African Journal of Research in Mathematics, Science and Technology Education, 2025
Vol. 29, No. 3, 322–334, https://doi.org/10.1080/18117295.2025.2533736

Teaching Programming in Township Schools: A Mental Model Approach to Developing Computational Thinking

Mashite Tshidi *

Department of Science, Mathematics and Technology Education, University of Pretoria, Pretoria, South Africa
**Email: mashite.tshidi@up.ac.za*

Computational thinking is essential for problem-solving in the digital age, yet disparities in resources, infrastructure and teacher training create unequal learning experiences in programming education. These challenges are especially evident in township schools, where teachers must foster computational thinking with limited access to digital tools and resources. Addressing misconceptions and cognitive overload is central to this process, as flawed mental models hinder learners' ability to grasp programming concepts. Given the high failure and dropout rates in introductory programming, there is a pressing need to understand how teachers adapt their instructional practices in these contexts. Grounded in mental model theory, this study investigates how teachers impact learners' conceptual understanding of programming, despite pedagogical and infrastructural constraints. Using a qualitative case study design, semi-structured interviews were conducted with township school teachers, and thematic analysis identified four key instructional strategies. Teachers employ stepwise algorithmic thinking to support problem decomposition, use real-world examples to bridge abstract concepts, incorporate peer-assisted learning to accommodate varied learning paces and engage in practical activities to address prior knowledge gaps. These findings underscore the role of teachers in overcoming learning barriers and promoting computational thinking in township contexts.

Keywords: *Computational thinking; instructional practices; mental models; programming education; teacher pedagogy; township schools*

Introduction

In today's rapidly evolving digital world, programming skills are essential for fostering critical thinking and problem-solving abilities across disciplines and preparing individuals for careers in science and technology (Belmar, 2022). Many countries have integrated programming education into school curricula, recognising computational thinking (CT) as a core twenty-first-century competency (Ezeamuzie & Leung, 2022). Computational thinking involves structuring tasks logically through abstraction, decomposition, debugging and algorithmic design, skills that extend beyond computing and apply to various fields (Belmar, 2022; Ogegbo & Ramnarain, 2022).

In South Africa, efforts to integrate CT into basic education have led to the inclusion of Coding and Robotics as part of the Department of Basic Education's digital skills strategy (Tshidi & Dewa, 2024). Although Information Technology (IT) and Computer Applications Technology are formal subjects in the senior grades, stark inequalities persist in how programming education is delivered across the country. Affluent schools often have well-maintained infrastructure, reliable internet and trained staff, which places them in a better position to implement these subjects effectively. In contrast,

many township schools, though located in urban or peri-urban areas, face systemic challenges such as inadequate infrastructure, unreliable internet access and limited exposure to digital tools (Mkhize & Davids, 2021). These disparities widen the digital divide and hinder learners' development of essential programming and problem-solving skills (Aruleba & Jere, 2022).

While there have been initiatives to address these gaps, including government-funded ICT rollouts and partnerships with private organisations, their impact has been uneven and often lacks long-term sustainability (Cheruiyot & Chepngetich, 2023). A critical aspect of programming education is the development of accurate mental models—cognitive structures that help learners understand programming concepts, anticipate code behaviour and solve computational problems (Heinonen et al., 2023). Although challenges in developing these models are observed across various contexts, they are often intensified in township schools, where learners typically enter with minimal prior exposure to computing (Mkhize & Davids, 2021).

These systemic issues are compounded by limited access to digital devices, poorly resourced learning environments and a lack of structured instructional scaffolding (Mkhonto & Mubangizi, 2024). These context-specific challenges restrict learners' opportunities to internalise CT skills and contribute to persistent inequalities in programming education. Teachers play a pivotal role in shaping learners' mental models and correcting misconceptions that hinder programming learning. While this challenge is reported globally in under-resourced contexts (Rich et al., 2021), it is especially prominent in township schools, where many teachers feel unprepared to teach programming owing to limited professional development and outdated instructional resources.

Several teacher training initiatives, including those led by Code College, CodeX and CodeSpace, aim to improve programming instruction in South Africa. These efforts are recognised in broader African research (Cheruiyot & Chepngetich, 2023), highlighting South Africa's leading role in piloting school coding. However, systemic challenges persist in undermining the effectiveness of such initiatives. Notably, there is limited empirical research on how teachers' instructional practices influence learners' development of mental models in programming, particularly in resource-constrained environments (Heinonen et al., 2023).

This study investigates the impact of teachers' instructional practices on the development of mental models in the introductory programming subject in township schools, where resource disparities affect learning experiences. Employing a qualitative research approach, the study provides insights into instructional strategies to enhance programming education in township school contexts. The following research questions guide the study:

1. How do teachers in township schools use instructional practices to support learners' development of mental models in computational thinking for introductory programming?
2. What resource constraints influence these instructional practices, and how do they shape learners' development of mental models in computational thinking?

Literature Review

Teachers' Role in Developing Computational Thinking

In the twenty-first century, CT has emerged as a foundational skill underpinning modern technological advancements, including artificial intelligence, data science and robotics. The increasing reliance on digital technologies has prompted educators to foster CT through programming curricula to equip learners with problem-solving abilities, algorithmic reasoning and logical thinking (Belmar, 2022). Although CT is embedded within programming, its development in learners requires intentional pedagogical strategies.

