm first defines the function, then calculates important values such as the vertex point and x-axis intercepts, then visualizes them. I also added text output for important information to make it easier to understand."

Analysis of Subject S2's Computational Thinking Capabilities

Subject S2 demonstrated computational thinking capabilities in the moderate category when solving quadratic function problems with Python assistance.

Decomposition Capability

Subject S2 was able to break down most of the problem into smaller parts, but less systematically compared to S1. S2's work on the manufacturing company profit model question is shown in Figure 7.



```python
# Visualisasi grafik
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 60, 0.1)
y = profit(x)

plt.plot(x, y)
plt.grid(True)
plt.title("Grafik Keuntungan")
plt.xlabel("Jumlah Produksi (ratusan unit)")
plt.ylabel("Keuntungan (ribu rupiah)")
plt.show()
```

```python
# Fungsi keuntungan P(x) = -2x² + 120x - 300
def profit(x):
    return -2*x**2 + 120*x - 300

# Mencari nilai x untuk keuntungan maksimal
# Dari rumus x = -b/2a
x_optimal = 120/(2*2)  # x = 30

# Menghitung keuntungan maksimal
max_profit = profit(x_optimal)

print("Jumlah produksi optimal:", x_optimal, "ratusan unit")
print("Keuntungan maksimal:", max_profit, "ribu rupiah")
```

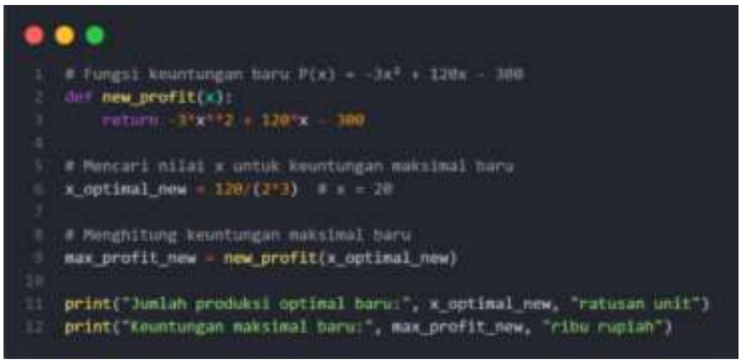
Figure 7. Problem Decomposition by Subject S2
Source: VSCode Text Editor

From the interview results, subject S2 explained:

> "*I divided this problem into several parts. First, defining the profit function. Then finding the optimal x value using the parabola vertex formula. After that, calculating the maximum profit by inputting the optimal x value into the function. Finally, I visualized the graph.*"

Pattern Recognition Capability

Subject S2 was able to identify patterns well but sub-optimally utilized them. In the profit function modification question, S2 recognized the changes occurring but did not analyze them as deeply as shown in Figure 8.



```python
# Fungsi keuntungan baru P(x) = -3x² + 120x - 300
def new_profit(x):
    return -3*x**2 + 120*x - 300

# Mencari nilai x untuk keuntungan maksimal baru
x_optimal_new = 120/(2*3)  # x = 20

# Menghitung keuntungan maksimal baru
max_profit_new = new_profit(x_optimal_new)

print("Jumlah produksi optimal baru:", x_optimal_new, "ratusan unit")
print("Keuntungan maksimal baru:", max_profit_new, "ribu rupiah")
```

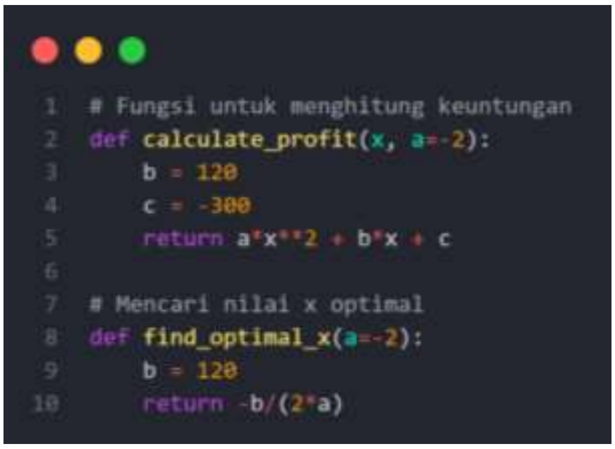
Figure 8. Pattern Recognition by Subject S2
Source: VSCode Text Editor

From the interview results, subject S2 explained:

> "*I saw that the x^2 coefficient changed from -2 to -3, so the optimal x value would also change. I used the same* $x = -\frac{b}{2a}$*, where a is now* 3*, so the optimal* x *becomes* 20."

