Journal of Education, 2024
Issue 97, http://journals.ukzn.ac.za/index.php/joe doi: http://dx.doi.org/10.17159/2520-9868/i97a11

Towards data-driven interventions: Leveraging learning analytics to support programming education for grade 10 learners in South African schools

Mashite Tshidi

Division of Science and Technology, Wits School of Education, University of the Witwatersrand, Johannesburg, South Africa
tshidimashite@gmail.com
https://orcid.org/0000-0002-1288-1576

Alton Dewa

Division of Science and Technology, Wits School of Education, University of the Witwatersrand, Johannesburg, South Africa
alton.dewa@wits.ac.za
https://orcid.org/0000-0002-0776-4447

(Received: 30 May 2024; accepted: 14 November 2024)

Abstract

Programming is increasingly incorporated into school curricula worldwide to foster essential 21st century skills. However, many educational systems face challenges in integrating it effectively because of limited resources and support. In South Africa, a lack of tools further compounds these challenges, making it difficult for teachers to identify and address learners' specific needs. In recognising these challenges, we aimed in this study to develop and validate a Learning Analytics (LA) model to identify challenging programming concepts for Grade 10 learners in South Africa. Using the LA five-step model, we employed Microsoft Power BI for its analytical, visualisation, and AI-driven forecasting capabilities to analyse historical examination data systematically. The resulting forecasting model identified five key areas in which learners struggle: conditional statements; problem conceptualisation/solution design; debugging/exception handling; abstraction/pattern recognition; and class/object differentiation. Our findings demonstrate the potential of LA-powered models to guide targeted, data-driven interventions, supporting improved learning outcomes, and engagement in programming for Grade 10 learners.

Keywords: programming, learning analytics, Grade 10 learners, forecasting model, data-driven interventions

Introduction

Programming is a cornerstone of Computer Science, or Information Technology (IT) as it is referred to in South African public schools. It integrates knowledge from science, technology, engineering, arts, and mathematics (STEAM) to address problems through computational thinking. The positive impact of computational thinking on cognitive development, problem-solving abilities, and preparing learners for a technology-driven future is widely acknowledged (Tikva & Tambouris, 2021). South Africa, along with many other countries, recognised this value and integrated programming into their grades R–12 curriculum to cultivate essential digital skills (Sun et al., 2022). For instance, in 2021, on South Africa, the Department of Basic Education (DBE) introduced the "Coding and Robotics" subject spanning Grades R–9 to cultivate problem-solving, critical thinking, collaboration, and creativity in young minds in preparing them for a digital age (Department of Basic Education, 2021).

Concurrently, the DBE has long offered IT as a subject designed for secondary learners in Grades 10–12 aimed at developing foundational programming skills (Department of Basic Education, 2011). However, programming is perceived as challenging, particularly for novices, because of its abstract concepts and problem-solving demands (Cheah, 2020; Kadar et al., 2021). For many Grade 10 learners in South African schools, programming is their first encounter with structured logic and computational thinking, and this initial experience often shapes their long-term attitudes toward the subject. Research indicates that these learners encounter significant challenges in grasping programming fundamentals and this leads to negative perceptions, a lack of motivation, and deficiencies in problem-solving skills (Marimuthu & Govender, 2018). Addressing these issues requires programming teachers to gain a nuanced understanding of these challenges and use effective tools to support learners in overcoming them (Cheah, 2020; Tikva & Tambouris, 2021).

Educators who teach programming in South African secondary schools face challenges in helping learners develop foundational programming skills, such as understanding selection and iteration statements. These challenges are compounded by the structure of secondary school schedules, with short daily lessons that limit opportunities to reinforce these concepts effectively (Koorsse et al., 2015). Given the limited classroom time available, this time constraint reduces teachers' ability to respond to individual learners' difficulties and monitor their understanding. Furthermore, the lack of readily available tools to identify and address specific learning challenges hampers teachers' capacity to provide timely and targeted interventions for learners experiencing difficulty with foundational concepts (Lee et al., 2023).