Research highlights the benefits of CT, including improved critical thinking, teamwork, communication and self-management (Ezeamuzie & Leung, 2022). However, learners' acquisition of these competencies depends significantly on teachers' instructional approaches and level of preparedness (Ogegbo & Ramnarain, 2022). The shift from viewing programming as a specialised skill to treating it as a core educational component underscores the importance of teacher facilitation in promoting CT within diverse learning environments (Belmar, 2022).

Studies have identified core CT practices relevant to programming education, such as decomposition, pattern recognition, abstraction and algorithmic design (Belmar, 2022; Ezeamuzie & Leung, 2022). Their development relies on purposeful instructional design and teacher facilitation. Research-based principles for promoting CT emphasise the value of structured yet flexible environments, where teachers scaffold understanding and guide learners through problem-solving (Mills et al., 2024).

One important concept is distributed expertise, which involves shared cognitive processes and problem-solving responsibilities distributed across teachers, learners and curricular or digital resources (Qian & Lehman, 2017). Decomposition, in particular, is effectively developed in classrooms where teachers foster collaborative environments with peer interaction, structured feedback and clarification of conceptual misunderstandings (Mills et al., 2024). In such settings, teachers act as facilitators and guides, promoting learner autonomy and independence. This aligns with Yang et al. (2025), who emphasise teacher scaffolding in early programming instruction.

Explicit instruction, which refers to direct and structured teaching where learning goals, strategies and outcomes are modelled, is also crucial for developing CT understanding (Yang et al., 2025). Teachers model algorithmic processes, support debugging and teach decomposition strategies to learners. When used in contexts such as robotics and programming simulations, structured sequences promote conceptual understanding while allowing independent exploration (Pellas, 2023; Yang et al., 2025).

However, a balance must be struck between problem-based learning (PBL) and structured guidance. While PBL fosters creativity and exploration, insufficient scaffolding may lead to inefficient strategies. Conversely, overly rigid instruction may limit learners' creative thinking (Waite & Sentance, 2021). Teachers are thus encouraged to adopt adaptive instructional approaches that are responsive to learners' prior experiences and cognitive development. Such strategies support equitable CT development, particularly in contexts with limited instructional resources (Waite & Sentance, 2021; Mills et al., 2024).

The Role of Mental Models in Programming Instruction

Teaching programming requires understanding how learners develop mental models of programming concepts. These models serve as cognitive frameworks that help learners interpret source code, recognise programming patterns and predict execution outcomes (Heinonen et al., 2023). Ortiz et al. (2023) note that mental models are central to CT, as they link conceptual understanding with practical application. However, because model development is fluid and abstract, it is often difficult for teachers to identify misconceptions and refine learners' understanding (Qian & Lehman, 2017). Inconsistent development can lead to persistent misconceptions that hinder programming proficiency (Julie & Bruno, 2020).

Misconceptions in programming tend to fall into three categories: syntactic, conceptual and strategic (Qian & Lehman, 2017). Syntactic misconceptions stem from difficulties with programming language syntax, such as misplaced semicolons in Java or incorrect string delimiters. Conceptual misconceptions occur when learners misinterpret the function of programming constructs, including variable assignment, loop iteration or control structures (Waite & Sentance, 2021). Strategic misconceptions arise when learners struggle to apply their knowledge in problem-solving, often resorting to inefficient debugging or excessive trial and error (Qian & Lehman, 2017). Addressing these issues requires targeted interventions that reinforce accurate mental models and strengthen CT (Cheah, 2020).

Several instructional approaches have been explored to support the development of models. Henley et al. (2021) introduced a Python-based code editor with real-time feedback to correct misconceptions as they emerge. These interventions combine guided instruction with learner autonomy, promoting CT through structured support (Mills et al., 2024). Johnson et al. (2022) also found that scaffolding strategies, including algorithm animations and memory visualisations, improve conceptual understanding and facilitate knowledge transfer across programming paradigms. The success of such approaches depends largely on teachers' pedagogical expertise and the quality of instructional design.

Teacher Preparedness and Instructional Challenges in Computational Thinking

Although CT is an essential aspect of programming education, its implementation depends on teachers' preparation and instructional methodologies (Ezeamuzie & Leung, 2022). Many teachers lack formal training in programming pedagogy, specialised approaches for teaching programming and CT-focused instructional practices, which limits their capacity to deliver meaningful learning experiences (Ezeamuzie & Leung, 2022). In South Africa, many IT teachers have limited programming experience owing to gaps in their tertiary education or the absence of specialised training (Tshidi & Dewa, 2024).

These challenges are especially pronounced in township and under-resourced schools, where learners' engagement with programming concepts is often poor, reducing the effectiveness of instructional activities. Teachers' confidence and attitudes toward educational technology influence their willingness to incorporate CT into instruction, with lower competence often linked to reduced motivation (Tshidi & Dewa, 2024).

The lack of structured, ongoing professional development exacerbates these challenges. Many teachers are unfamiliar with the educational value of programming concepts such as abstraction and logical reasoning, or lack the skills to teach them effectively (Belmar, 2022). This often stems from limited access to professional training and weak collaboration between schools and higher education institutions, which in turn undermines the long-term impact (Ni et al., 2023).