Abstraction Capability

Subject S2 was able to identify relevant information well but sometimes still included some unnecessary details. This was evident from how S2 modeled the problem as shown in Figure 9.



```python
# Fungsi untuk menghitung keuntungan
def calculate_profit(x, a=-2):
    b = 120
    c = -300
    return a*x**2 + b*x + c

# Mencari nilai x optimal
def find_optimal_x(a=-2):
    b = 120
    return -b/(2*a)
```

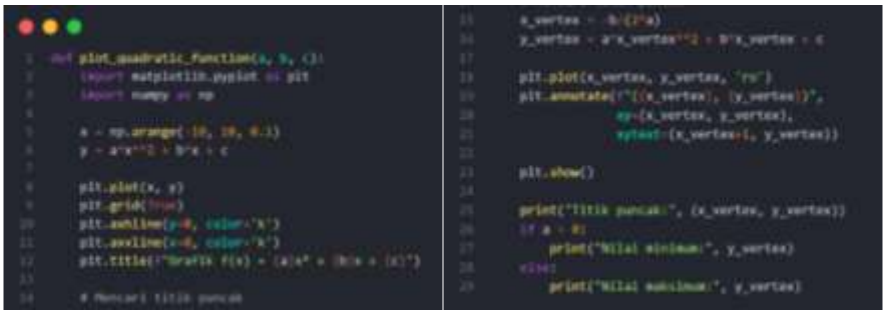
Figure 9. Abstraction by Subject S2
Source: VSCode Text Editor

From the interview results, subject S2 explained:

> *"I focused on the profit function and formulas for finding the optimal point. I created a function that can accept different a value to facilitate analysis of coefficient changes. But I still included b and c values within the function even though their values don't change.*

Algorithm Capability

Subject S2 was able to develop ordered steps to solve problems well, but less comprehensively compared to S1. The algorithm developed by S2 to visualize quadratic functions is shown in Figure 10.



```python
def plot_quadratic_function(a, b, c):
    import matplotlib.pyplot as plt
    import numpy as np

    x = np.arange(-10, 10, 0.1)
    y = a*x**2 + b*x + c

    plt.plot(x, y)
    plt.grid(True)
    plt.axhline(y=0, color='k')
    plt.axvline(x=0, color='k')
    plt.title(f"Grafik f(x) = {a}x² + {b}x + {c}")

    # Mencari titik puncak
    x_vertex = -b/(2*a)
    y_vertex = a*x_vertex**2 + b*x_vertex + c

    plt.plot(x_vertex, y_vertex, 'ro')
    plt.annotate(f"({x_vertex}, {y_vertex})",
                 xy=(x_vertex, y_vertex),
                 xytext=(x_vertex+1, y_vertex))

    plt.show()

    print("Titik puncak:", (x_vertex, y_vertex))
    if a > 0:
        print("Nilai minimum:", y_vertex)
    else:
        print("Nilai maksimum:", y_vertex)
```

Figure 10. Algorithm Development by Subject S2
Source: VSCode Text Editor

In the interview, subject S2 explained:

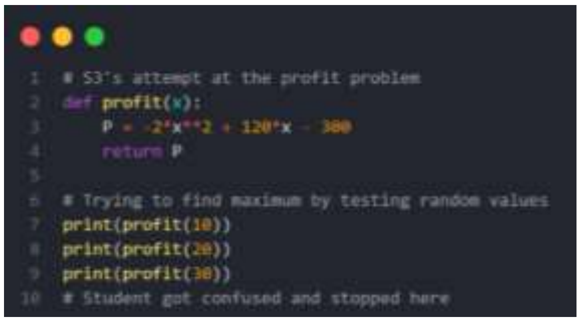
> *"I created a function to visualize quadratic functions with parameters* a, b, *and c. This algorithm draws the graph, determines the vertex point, and displays important information. I didn't add calculation of* $x-axis$ *intercepts because I didn't think they were very important for the profit problem."*

Analysis of Subject S3's Computational Thinking Capabilities

Subject S3 demonstrated computational thinking capabilities in the low category when solving quadratic function problems with Python assistance. The detailed analysis reveals specific areas where S3 struggled, providing crucial insights for educators working with students who have similar challenges.