In response to these ongoing challenges, we propose a LA-powered model designed to measure and analyse learner interactions in the programming environment, giving teachers insights and opportunities to improve learning outcomes. Research indicates that LA offers a data-driven framework to understand learner challenges in programming, with evidence showing its effectiveness in predicting learner performance, identifying at-risk students, and enhancing instructional strategies through personalised insights (López-Pernas et al., 2021; Omer et al., 2023; Veerasamy et al., 2022). However, while these findings underscore the potential of data-driven approaches in improving programming education, existing applications of LA in programming education have focused predominantly on higher education, leaving its use at the secondary school level relatively unexplored.

In this study, we address this gap by proposing a model that leverages LA through Power BI to identify programming concepts that challenge Grade 10 learners in South African schools. The goal is to provide teachers with data-driven insights to tailor instruction and offer individualised support, thereby bridging the gap between pedagogical programming needs and available resources. This study aims to demonstrate the effectiveness of LA in identifying common programming challenges and delivering targeted interventions that improve learner comprehension and performance. This investigation is guided by the following questions:

- What programming concepts, identified through LA, present challenges to Grade 10 learners in South African programming classrooms?
- How can insights from LA guide interventions to improve learning outcomes and engagement for these learners?

By exploring the application of LA in programming education, we address challenges in the field while contributing to the ongoing discourse on exploiting data analytics to improve educational outcomes (see Msweli et al., 2023).

Literature review

Programming occupies a central role in preparing learners for a future increasingly defined by digital technologies. Its prominence stems from its perceived effectiveness in cultivating essential 21st-century skills such as problem-solving, critical thinking, and computational logic (Tikva & Tambouris, 2021). As a component of STEAM education, programming equips learners to apply fundamental computer science principles to real-world challenges (Sun et al., 2022). However, programming languages are often complex in both syntax and semantics, posing substantial challenges to grades R–12 learners worldwide (Lee et al., 2023). These complexities can result in negative perceptions of programming courses, and this contributes to high dropout rates and a general aversion to the subject (Cheah, 2020).

One of the fundamental challenges of programming education lies in the abstraction required to understand its core concepts. These concepts are often not immediately tangible or easily visualised, making programming particularly challenging for both learners and teachers. The process requires a blend of knowledge from many different disciplines, such as mathematics and science, to interpret and apply programming concepts effectively. Teachers and learners must possess both declarative knowledge (understanding the syntax and semantics of programming languages) and procedural knowledge, which involves logical thinking and generalisation skills for program design and problem-solving (Santos et al., 2022). However, many learners struggle with logical reasoning and abstraction, often as a result of limited problem-solving abilities in mathematics (Kadar et al., 2021). These difficulties impact their ability to understand programming structures, implement syntax correctly, and troubleshoot errors effectively (Dlamini & Dewa, 2022). With their teachers continuing to rely on traditional teaching methods, such as using PowerPoint as the primary mode of instruction, these approaches may not align with the interactive learning preferences of these learners (Cheah, 2020; Dlamini & Dewa, 2022). To enhance learners' comprehension of programming concepts, instructional methods incorporating programming-specific tools are needed since they help make abstract concepts tangible and enable active problem-solving and practical tool usage (Cheah, 2020). In the absence of these instructional modifications, programming courses are perceived as monotonous and tedious, thereby highlighting the need for adaptive instructional methods to facilitate learners' understanding and coding abilities (Kadar et al., 2021).

Innovations in programming education tools

Research has explored innovative tools and methodologies for teaching programming at the R –12 level. For instance, Kroustalli and Xinogalos (2021) investigated the educational effectiveness of text-based serious games using conventional programming languages. In particular, their study examined how CodeCombat, a text-based game that uses Python, impacted lower secondary learners' understanding of programming fundamentals. By comparing CodeCombat with traditional teaching methods, their study found that this serious game was effective in enhancing learners' grasp of core programming concepts. Furthermore, learners expressed positive attitudes towards the use of serious games, even when the game was text-based and used a conventional programming language. In a similar effort to address the need for timely intervention in programming education, Lee et al. (2023) developed the Precision Education-based Timely Intervention System (PETIS) that uses deep learning and image processing to detect learner difficulties in programming and provides tailored assistance to teachers. A quasi-experimental study evaluated the impact of PETIS on the grades R–12 programming curriculum, revealing notable improvements in both learners' programming skills and affective learning outcomes. Subsequently, qualitative data from interviews with teachers and learners indicated a shared perception that PETIS positively enhanced the programming learning experience.