Several initiatives, including those by Code College, CodeX and CodeSpace, have aimed to improve teacher preparedness by focusing on CT pedagogy and adaptive teaching methods (Cheruiyot & Chepngetich, 2023). However, without sustained support, these programmes often fall short of their goals. This reflects a broader neglect of discipline-specific pedagogical content knowledge in teacher development efforts, leaving many teachers unprepared for the demands of programming instruction (Ni et al., 2023).

Gaps in the Literature and Research Contributions

Despite progress in integrating CT into programming education, several gaps remain. While prior research has focused mainly on learner-centred interventions, limited attention has been paid to how teachers actively promote CT through their instructional strategies (Belmar, 2022; Ezeamuzie & Leung, 2022). In particular, few empirical studies examine how teachers conceptualise, implement and adapt CT in township programming classrooms. Although professional development programmes for CT instruction exist, minimal research has explored how these initiatives build sustained instructional capacity and discipline-specific pedagogical content knowledge (Cheruiyot & Chepngetich, 2023; Ni et al., 2023). The long-term influence of such training on teacher readiness and classroom practice remains under-researched. Addressing these gaps is essential for informing targeted interventions that strengthen teachers' capacity to facilitate CT instruction. This study provides empirical insights into how teachers support mental model development through CT, respond to learner misconceptions and adopt instructional practices to enhance programming outcomes. Thus, the analytical focus shifts from learner activity to teachers' instructional practices and decision-making.

Conceptual Framework

This research employs mental model theory (MMT) as a conceptual lens to examine how teachers' instructional practices influence learners' development of mental models in programming. Mental model theory posits that individuals reason and solve problems using mental models, internal representations based on their comprehension of premises and general knowledge, rather than formal inference rules (Johnson-Laird, 2001). These models serve a dual function: abstract representations of reasoning outcomes and cognitive instruments for manipulating mental objects during problem-solving (Johnson-Laird, 2001). Three foundational principles guide reasoning processes in MMT: (1) individuals construct mental models that represent distinct possibilities consistent with a given set of premises; (2) these models reflect what is true in the given situation and often do not explicitly

represent what is false; and (3) deductive reasoning involves forming, inspecting and revising these models to conclude (Johnson-Laird, 2001).

Mental Models as Representations of Possibility

Each mental model represents a distinct possibility based on given premises, allowing individuals to simulate different outcomes. These models are iconic, meaning their structure mirrors the relationships within a concept (Johnson-Laird, 2001). For example, in the context of teaching CT, consider a programming scenario where a teacher explains the concept of conditional statements using an if–else construct:

> *If it rains, take an umbrella; otherwise, leave it at home.*

This example allows two distinct possibilities that learners must mentally model:

1. 'It is raining, so you take an umbrella'.
2. 'It is not raining, and you do not have an umbrella'.

These possibilities can be represented as mental models:

- Raining → Take an umbrella
- ¬Raining → ¬Take an umbrella

Here, the ¬ denotes negation, indicating the absence of rain and the action of not taking an umbrella. Mental models help learners simplify conditional logic by clarifying the relationship between the condition (if) and the outcome (then). The iconic nature of these models ensures that their structure mirrors the syntax and logic of conditional statements, allowing learners to visualise and internalise the scenarios. In developing CT skills, especially logic and abstraction, constructing and manipulating accurate mental models is foundational for understanding the structures that underpin programming (Heinonen et al., 2023). This instructional process involves providing clear examples, scaffolding reasoning and linking abstract concepts to relatable scenarios. Through these strategies, teachers bridge theory and practice, fostering the development of CT competencies (Ezeamuzie & Leung, 2022).

The Principle of Truth

The principle of truth in MMT asserts that mental models represent only what is explicitly true according to the given premises, implicitly omitting false possibilities. This streamlined representation minimises unnecessary cognitive effort by focusing on valid reasoning components, promoting mental clarity and efficiency (Johnson-Laird, 2001). In the context of CT, particularly in logical reasoning, this principle helps learners understand conditional and iterative programming structures, which often involve binary states such as true/false or on/off (Fagerlund et al., 2021). For instance, when learning about if–else statements, learners typically construct models around the proper conditions to predict a program's outcome. However, this selective representation is not arbitrary; it operates at two interconnected levels. At a broader level, mental models represent only possibilities consistent with the premises. At a finer level, they only include specific components of a premise when those components hold true in a given scenario (Johnson-Laird, 2001).

While this truth-based modelling approach supports cognitive efficiency, it may also initially lead learners to overlook false possibilities. Teachers can address this by designing tasks that prompt learners to build 'mental footnotes' (Johnson-Laird, 2001: 435), annotations within their mental models that track omitted possibilities. This strategy enables learners to expand their mental models to account for true and false conditions, refining their logical reasoning in programming contexts.

Deductive Reasoning Hinges on Mental Models

The third foundational principle of MMT emphasises that deductive reasoning relies on mental models to evaluate the validity, probability and plausibility of conclusions drawn from premises. In CT, this is critical for testing hypotheses, debugging programs and verifying code correctness. A conclusion is

logically sound if it holds true across all mental models generated from the premises; otherwise, its probability depends on the proportion of models that support it (Johnson-Laird, 2001).