Decomposition Capability

Subject S3 showed significant difficulties in breaking down the quadratic function problem into manageable components. Unlike S1 and S2 who could identify distinct steps, S3 attempted to solve the entire problem as one unit without systematic breakdown. S3's work on the manufacturing company profit model question is shown in Figure 11.



```python
# S3's attempt at the profit problem
def profit(x):
    P = -2*x**2 + 120*x - 380
    return P

# Trying to find maximum by testing random values
print(profit(10))
print(profit(20))
print(profit(30))
# Student got confused and stopped here
```

Figure 11. Problem Decomposition Attempt by Subject S3
Source: VSCode Text Editor

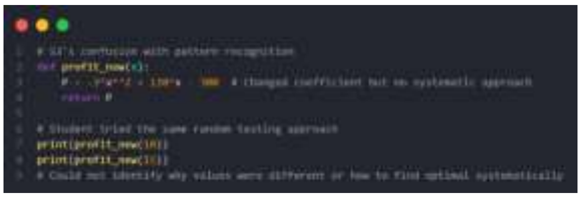
During the interview, subject S3 explained their struggle with decomposition:

"I knew I needed to find the maximum profit, but I wasn't sure how to break it down. I tried plugging in different x values to see which gave the highest profit, but I realized there were too many possibilities to check. I got stuck because I couldn't figure out what steps to take first."

This difficulty in decomposition appears to stem from S3's limited understanding of how mathematical concepts translate into programming logic. The student could not identify the intermediate steps needed (finding the derivative, setting it to zero, solving for x) and instead resorted to a trial-and-error approach that quickly became overwhelming.

Pattern Recognition Capability

Subject S3 struggled significantly with pattern recognition, showing minimal ability to identify mathematical relationships or programming patterns. When faced with the profit function modification question (changing the coefficient from -2 to -3), S3 could not recognize the underlying pattern or adapt their solution approach, as shown in Figure 12.



```python
# S3's confusion with pattern recognition
def profit_new(x):
    P = -3*x**2 + 120*x - 380  # Changed coefficient but no systematic approach
    return P

# Student tried the same random testing approach
print(profit_new(10))
print(profit_new(15))
# Could not identify why values were different or how to find optimal systematically
```

Figure 12. Pattern Recognition Struggle by Subject S3
Source: VSCode Text Editor

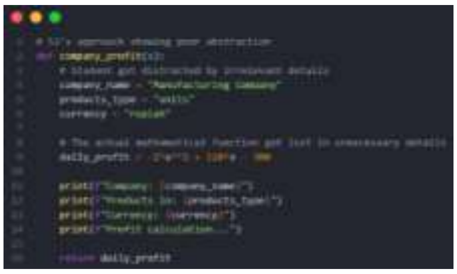
From the interview results, subject S3 explained:

"When the coefficient changed to -3, I saw that the answers were different, but I couldn't understand why or what it meant. I tried the same way as before, testing different numbers, but I couldn't see any pattern in how the change affected the optimal point. I didn't know how to connect the mathematical change to the programming solution."

This struggle with pattern recognition indicates that S3 has difficulty connecting mathematical concepts across different contexts. The student could not recognize that the same optimization principles apply regardless of coefficient values, nor could they identify programming patterns that could be reused with modifications.