Given that these interventions were designed and tested in Western contexts, this can pose challenges for their implementation in African settings. Educational initiatives created in Western settings may fail to account for historical, socio-economic, and infrastructural differences in African countries, often resulting in limited success or even exacerbating existing educational inequalities (Prinsloo & Kaliisa, 2022). However, the availability of learner data in educational institutions in these contexts and the expanding discipline of LA has made it possible to gain insights into the methods and approaches learners use to acquire new knowledge and skills (López-Pernas et al., 2021). In the context of African programming education, Ihantola et al. (2015) noted that researchers can find hidden dimensions for creating optimal interventions by examining learner data from programming courses. Using LA, efforts can be directed to assist instructors in addressing the challenges programmers face and enhancing their learning experiences (Omer et al., 2023).

Learning analytics applications and techniques in programming education

The application of LA in programming education is a burgeoning field of research, although its methodologies and techniques remain largely unvaried. Most LA studies in programming education rely heavily on descriptive and predictive analysis using frequency data to monitor learner activity or forecast academic progress (López-Pernas et al., 2021). Pereira et al. (2020) used an Online Judge system called CodeBench to provide fine-grained insights into the early detection of effective and ineffective behaviours among learners in introductory programming. Building on this momentum, Helling & Haelermans (2022) applied an LA Dashboard and email notifications to monitor learner performance in a Java course. This randomised controlled experiment involving 556 first-year students revealed improvements in online course performance, although effects varied regarding learner behaviour and final examination scores. This prompted a suggestion that LA tools need adaptation to fit diverse learning contexts and assessment structures. A further study by Veerasamy et al. (2022) focused on ongoing formative assessment scores as indicators of learner engagement in programming education. The findings suggested a potential relationship between continual assessment performance and learner engagement that led to the development of a learner engagement indicator as a practical model of LA to track and enhance engagement in ongoing assessment tasks. Collectively, these studies highlight the nature of LA in programming education and its potential to personalise learning experiences by providing targeted support based on behavioural indicators and performance (Brocker et al., 2022).

Despite these findings, LA's application in programming education comes with technical and contextual challenges and limitations. A primary limitation lies in the data quality and model interpretability of LA models, which often depend on a subset of Artificial Intelligence (AI) known as machine learning. These models, although powerful, frequently require computational resources that are challenging to obtain in under-resourced environments (Veerasamy et al., 2022). Furthermore, the interpretability of the findings from these models can pose issues for teachers unfamiliar with advanced statistical methods, thereby reducing their practical effectiveness in everyday teaching contexts (Brocker et al., 2022).

Ethical concerns about data privacy add complexity to LA's application since unregulated data collection and analysis can reinforce biases, particularly in contexts comparable to South Africa where data quality varies widely and may reflect historical inequities (Prinsloo & Kaliisa, 2022). Socio-economic disparities in South African schools such as limited digital access and insufficient technical support pose further barriers to the effective implementation of LA solutions (Msweli et al., 2023). These limitations indicate the need for LA models that are technically sound and culturally adaptable to the unique contexts of South African classrooms (Janse van Vuuren, 2020).