Teachers can enhance learners' logical reasoning by guiding them in constructing and manipulating mental models that reflect necessity, probability and possibility. This process enables learners to debug programs systematically, assess code validity across scenarios and recognise the broader implications of their programming decisions (Kadar et al., 2021). Through structured problem-solving and targeted scaffolding, teachers help learners refine their reasoning by ensuring that their conclusions hold across all relevant models.

Mental Model Theory in Township Schools

For CT to be integrated into programming education, teachers must understand the cognitive processes that underpin CT and the strategies needed to foster these skills in learners (Qian & Lehman, 2017). Educators' perceptions and instructional approaches influence how CT is incorporated into teaching practices (Ogegbo & Ramnarain, 2022). In township schools, where learners may face challenges with abstract reasoning partly owing to limited prior engagement with digital tools and representations, how teachers conceptualise and deliver programming content significantly impacts learning outcomes (Mkhonto & Mubangizi, 2024). As a result, the development of learners' mental models in township schools cannot be explained solely through cognition but is also shaped by contextual and systemic influences.

Teachers' ability to support CT development is further mediated by pedagogical and institutional constraints, including limited professional development in programming pedagogy, weak collaboration between schools and higher education institutions, and varying levels of teacher confidence in using educational technology (Tshidi & Dewa, 2024; Ni et al., 2023). Guided by MMT, this study examines how teachers' instructional practices influence the development of mental models for CT in introductory programming. Adopting a qualitative approach, the study investigates how teachers translate CT into practice and adapt their strategies to support learners' reasoning and conceptual understanding in township environments.

Methodology

This study employs a qualitative approach to investigate how teachers' instructional practices influence the development of learners' mental models that facilitate CT in introductory programming. Rooted in the interpretive paradigm, this approach seeks to understand how teachers utilise instructional strategies to construct meaning in classroom settings, grounded in the premise that experience and context shape knowledge—a foundational idea in interpretivist research traditions (Creswell & Poth, 2018).

With its focus on subjective experiences and the varied ways instructional practices shape cognitive development, interpretivism provides a lens to capture the intricacies of teaching programming. As programming involves symbolic logic, unseen processes and rule-based systems, learners must engage in structured reasoning that draws on logical and sequenced thought (Fagerlund et al., 2021; Qian & Lehman, 2017). This study examines how teachers' practices support learners in developing internal representations of programming concepts, aligned with the principles of MMT (Johnson-Laird, 2001).

Research Design

This study employed a multiple-case study design to explore how teachers' instructional practices influence the development of learners' mental models that support CT in introductory programming. Each case constituted a bounded system, defined by an individual teacher in a specific township school in the Ekurhuleni South district, responsible for delivering programming instruction within the CAPS curriculum. The boundedness of each case lay in the teacher's role, school context and instructional decision-making within township conditions. This design enabled a context-sensitive

investigation into how teachers conceptualise CT, design instruction and adapt strategies to support conceptual understanding.

MMT posits that individuals construct internal cognitive representations based on their understanding of premises, rather than relying solely on formal logical rules. These mental models serve as cognitive tools that facilitate problem-solving and provide abstract representations resembling possible outcomes (Johnson-Laird, 2001). This study, therefore, examines how teachers in township schools use instructional practices to scaffold programming concepts in ways that support learners' construction of accurate and functional mental models for CT.

Participation Selection

A purposeful sampling approach was employed to select four IT teachers who teach introductory programming to learners in Grades 10–12, ensuring rich and contextually relevant insights. The sample size reflects the limited number of schools in the study's region offering IT as a subject. This is consistent with the case-oriented nature of qualitative research, where depth of insight is prioritised over breadth of coverage (Creswell & Poth, 2018). Participants were selected based on the subsequent criteria, which are presented in Table 1.

Although the teachers satisfied these criteria, variations in their pedagogical approaches, instructional practices and conceptual scaffolding procedures presented distinct perspectives on how learners internalise programming logic. Participants were assigned pseudonyms to maintain confidentiality.

Data Collection

This study employed semi-structured interviews to probe teachers' pedagogical choices, conceptual scaffolding strategies and the challenges of guiding mental model construction, capturing the nuances of instructional decision-making. Since mental models function as representations of reasoning outcomes and tools for manipulating problem-solving elements (Heinonen et al., 2023), the interviews explored how teachers support learners in constructing, refining and applying these models in CT tasks, such as algorithm development, debugging and program design. Interview questions focused on how teachers introduce and explain programming concepts, scaffold problem-solving, address common misconceptions and use instructional strategies to strengthen learners' reasoning structures. Each interview lasted between 30 and 45 minutes and was audio-recorded and transcribed verbatim to ensure a complete account of the teachers' narratives.

Data Analysis

This study used a reflexive thematic analysis approach, which provided a structured yet flexible framework for capturing how teachers support the formation of mental models for CT. Given that instructional practices are shaped by teachers' experiences, beliefs and contextual constraints, reflexive thematic analysis emphasises the researcher's active role in identifying and interpreting patterns rather than seeking thematic consensus (Braun & Clarke, 2019).