Abstraction Capability

Subject S3 demonstrated particular difficulty with abstraction, showing an inability to distinguish between relevant and irrelevant information in the problem context. This was evident in how S3 approached the profit modelling task, as shown in Figure 13.



```python
# S3's approach showing poor abstraction
def company_profit(x):
    # Student got distracted by irrelevant details
    company_name = "Manufacturing Company"
    products_type = "units"
    currency = "rupiah"

    # The actual mathematical function got lost in unnecessary details
    daily_profit = -2*x**2 + 120*x - 300

    print(f"Company: {company_name}")
    print(f"Products in: {products_type}")
    print(f"Currency: {currency}")
    print("Profit calculation...")

    return daily_profit
```

Figure 13. Abstraction Difficulties by Subject S3
Source: VSCode Text Editor

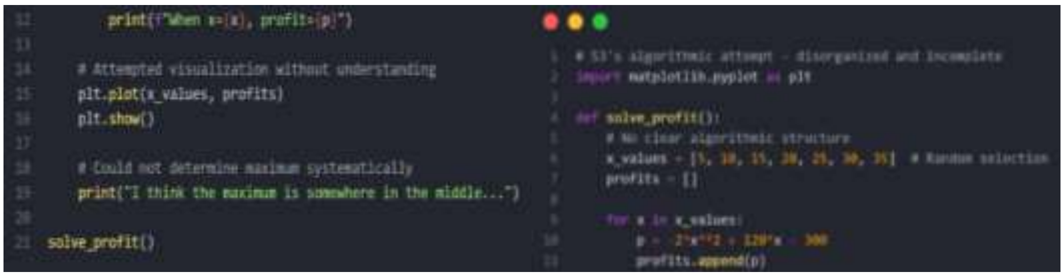
From the interview results, subject S3 explained their abstraction process:

> "*I thought I needed to include information about the company and the products because it was mentioned in the problem. I spent time trying to make the program look realistic with company details. I wasn't sure which parts of the problem were important for the math and which parts were just story context.*"

This difficulty with abstraction reveals that S3 cannot effectively filter information to focus on the essential mathematical relationships. The student becomes overwhelmed by contextual details rather than extracting the core computational problem that needs to be solved.

Algorithm Capability

Subject S3 showed the most significant struggles with algorithmic thinking, demonstrating an inability to develop systematic, sequential steps for problem-solving. The student's attempt at creating a solution algorithm is shown in Figure 14.



```python
# S3's algorithmic attempt - disorganized and incomplete
import matplotlib.pyplot as plt

def solve_profit():
    # No clear algorithmic structure
    x_values = [5, 10, 15, 20, 25, 30, 35]  # Random selection
    profits = []

    for x in x_values:
        p = -2*x**2 + 120*x - 300
        profits.append(p)
        print(f"When x={x}, profit={p}")

    # Attempted visualization without understanding
    plt.plot(x_values, profits)
    plt.show()

    # Could not determine maximum systematically
    print("I think the maximum is somewhere in the middle...")

solve_profit()
```

Figure 14. Algorithm Development Struggle by Subject S3
Source: VSCode Text Editor

In the interview, subject S3 explained their algorithmic challenges:

> "*I knew I needed to make steps to solve the problem, but I couldn't figure out the right order. I tried to copy what I saw in examples, like making a graph, but I didn't understand how to make the computer find the exact answer. I could see the highest point on the graph, but I couldn't make the program tell me the exact numbers.*"

This algorithmic struggle demonstrates that S3 lacks the ability to translate mathematical procedures into computational steps. The student cannot bridge the gap between understanding that an optimization problem exists and implementing a systematic approach to solve it.

Key Areas Where S3 Struggled

The analysis reveals that S3's difficulties stemmed from several interconnected issues: (1) *Mathematical-Programming Translation Gap*: S3 could not effectively translate mathematical concepts (like finding derivatives or using vertex formulas) into programming logic. This suggests a need for more scaffolded instruction that explicitly connects mathematical procedures to coding implementations; (2) *Procedural vs. Conceptual Understanding*: S3 appeared to rely heavily on memorized procedures without deep conceptual understanding. When faced with variations in the problem, the student could not adapt because they lacked understanding of underlying principles; (3) *Cognitive Load Management*: S3 became overwhelmed when trying to handle multiple aspects of the problem simultaneously (mathematical concepts, Python syntax, problem context), suggesting a need for more structured, step-by-step instruction; (4) *Debugging and Iteration Skills*: Unlike S1 and S2 who could refine their approaches when initial attempts didn't work, S3 lacked the metacognitive skills to evaluate and improve their solutions systematically.