Limitations in current learning analytics research

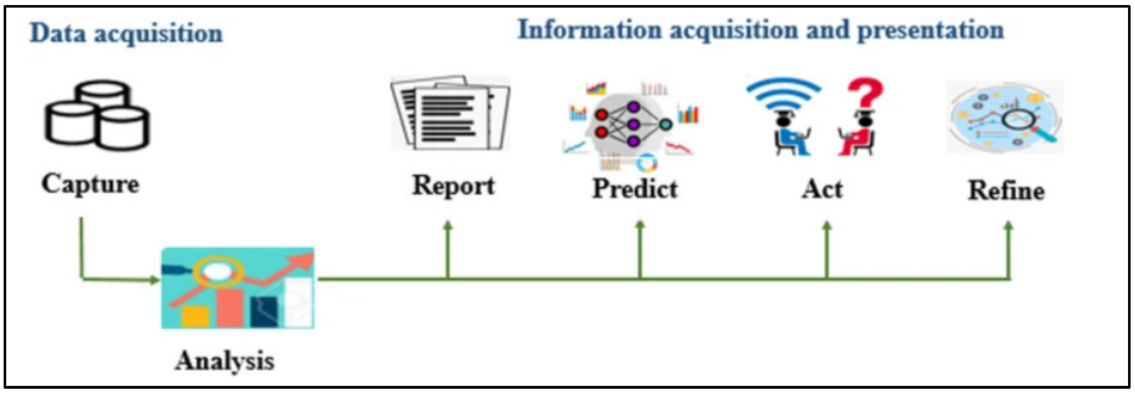
A granularity gap in the literature exists in relation to applying LA to programming education across different grade levels. Most research in this area has focused on higher education and offers broad insights that may not address the distinct challenges and requirements at the lower grades (Prinsloo & Kaliisa, 2022). Specifically, there is limited research on how LA can support early-stage programming education (Sun et al., 2022). In South Africa's IT subject, Grade 10 is pivotal because it introduces learners to fundamental programming concepts and structures (Department of Basic Education, 2011). However, programming education at this level is often challenging because of the abstract nature of the concepts, syntax complexity, and the problem-solving skills required (Kadar et al., 2021). Consequently, Grade 10 represents a particularly challenging phase for learners who are still in the novice stages of programming (Marimuthu & Govender, 2018). Although some studies have examined the use of LA to identify learning difficulties in programming, there remains a gap in developing forecasting models tailored to identify specific programming concepts that may be challenging for learners. Such models would bridge the gap between historical performance data and proactive interventions and provide tailored support to enhance learners' understanding and engagement (Utamachant et al., 2023).

In this study, we aimed to address these gaps by investigating the application of LA to enhance programming education for Grade 10 learners in South Africa. Through LA, this research seeks to identify challenging programming concepts, develop a forecasting model to foresee learning difficulties, and provide targeted support. In doing so, it considers the socio-economic disparities and digital access limitations unique to South African educational contexts to contribute to the broader literature on how LA can support programming education in diverse environments.

Conceptual framework

The conceptual underpinning of this study is grounded in the LA five-step model developed by Campbell and Oblinger (2007) to provide a structured framework for applying LA in the domain of programming education. Figure 1 depicts this model, and this study leverages this model to address the conceptual challenges faced by Grade 10 programming learners in South Africa.

Figure 1
An illustration of the LA five-step model (Omer et al., 2023).



The LA five-step model outlined ensures a structured and iterative approach for capturing, analysing, predicting, acting on, and developing interventions designed to improve learning outcomes and provide an engaging programming education experience for learners.

1. *Capture* entails systematic data collecting on learner interactions, performance, and engagement in the programming environment. This includes capturing code snippets, debugging steps, error messages, and time spent completing the task (Kadar et al., 2021). This granular data forms the foundation layer for the *analysis* carried out to disclose the features that have been evaluated and to find out information about prediction, intervention, and refining (Omer et al., 2023).
2. *Report* follows the *analysis* and its data is processed and transformed into actionable insights through reports tailored for both researchers and teachers (Campbell & Oblinger, 2007). These reports identify patterns and concepts of programming that learners find difficult but also visualise learners' progress and highlight potential future challenges that can be mitigated with target interventions (Lee et al. 2023; Msweli et al., 2023).
3. *Prediction* involves applying machine learning algorithms to the analysed reported data to determine challenges that learners face in programming concepts or tasks (Omer et al., 2023). This enables the proactive identification of learners who may be at risk of falling behind and the anticipation of events in which targeted interventions may be required (Veerasamy et al., 2022).
4. *Act* uses the information gleaned from predictions and reports so that tailored strategies can be created and put into practice to address the learning obstacles and programming requirements of learners (Omer et al., 2023).
5. *Refinement* fosters an iterative process of reflection and refinement in which teachers regularly analyse the efficacy of interventions, adjust techniques based on continuing data input, and change strategies under the evolving needs of learners (Campbell & Oblinger, 2007).

In considering this approach, the two research questions (RQs) that this study seeks to answer are:

> RQ1: What programming concepts, identified through LA, pose a challenge for Grade 10 learners in South African programming classrooms?
>
> RQ2: How can insights from LA guide interventions to improve learning outcomes and engagement for these learners?