The analysis followed an immersive, iterative process that began with repeated readings of interview transcripts to familiarise oneself with teacher narratives. Initial coding involved systematically

Table 1. Criteria for selecting participants in the study

Criteria	Description
Teaching subject	IT
Teaching qualification and experience	Qualified to teach IT with a minimum of two years' experience teaching Grades 10–12
Province	Gauteng
Area	Ekurhuleni South
Programming languages	Familiarity with Delphi and Java [aligned the CAPS curriculum]

categorising excerpts based on emerging patterns, with attention to how teachers rationalised their pedagogical choices using MMT principles. From these codes, themes were developed to document how instruction was structured to support CT. Themes were refined through ongoing engagement with the data to reflect the diverse nature of instructional strategies rather than imposing rigid classifications (Braun & Clarke, 2019). Final themes were selected to present a coherent account of how teachers promote CT, enabling a context-sensitive understanding of how pedagogical practices shape learners' cognitive representations of programming concepts.

Credibility Measures

The study's credibility was enhanced through member checking, where participants reviewed initial findings and provided feedback on their accuracy. This iterative validation ensured that the findings reflected teachers' instructional realities. In addition, thick description offered context-rich narratives that immersed the reader in the instructional settings, reinforcing both credibility and transferability. Together, these measures strengthen the trustworthiness of the findings and provide a solid foundation for the discussion.

Findings

The findings of the semi-structured interviews suggest that teachers' instructional practices enable learners to develop mental models that facilitate CT in introductory programming. Teachers employ various strategies to help learners develop their CT, including contextualised learning, collaborative engagement, structured problem-solving procedures and adaptive scaffolding. These key themes are presented in the following sections in order of prominence.

Theme 1: Developing Algorithmic Thinking Through Structured Problem-solving

Teachers emphasised algorithmic thinking as a prerequisite for CT. A structured approach to problem-solving was often employed to help learners systematically decompose programming challenges. Teacher A outlined his six-step approach:

> At Grade 10, learners enter the subject without a conceptual foundation of programming. To develop algorithmic thinking, I emphasise the six steps of problem-solving: (1) defining the problem, (2) understanding the problem, (3) applying decomposition, (4) planning the logic, (5) using solution development tools like IPO [input–process–output] models and flowcharts, and (6) creating an algorithm.

Other teachers endorsed this structured approach, highlighting how it facilitates learners' transition from conceptual understanding to practical programming implementation. However, challenges surfaced since learners' problem-solving skills vary, necessitating the integration of additional support mechanisms. Teachers noted that prior computing experience influenced learners' ability to grasp algorithmic thinking, necessitating the need for differentiated instruction.

Theme 2: Contextualising Programming Concepts for Conceptual Understanding

Teachers used everyday scenarios and context-based examples to make programming concepts accessible to learners. Teacher B described how they assist learners in developing mental models by using everyday scenarios:

> I use everyday examples for them, like before you cook, you know that you need to follow the recipe first. To do this, follow the recipe up to the last step. So, I then ask them, 'How do you bake the cakes?' What is it that you need? Then, you ask them to mention all those steps, which will inform the algorithm course. So those are the examples that I can use next.

This reinforces problem decomposition and structured planning by reflecting the logical sequence required in programming. Likewise, Teacher C described how they engage learners through including pertinent socio-economic issues:

> Regarding problem-solving, I structure our questions for that lesson around whatever is topical. For example, during an election season, I have my grade 10 students write a programme to facilitate online voting and record the votes. So, for example, when we were having problems with the licence system with people struggling to log online to book a driver's licence test, we wrote a much smaller program to do that.

These contextualised approaches were reported to support CT by enabling teachers to bridge the gap between abstract programming concepts and practical applications.

Theme 3: Fostering Collaborative Learning to Improve Computational Thinking

Collaboration was one of the teachers' most essential strategies to accommodate different learning paces. Teacher B expressed her approach to promoting peer-assisted learning:

> When teaching a new programming concept, one learner will immediately catch it and understand it. Once he is done working on the programme, he is often willing to assist other learners who are slower. Also, I put them on pair work when they have a challenging programme. However, I get them closer by walking around to see as they develop programmes. I can see how they are either succeeding or not.

Teachers also emphasised the challenge of balancing collaborative tasks and independent problem-solving. While group projects facilitate learning, rigorous facilitation is necessary to ensure equitable engagement and prevent learners from becoming overly reliant on their peers.

Theme 4: Adapting Instruction to Address Prior Knowledge Gaps

Disparities in learners' prior computing experience presented challenges at the beginning of the programming course. To bridge these gaps, teachers outlined how they modified their instruction. Teacher D stated:

> Teaching programming is incredibly challenging. I am concerned that they do not have prior knowledge when they choose the subject in Grade 10. One way to make it enjoyable is through gaming. I sometimes get the feeling that when kids think they can be good gamers, they also think they can be good programmers. I make it extremely exciting for them, especially when they are starting. They work with the properties and components of Delphi. First, I ask them to insert a form, change the colour, height and width, and make those visual changes. You can see that they get excited. From there onwards, we move into coding.

Programming fluency was also influenced by typing ability. Teacher A observed:

> At Grade 10, I teach them how to use a computer because many have never used one before. I teach them how to type, and once they get comfortable with typing, they move faster when we start coding programs.