These findings have important implications for educators working with students at similar levels, highlighting the need for differentiated instruction that provides additional support in connecting mathematical concepts to computational implementation.

Discussion

Based on the comprehensive research findings examining all three students (S1, S2, and S3), there are significant variations in computational thinking (CT) abilities among tenth-grade Software Engineering students at vocational high schools when solving quadratic function problems using Python. The following discussion compares and contrasts these findings while connecting them to relevant theories and previous research.

Comparative Analysis of Computational Thinking Components

Decomposition Skills Across Proficiency Levels

The analysis reveals a clear progression in decomposition abilities across the three proficiency levels. Subject S1 demonstrated systematic and comprehensive problem breakdown, organizing the quadratic function optimization into distinct, logical steps: function definition, derivative calculation, optimization point finding, and result verification. This sophisticated approach aligns with (Wing's, 2006) assertion that strong decomposition abilities enable more effective complex problem-solving by reducing cognitive load through structured problem partitioning.

Subject S2 showed intermediate decomposition skills, successfully identifying major problem components but with less systematic organization than S1. While S2 could recognize the need to separate function definition from optimization calculations, their approach lacked the comprehensive structure that made S1's solution more robust and reusable.

In stark contrast, Subject S3 demonstrated significant decomposition difficulties, attempting to solve the entire problem as a monolithic unit without recognizing the need for systematic breakdown. This finding is particularly concerning as it suggests S3 lacks the fundamental CT skill that underlies all other computational thinking components. S3's struggle with decomposition appears to stem from an inability to recognize that complex problems require systematic partitioning, which (Syari et al., 2024) identifies as a critical gap in many Indonesian students' computational thinking abilities.

The progression from S3's holistic but ineffective approach to S1's systematic decomposition illustrates how decomposition skills develop from novice pattern recognition to expert systematic analysis. This gradient suggests that decomposition skills can be scaffolded through explicit instruction in problem-breaking strategies.

Pattern Recognition: From Recognition to Strategic Application

Pattern recognition abilities showed equally dramatic variation across subjects, revealing different levels of mathematical and computational pattern awareness. Subject S1 demonstrated advanced pattern recognition by not only identifying mathematical relationships (how coefficient changes affect optimization) but also recognizing programming patterns that could be generalized and reused. S1's ability to abstract the optimization pattern into a reusable function demonstrates what (Chan et al., 2021) describe as recursive thinking capability.

Subject S2 exhibited good pattern identification but struggled with optimal pattern utilization. While S2 could recognize that coefficient changes would affect the optimal point, they could not fully leverage this recognition to create more efficient or generalizable solutions. This suggests an intermediate stage where students can perceive patterns but lack the strategic thinking to fully exploit them.

Subject S3's pattern recognition difficulties were profound, showing minimal ability to identify even basic mathematical relationships between coefficient changes and function behavior. This finding aligns with (Maifi et al., 2021) observation that Indonesian students' pattern recognition skills need significant improvement. S3's struggles suggest that pattern recognition may require explicit instruction in both mathematical relationship identification and computational pattern awareness.

The comparison reveals that pattern recognition in computational contexts requires both mathematical understanding and programming fluency, creating a compound learning challenge that may explain why this skill varies so dramatically among students.

Abstraction: Information Filtering and Focus Management

Abstraction abilities demonstrated perhaps the most educationally significant variations among the three subjects. Subject S1 exhibited sophisticated abstraction skills, effectively filtering relevant mathematical information while ignoring contextual details that didn't contribute to the computational solution. This selective attention aligns with (Lester & Cai's, 2016) emphasis on metacognitive awareness in mathematical problem-solving.

Subject S2 showed intermediate abstraction abilities, generally identifying relevant information but occasionally including unnecessary computational details. This suggests developing but not fully mature abstraction skills, where students understand the need to focus on relevant information but struggle with consistently applying this principle.