The reports identify patterns and areas of difficulty that map them to RQ1 to understand the programming concepts that pose a challenge to Grade 10 learners. Prediction algorithms refine this understanding by anticipating individual learners' potential challenges with future programming concepts (Omer et al., 2023) and probe targeted interventions that can be developed to address RQ2. These interventions address expected challenges and improve learning outcomes and engagement (Campbell & Oblinger, 2007). The goal is to answer these research questions by offering an understanding of the challenges faced by Grade 10 programming learners in South Africa by using the LA five-step model as a structured framework to demonstrate the potential of LA to personalise learning and optimise engagement (Utamachant et al., 2023) in South Africa, thereby enriching the field of programming education with data-driven interventions.

Methodology

In this study, we used a quantitative research approach to explore the programming concepts that pose challenges for Grade 10 learners in South Africa. The decision to use a quantitative method was based on its ability to deliver objective and statistical insights into the identified research problem (see Creswell & Plano Clark, 2018). This approach allowed for systematically gathering, analysing, and interpreting examination marks by concentrating solely on quantitative data to facilitate learner performance assessment (Campbell & Oblinger, 2007). This method also adhered to the LA five-step model in leveraging the ability to systematically capture, analyse, predict, and intervene based on the understanding gleaned from quantitative data (Omer et al., 2023). Consequently, the quantitative data gathered from examination marks in this study aimed to corroborate findings and provide perspective on the use of LA in programming education.

Data collection

Data was collected in the format of anonymised examination marks of Grade 10 programming examinations across four schools to ensure a diverse representative sample of 120 learners. The examination, which spanned 2 hours, was designed to test key programming concepts through practical coding tasks. Table 1 outlines the topics covered in the Grade 10 programming curriculum, as derived from DBE (2011), that formed the basis of the examination. A pilot study was conducted at one school to validate the examination content and ensure that it assessed learners' understanding accurately.

Table 1
Grade 10 programming topical areas (Department of Basic Education, 2011).

Topical areas	Subtopics
Introduction to Delphi programming	Familiarity with the Delphi Integrated Development Environment
Variables and data types	Understanding of global and local variables, naming conventions, and data types (integers, strings, floats, and Booleans).
Operators and expressions	Basic arithmetic operators, logical comparisons, calculations
Problem conceptualisation/Solution design	Applying algorithms.
Event handling	Basic event handling
Conditional statements	If and if-then-else constructs

Topical areas	Subtopics
Debugging/Exception handling	Basic validation and debugging techniques
Abstraction/Pattern Recognition	Using programming principles and algorithmic development; iteration constructs (for)

The examination marks collected were enhanced by anonymised code excerpts. By including code excerpts in examination marks, we gained insight into learners' comprehension of programming concepts, procedures taken to solve problems, encountered errors, and debugging approaches (Kadar et al., 2021). Furthermore, timestamps recorded in the pilot allowed an analysis of the time spent on each section of the examination, thereby, providing insights into how task complexity affected time allocation and performance. This data informed the development of a forecasting model to identify programming concepts that most challenged the learners.

Data analysis

The data was analysed using Power BI, selected for its advanced analytical and visualisation and AI-driven forecasting capabilities. Power BI's rich suite of tools, including stacked bar charts, clustered column charts, and scatter plots facilitated the visual exploration of data patterns and the application of AI for trend analysis (Singh et al., 2023). Power BI's predictive insights enabled us to forecast challenging programming concepts and develop targeted interventions aimed at mitigating these learning difficulties.

To ensure the rigour of the analysis, statistical techniques such as variable selection, normalisation, and regression analysis were applied. Variable selection was used to isolate key indicators, such as anonymised examination scores and coding excerpts, that reflect learner performance. Normalisation was subsequently conducted to standardise these variables to enhance comparability across data points. A regression analysis was then employed to predict learner performance based on coding features such as length, complexity, readability, and constructs used. This made it possible to identify programming challenges and factors affecting performance systematically.