After learners grasp basic computing skills, such as typing, teachers transition them to Delphi programming. Although CAPS prioritises Delphi, teachers reported having to compensate for learners' lack of prior exposure to block-based tools like Scratch, typically introduced earlier. They emphasised strategies that leverage Delphi's Integrated Development Environment to support learners with limited prior experience. By scaffolding foundational skills, teachers enable learners to engage meaningfully in programming tasks, contributing to their CT development. These findings highlight the varied instructional practices teachers use to shape learners' mental models for CT in introductory programming. The implications of these findings within the context of MMT are discussed in the next section.

Discussion

The study's findings indicate that teachers employ various instructional strategies such as contextualised learning, adaptive scaffolding, collaborative engagement and structured problem-solving to support learners in developing CT in introductory programming. These practices help learners

address programming challenges through structured guidance and meaningful learning experiences. In addressing Research Questions (RQ) 1 and 2, these strategies reflect how teachers in township schools support the development of mental models for CT in programming instruction. Resource constraints, including limited access to computing devices, underdeveloped digital skills and gaps in prior exposure to visual programming environments, shaped these instructional decisions. Teachers responded by adapting instruction to maximise available tools such as Delphi's Integrated Development Environment and by using analogies or stepwise logic to simplify abstract concepts. Mental model theory offers a valuable lens for interpreting these findings, as it posits that learners construct internal representations of a system through prior knowledge, instruction and experience (Johnson-Laird, 2001). In programming education, instruction should facilitate the development of accurate and flexible mental models that align with current teaching practices (Mills et al., 2024). To provide interpretation, this study links its findings to MMT principles, relates them to existing literature, and examines their broader implications.

One of the most prevalent methods observed in this study is the structured problem-solving approach, which involves improving mental models through monitored exposure and guided practice. In response to RQ1, teachers used a step-by-step instructional procedure that included defining the problem, understanding it, breaking it down, outlining the logic, utilising tools including flowcharts and input–process–output models, and then developing an algorithm. This procedure provides learners a scaffolded approach from conceptual understanding to practical application. This method supports Johnson-Laird's (1983) three principles of MMT as learners construct models based on given premises, prioritise what is true (omitting false alternatives), and adapt models through reasoning and manipulation. Through sequential steps, learners build internal representations of how a solution unfolds, models that evolve as they encounter contradictions or test outcomes. The significance of structured instructional design has been reinforced by a review conducted by Yeni et al. (2024), which indicates that sequential problem decomposition—encompassing the process of breaking down complex problems into manageable sub-problems—promotes CT development.

The limitation of this approach is that different learners have varying capacities for interacting with structured frameworks, prompting the use of additional scaffolding mechanisms to guide less adept learners. Incomplete or inaccurate models have also been found to render it challenging for learners to effectively develop mental models, hindering their ability to apply problem-solving frameworks in unfamiliar circumstances (Qian & Lehman, 2017).

Contextualised learning is another instructional strategy in which teachers ground programming concepts in everyday examples. As part of their response to RQ1, teachers framed algorithms as step-by-step processes, akin to following a recipe or structuring a voting system, thereby facilitating learners' capacity to translate abstract programming concepts into tangible, everyday circumstances. This method coincides with MMT's principle that experiences and prior knowledge shape mental models (Johnson-Laird, 2001). Pursuant to research by Qian and Lehman (2017), learners who become overly reliant on analogies may find it challenging to generalise problem-solving models to other circumstances. This raises the concern of whether learners can derive underlying computing principles and apply them elsewhere beyond the immediate learning environment.

Collaborative learning has emerged as another effective instructional strategy for building mental models. In addressing RQ1, teachers emphasised peer-assisted learning, where learners with greater expertise support their peers, allowing knowledge to be co-constructed through social interaction. Research by Mills et al. (2024) reinforces the importance of distributed expertise, where cognition is not confined to individuals but shared across a learning network. This enables learners to act as 'expert novices,' strengthening their understanding by assisting others and engaging in structured peer dialogue (Mills et al., 2024: 8).

This process resonates with the third principle of MMT, which states that reasoning involves constructing and manipulating mental models (Johnson-Laird, 2001). Peer interaction provides opportunities for learners to test, adjust and reinforce these models through social comparison and explanation, thereby enhancing their understanding of the concepts. Although collaboration can encourage cognitive development, a conceivable detriment is the risk of dependency, where less developed learners rely on peers instead of developing independent problem-solving abilities. This

challenges mental model stability, given that misconceptions may persist if not actively corrected by the teacher (Qian & Lehman, 2017).

Lastly, the study found that adaptive scaffolding is a key instructional strategy for addressing learners' prior knowledge gaps and supporting the development of accurate mental representations of programming concepts. In further addressing RQ1, teachers used scaffolding techniques to sustain engagement, such as introducing basic computing skills and incorporating game-based learning elements. Some also reported using bridging strategies, including visual or block-based programming, before transitioning to text-based environments, such as Delphi. Although not a central theme across all cases, these practices align with existing literature. Consistent with Bidlake et al. (2020), the findings reinforce that novice programmers benefit from structured, adaptive feedback to help them restructure and refine their mental models as they build conceptual fluency.

In addressing RQ2, the study highlights a limitation in introductory programming instruction, where some teachers, owing to curriculum constraints and contextual factors, rely primarily on text-based tools such as Delphi to compensate for learners' limited exposure to visual programming. Although this approach aligns with the current CAPS curriculum for IT, which designates Delphi as the primary language from Grade 10 onwards, research indicates that block-based programming provides a more accessible scaffold for novice learners to construct mental models (Fagerlund et al., 2021; Stamatios, 2024). This raises a broader pedagogical concern about whether the structure and sequencing of CAPS effectively support novice learners in building foundational mental models for programming in the township context.