Subject S3's abstraction difficulties were particularly revealing for educators. S3 became overwhelmed by contextual problem details (company names, product types, currency) rather than extracting the essential mathematical relationships. This suggests that S3 lacks the metacognitive awareness to distinguish between story context and computational requirements, a skill that (Killpatrick et al., 2010) identify as crucial for mathematical proficiency.

The abstraction skill progression illuminates how students develop from being overwhelmed by surface details to focusing on underlying mathematical structures, suggesting that abstraction instruction should explicitly address information filtering strategies.

Algorithmic Thinking: From Trial-and-Error to Systematic Solutions

Algorithmic thinking capabilities showed the most dramatic differences across subjects, revealing fundamentally different approaches to systematic problem-solving. Subject S1 demonstrated sophisticated algorithmic development, creating well-structured, sequential, and efficient solution pathways that could be easily modified and reused. This systematic approach reflects what describe as mathematical modeling capability.

Subject S2 exhibited developing algorithmic skills, creating sequential solution steps but with less comprehensive planning and efficiency than S1. S2's algorithms worked but lacked the elegance and reusability that characterized S1's approach, suggesting intermediate systematic thinking skills.

Subject S3's algorithmic difficulties were most pronounced, showing an inability to move beyond trial-and-error approaches to systematic problem-solving strategies. S3's reliance on random value testing rather than mathematical optimization procedures suggests fundamental gaps in understanding how to translate mathematical procedures into computational algorithms.

This algorithmic skill progression reveals how students develop from unsystematic problem-solving attempts to sophisticated computational thinking, highlighting the importance of explicit algorithm development instruction.

Cross-Component Interactions and Dependencies

The comparative analysis reveals important interactions between CT components that have significant educational implications. Students with stronger mathematical foundations (S1) demonstrated superior performance across all CT components, while students with weaker mathematical understanding (S3) struggled with multiple components simultaneously. This suggests that CT development may be constrained by mathematical conceptual understanding, supporting (Schoenfeld, 2016) assertion that foundational knowledge affects problem-solving capabilities.

Furthermore, the analysis reveals that CT components are not independent but form an interconnected skill system. Students who struggle with decomposition (S3) also have difficulty with pattern recognition and algorithmic development, suggesting that these skills may need to be developed together rather than in isolation.

Educational Implications from Comparative Analysis

The three-student comparison provides crucial insights for mathematics education practice. The dramatic differences in CT abilities suggest that one-size-fits-all approaches to computational thinking instruction may be ineffective. Instead, the findings support differentiated instruction approaches that provide varying levels of scaffolding based on student proficiency.

For students at S3's level, the analysis suggests that CT instruction should begin with explicit decomposition training, focusing on problem-breaking strategies before progressing to more advanced skills. The findings also indicate that students like S3 need more structured connections between mathematical concepts and programming implementation, supporting (Gadanidis et al., 2017) recommendation for explicit mathematical-computational bridging instruction.

For students at S2's level, instruction should focus on optimization and strategic thinking, helping them leverage their pattern recognition abilities more effectively and develop more systematic algorithmic approaches.

For advanced students like S1, instruction can focus on generalization and abstraction refinement, encouraging them to develop increasingly sophisticated and reusable computational solutions.

Implications for Curriculum Development

This comparative analysis has significant implications for curriculum development in mathematics education, particularly for programs integrating computational thinking. The findings suggest that curriculum designers should consider creating multiple pathways or tracks that accommodate different CT development levels rather than assuming uniform student capabilities.

The research also supports the integration of programming into mathematics education but suggests that this integration requires careful scaffolding and explicit instruction in mathematical-computational connections. Simply providing programming tools without systematic CT skill development may not be sufficient for students at lower proficiency levels.

Finally, the findings suggest that assessment strategies should evaluate CT components both individually and in integration, recognizing that these skills develop as interconnected systems rather than isolated capabilities.

Research Limitations

This study has several limitations that should be considered when interpreting the results and for future research consideration. First, this research involved only three research subjects, so generalization of research results should be done cautiously. For future research, the number of research subjects could be increased to obtain a more comprehensive picture of students' computational thinking abilities.

Second, this research is limited to the context of quadratic function material. Students' computational thinking abilities may differ when facing different mathematical material.