Despite these findings, several limitations remain in this study. Since teachers were the primary source of information, the reliance on self-reported data introduces potential bias owing to the lack of detail on how they address learners' programming misconceptions. Another limitation is the absence of learners' perspectives, as the research focuses primarily on teachers' instructional practices. The small sample of four teachers limits the generalisability of the findings. Further investigation is needed to explore how learners develop and refine their mental models through these practices and to validate the observed instructional patterns across broader contexts.

Although contextualised learning supports understanding (Fagerlund et al., 2021), it is unclear whether the mental models formed through such strategies are effective across varied programming environments. Another area requiring attention is the impact of different programming tools. The comparative effects of block-based and text-based environments on mental model development remain uncertain. Given ongoing debates about whether block-based tools adequately prepare learners for long-term CT skills (Stamatios, 2024), future research should assess their relative effectiveness in promoting lasting programming competencies.

Conclusions and Recommendations

In conclusion, this study expands upon prior research by providing a mental model perspective on instructional practices in introductory programming education in South African township schools. Compared with previous research on student learning outcomes, this study highlights how teachers can shape learners' CT skills through structured instructional design. By integrating MMT principles with instructional strategies, this study provides a theoretical basis for understanding how diverse instructional approaches shape the mental models of novice programmers.

The findings underscore the importance of structured problem-solving frameworks in supporting the development of CT. Teachers should design contextualised learning experiences that promote knowledge transfer across programming environments. Peer-assisted learning must strike a balance between collaboration and individual problem-solving to prevent over-reliance on peers.

At the policy level, programming curricula should incorporate both text-based and block-based learning strategies to accommodate diverse learner backgrounds in township schools with limited exposure to foundational computing concepts. Sustained professional development in scaffolding techniques is crucial for equipping teachers with strategies to support learners with limited computing experience (Ni et al., 2023). However, it is recognised that the professional development practices

reported by teachers may reflect aspirational or idealised strategies rather than the full extent of constraints faced in practice.

Addressing resource constraints and the lack of institutional collaboration is necessary to improve programming education in township schools. Interventions should focus on supporting teachers in adapting curriculum content to local realities, including limited access to devices, patchy connectivity and uneven prior computing knowledge. Where access to block-based tools is absent, alternative strategies such as unplugged programming can serve as interim scaffolds. Nonetheless, it is acknowledged that implementing these recommendations may face limitations related to cost, infrastructure and sustainability. Therefore, any intervention must be responsive and developed in partnership with schools to ensure feasibility. Ultimately, this study contributes to discussions on evidence-based programming instruction, reinforcing the need for structured teaching approaches that develop transferable CT skills while remaining sensitive to contextual constraints in township classrooms.

Disclosure Statement

No potential conflict of interest was reported by the author.

Data Availability Statement

The data supporting this study's findings are not publicly available due to ethical considerations and participant confidentiality. However, de-identified excerpts or aggregated results may be shared upon reasonable request and with appropriate institutional approvals.

ORCID


Mashite Tshidi http://orcid.org/0000-0002-1288-1576

References

Aruleba, K., & Jere, N. (2022). Exploring digital transforming challenges in rural areas of South Africa through a systematic review of empirical studies. *Scientific African, 16*, e01190. https://doi.org/10.1016/j.sciaf.2022.e01190

Belmar, H. (2022). Review on the teaching of programming and computational thinking in the world. *Frontiers in Computer Science, 4*, 997222. https://doi.org/10.3389/fcomp.2022.997222

Bidlake, L., Aubanel, E., & Voyer, D. (2020). Systematic literature review of empirical studies on mental representations of programs. *Journal of Systems and Software, 165*, 110565. https://doi.org/10.1016/j.jss.2020.110565

Braun, V., & Clarke, V. (2019). Reflecting on reflexive thematic analysis. *Qualitative Research in Sport, Exercise and Health, 11*(4), 589–597. https://doi.org/10.1080/2159676X.2019.1628806

Cheah, C. S. (2020). Factors contributing to the difficulties in teaching and learning of computer programming: A literature review. *Contemporary Educational Technology, 12*(2), ep272. https://doi.org/10.30935/cedtech/8247

Cheruiyot, K. C., & Chepngetich, K. J. (2023). Culture in design of coding toolkits for young learners in developing economies in Africa: A review. *Current Journal of Applied Science and Technology, 42*(24), 43–50. https://doi.org/10.9734/cjast/2023/v42i244177

Creswell, J. W., & Poth, C. N. (2018). *Qualitative inquiry and research design: Choosing among five approaches* (4th ed.). Sage Publications.