For future research, exploration of students' computational thinking abilities in more diverse mathematical contexts could be conducted.

Third, this research focuses only on tenth-grade vocational high school students in Software Engineering who already have basic knowledge of Python programming. Results might differ if applied to students from different majors or educational levels. Future research could involve students from various educational backgrounds to gain broader understanding of how educational background influences computational thinking abilities.

Fourth, this research uses a qualitative approach with a case study method, so it cannot examine causal relationships between Python use and computational thinking development. Future research could use experimental approaches to test the effectiveness of Python use in developing students' computational thinking abilities.

CONCLUSION AND SUGGESTIONS

Based on the research results and discussion, The authors conclude that the computational thinking abilities of tenth-grade vocational high school students in Software Engineering when solving quadratic function problems using Python vary significantly, categorized as high, medium, and low. Students with high ability (S1) demonstrated excellent decomposition, pattern recognition, abstraction, and algorithmic skills, characterized by systematic and comprehensive problem decomposition, accurate pattern identification with optimal utilization, and efficient structured algorithm development. Meanwhile, students with medium ability (S2) showed good capabilities but were less optimal in several aspects, such as less systematic decomposition, suboptimal pattern utilization, inclusion of unnecessary details in abstraction, and less comprehensive algorithm development. Students with low ability (S3) demonstrated difficulties in several computational thinking aspects. This research also found that using Python in mathematics education, particularly in quadratic function material, can help students develop their computational thinking abilities through abstract concept visualization and mathematical concept application in more authentic contexts. This indicates that integrating programming into mathematics education has potential to enhance students' computational thinking skills, which are essential in today's digital era.

Based on the research results, the authors recommend that educators integrate programming, particularly Python, into mathematics education to develop students' computational thinking skills. Teachers should provide adequate scaffolding for students with different abilities and design learning activities that encourage development across all computational thinking aspects. Schools should facilitate curriculum development that integrates computational thinking into mathematics education, including technological infrastructure provision and teacher training. For future research, the authors recommend involving more research subjects, exploring more diverse mathematical topics, and using different methodological approaches such as experimental research to test the effectiveness of various approaches in developing students' computational thinking skills. Curriculum developers should consider explicitly integrating computational thinking into the mathematics curriculum, including providing guidelines and resources that support its implementation. For students, the authors recommend utilizing technology, particularly Python programming, as a tool to aid mathematical concept understanding and problem-solving skill development, ultimately increasing their competitiveness in the increasingly developing digital era.

Acknowledgement
The authors thank all parties who assisted in this research. The authors express our gratitude to the principal, teachers, and students of the vocational high school who were willing to be the location and subjects of the research. Acknowledgments are also conveyed to the supervising professors who provided direction and input during the research process, as well as to colleagues who provided support and assistance. Not forgotten, the authors also thank their families for the support and prayers given during the implementation of the research.

REFERENCES

Angeli, C., & Giannakos, M. (2020). Computational thinking education: Issues and challenges. *Computers in Human Behavior*, *105*(January). https://doi.org/10.1016/j.chb.2019.106185

Azmi, N., & Yunita, R. (2022). Menyelesaikan Masalah Fungsi Kuadrat Di, *3*(1), 41–49.

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

Cahdriyana, R. A., & Richardo, R. (2020). Berpikir Komputasi Dalam Pembelajaran Matematika. *LITERASI (Jurnal Ilmu Pendidikan)*, *11*(1), 50. https://doi.org/10.21927/literasi.2020.11(1).50-56

Dull, E., & Reinhardt, S. P. (2014). An analytic approach for discovery. *CEUR Workshop Proceedings*.