Ezeamuzie, N. O., & Leung, J. S. (2022). Computational thinking through an empirical lens: A systematic review of literature. *Journal of Educational Computing Research, 60*(2), 481–511. https://doi.org/10.1177/07356331211033158

Fagerlund, J., Häkkinen, P., Vesisenaho, M., & Viiri, J. (2021). Computational thinking in programming with Scratch in primary schools: A systematic review. *Computer Applications in Engineering Education, 29*(1), 12–28. https://doi.org/10.1002/cae.22255

Heinonen, A., Lehtelä, B., Hellas, A., & Fagerholm, F. (2023). Synthesizing research on programmers' mental models of programs, tasks and concepts—A systematic literature review. *Information and Software Technology*, 107300. https://doi.org/10.1016/j.infsof.2023.107300

Henley, A., Ball, J., Klein, B., Rutter, A., & Lee, D. (2021). An inquisitive code editor for addressing novice programmers' misconceptions of program behavior. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)* (pp. 165–170). IEEE. https://doi.org/10.1109/ICSE-SEET52601.2021.00026

Johnson-Laird, P. N. (1983). Mental models: Towards a cognitive science of language, inference, and consciousness (No. 6). Harvard University Press.

Johnson, F., McQuistin, S., O'Donnell, J., & Cutts, Q. (2022). Experience report: Identifying unexpected programming misconceptions with a computer systems approach. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 1* (pp. 325–330). ACM. https://doi.org/10.1145/3502718.3524775

Johnson-Laird, P. N. (2001). Mental models and deduction. *Trends in Cognitive Sciences, 5*(10), 434–442. https://doi.org/10.1016/s1364-6613(00)01751-4

Julie, H., & Bruno, D. (2020). Approach to develop a concept inventory informing teachers of novice programmers' mental models. In *2020 IEEE Frontiers in Education Conference (FIE)* (pp. 1–9). IEEE. https://doi.org/10.1109/fie44824.2020.9274045

Kadar, R., Wahab, N. A., Othman, J., Shamsuddin, M., & Mahlan, S. B. (2021). A study of difficulties in teaching and learning programming: A systematic literature review. *International Journal of Academic Research in Progressive Education and Development, 10*(3), 591–605. https://doi.org/10.6007/IJARPED/v10-i3/11100

Mills, K. A., Cope, J., Scholes, L., & Rowe, L. (2024). Coding and computational thinking across the curriculum: A review of educational outcomes. *Review of Educational Research, 95*(3) . https://doi.org/10.3102/00346543241241327

Mkhize, T. R., & Davids, M. N. (2021). Towards a digital resource mobilisation approach for digital inclusion during COVID-19 and beyond: A case of a township school in South Africa. *Educational Research for Social Change, 10*(2), 18–32. https://doi.org/10.17159/2221-4070/2021/v10i2a2

Mkhonto, B. S., & Mubangizi, B. C. (2024). Digital divide or digital bridge? Evaluating the impact of ICT integration in South Africa's rural schools. *International Journal of Social Science Research and Review, 7*(7), 132–145. https://doi.org/10.47814/ijssrr.v7i7.2142

Ni, L., Bausch, G., & Benjamin, R. (2023). Computer science teacher professional development and professional learning communities: A review of the research literature. *Computer Science Education, 33*(1), 29–60. https://doi.org/10.1080/08993408.2021.1993666

Ogegbo, A. A., & Ramnarain, U. (2022). Teachers' perceptions of and concerns about integrating computational thinking into science teaching after a professional development activity. *African Journal of Research in Mathematics, Science and Technology Education, 26*(3), 181–191. https://doi.org/10.1080/18117295.2022.2133739

Ortiz, J. S., Moreira, C., Menezes, K., Ferrari, B., Silva Junior, D., & Pereira, R. (2023). Computational thinking and mental models: Promoting digital culture in the youth and adult education. *Interacting with Computers, 35*(2), 91–104. https://doi.org/10.1093/iwc/iwac028

Pellas, N. (2023). Exploring relationships among students' computational thinking skills, emotions, and cognitive load using simulation games in primary education. *Journal of Computer Assisted Learning, 39*(5), 1576–1590. https://doi.org/10.1111/jcal.12819

Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education, 18*(1), 1–24. https://doi.org/10.1145/3077618

Rich, P. J., Larsen, R. A., & Mason, S. L. (2021). Measuring teacher beliefs about coding and computational thinking. *Journal of Research on Technology in Education, 53*(3), 296–316. https://doi.org/10.1080/15391523.2020.1771232

Stamatios, P. (2024). Can preschoolers learn computational thinking and coding skills with ScratchJr? A systematic literature review. *International Journal of Educational Reform, 33*(1), 28–61. https://doi.org/10.1177/10567879221076077

Tshidi, M., & Dewa, A. (2024). The promise and peril of Coding & Robotics education in South Africa: A scoping review of teacher preparation and generative artificial intelligence's potential for delivering equity. Journal of Education (University of KwaZulu-Natal), (96), 140–164. https://doi.org/10.17159/2520-9868/i96a08

Waite, J., & Sentance, S. (2021). *Teaching programming in schools: A review of approaches and strategies*. Raspberry Pi Foundation.

Yang, S., Baird, M., O'Rourke, E., Brennan, K., & Schneider, B. (2025). Decoding debugging instruction: A systematic literature review of debugging interventions. *ACM Transactions on Computing Education, 24*(4), 1–44. https://doi.org/10.1145/3690652

Yeni, S., Grgurina, N., Saeli, M., Hermans, F., Tolboom, J., & Barendsen, E. (2024). Interdisciplinary integration of computational thinking in K–12 education: A systematic review. *Informatics in Education, 23*(1), 223–278. https://doi.org/10.15388/infedu.2024.08