Fadillah, A. (2019). Analisis Kemampuan Penalaran Deduktif Matematis Siswa. *JTAM | Jurnal Teori Dan Aplikasi Matematika*, *3*(1), 15. https://doi.org/10.31764/jtam.v3i1.752

Gadanidis, G., Hughes, J. M., Minniti, L., & White, B. J. G. (2017). Computational Thinking, Grade 1 Students and the Binomial Theorem. *Digital Experiences in Mathematics Education*, *3*(2), 77–96. https://doi.org/10.1007/s40751-016-0019-3

Harangus, K., & Kátai, Z. (2020). Computational thinking in secondary and higher education. *Procedia Manufacturing*, *46*(2019), 615–622. https://doi.org/10.1016/j.promfg.2020.03.088

Irawan, E., Rosjanuardi, R., & Prabawanto, S. (2024). Promoting Computational Thinking through Programming Trends, Tools, and Educational Approaches: a Systematic Review. *JTAM (Jurnal Teori Dan Aplikasi Matematika)*, *8*(4), 1327. https://doi.org/10.31764/jtam.v8i4.26407

Kamil, R., Imami, A. I., & Abadi, A. P. (2021). Analisis kemampuan berpikir komputasional matematis Siswa Kelas IX SMP Negeri 1 Cikampek pada materi pola bilangan Abstrak A . Pendahuluan Memasuki abad ke-21 yang disebut dengan abad digital , dimana perkembangan teknologi semakin maju dan berkembang san. *AKSIOMA: Jurnal Program Studi Pendidikan Matematika*, *12*(2), 259–270.

Killpatrick, J., Swafford, J., & Findell, B. (2010). *it UP ! October*.

Mackiewicz, J. (2018). *Writing center talk over time: A mixed-method study*. *Writing Center Talk over Time: A Mixed-Method Study*. https://doi.org/10.4324/9780429469237

Maifi, Y. K., Anwar, & Ahmad, A. (2021). Students' understanding of mathematical concepts and their self-confidence through a discovery learning model. *Journal of Physics: Conference Series*, *1882*(1). https://doi.org/10.1088/1742-6596/1882/1/012081

Marcelino, M. J., Pessoa, T., Vieira, C., Salvador, T., & Mendes, A. J. (2018). Learning Computational Thinking and Stratch at Distance. Computers on Human Behavior, *80*(80), 470–477.

Minarni, A. (2021). Pengaruh Pembelajaran Berbasis Masalah Dan Keterampilan Sosial Siswa Smp Negeri Di. *Jurnal Pendidikan Matematika PARADIKMA*, *6*(2), 162–174.

Mubarokah, H. R., Pambudi, D. S., Lestari, N. D. S., Kurniati, D., & Jatmiko, D. D. H. (2023). Kemampuan Berpikir Komputasi Siswa dalam Menyelesaikan Soal Numerasi Tipe AKM Materi Pola Bilangan. *JNPM (Jurnal Nasional Pendidikan Matematika)*, *7*(2), 343. https://doi.org/10.33603/jnpm.v7i2.8013

Nau, S., & Sulistyani, N. (2023). ISSN : 3047-2059 Pengembangan Modul Pembelajaran Interaktif Berbasis Computational Thinking Menggunakan Canva ISSN : 3047-2059. *Semnaptika2023*, 66.

polya. (1957). George_Polya_How_To_Solve_It_.pdf.

Schoenfeld, A. H. (2016). Learning to Think Mathematically: Problem Solving, Metacognition, and Sense Making in Mathematics (Reprint). *Journal of Education*, *196*(2), 1–38. https://doi.org/10.1177/002205741619600202

Simanjuntak, E., Armanto, D., & Dewi, I. (2023). Analisis Kemampuan Berpikir Komputasional Matematis Siswa Dalam Menyelesaikan Soal Pisa Konten Change And Relationship. *Jurnal Fibonaci: Jurnal Pendidikan Matematika*, *4*(1), 11. https://doi.org/10.24114/jfi.v4i1.46106

Syari, A. K., Fatra, M., & Diwidian, F. (2024). *Analisis Kemampuan Berpikir Komputasional Matematis Siswa Dalam Menyelesaikan Masalah Kontekstual Ditinjau Dari Kemandirian Belajar*. *ALGORITMA: Journal of Mathematics Education* (Vol. 6). https://doi.org/10.15408/ajme.v6i1.38380

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

Yadav, A., Gretter, S., Hambrusch, S., & Sands, P. (2017). Expanding computer science education in schools: understanding teacher experiences and challenges. *Computer Science Education*, *26*(4), 235–254. https://doi.org/10.1080/08993408.2016.1257418