The dataset was split into a training set (70%) and a testing set (30%), with iterative tuning to optimise predictive accuracy. The efficacy of the model's predictions in the testing set was measured using evaluation metrics which include accuracy, precision, and recall within the categories of false positives (FP), false negatives (FN), true positives (TP), and true negatives (TN), calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Accuracy assessed the model's overall reliability in predicting correctly.

$$Precision = \frac{TP}{TP + FP}$$

Precision quantified the model's accuracy in identifying true positive cases of challenging concepts.

$$Recall = \frac{TP}{TP + FN}$$

Recall measured the model's effectiveness in detecting all true cases of challenging concepts. Ultimately, Power BI's visualisation tools supported these analyses to provide representations of data patterns and model predictions that guided the identification of challenging programming concepts and the development of targeted interventions to address learning difficulties.

Ethical considerations

Although the primary goal of using LA was to integrate data-driven approaches into programming education, ethical considerations were fundamental to ensure participant protection and research integrity. Following approval from the university's ethics committee and obtaining permission from the DBE, consent was secured from participating schools and learners. Consent forms and information sheets provided transparency, and outlined the study's objectives, data use, and confidentiality measures, thus supporting informed consent and participant autonomy (Prinsloo & Kaliisa, 2022).

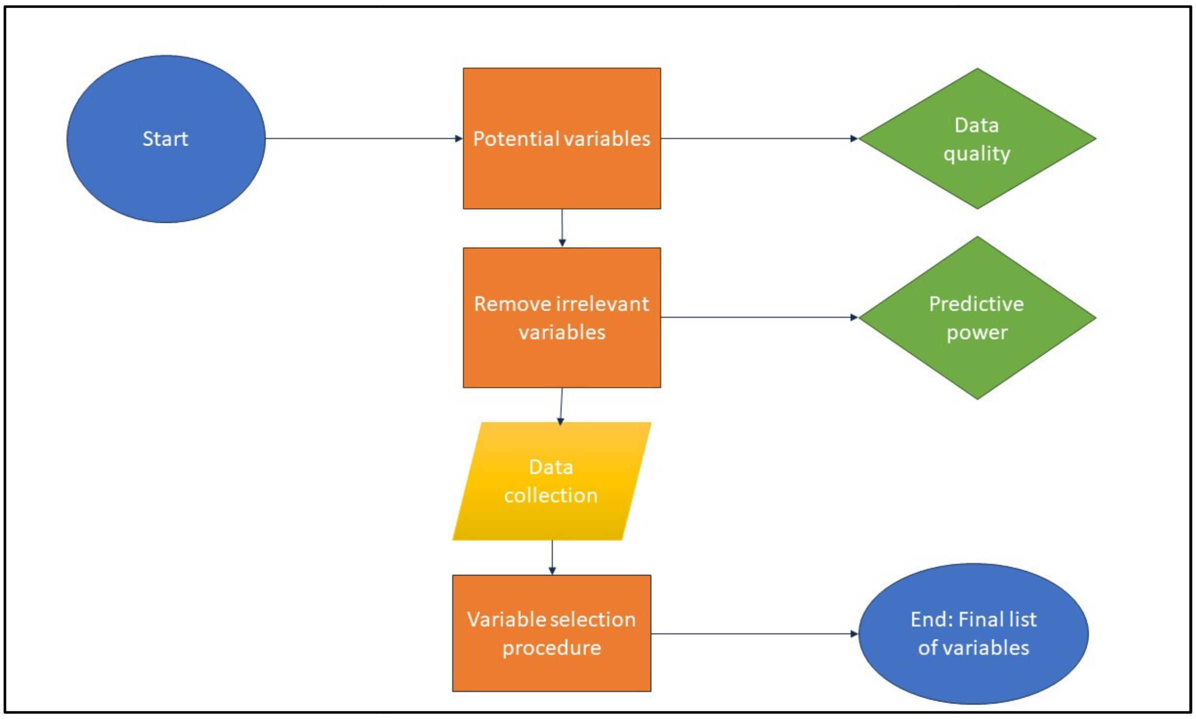
In addition to privacy and consent, this study considered the ethical implications of using LA. To promote equitable support, the LA-powered model focused on identifying challenging areas without stigmatising individual learners. Findings were used to inform instructional improvements without singling out low-performing learners. To further ensure confidentiality, anonymised data was securely stored and managed to prevent the identification of individual learners and schools. By prioritising participant privacy, well-being, and fairness in data usage, we sought to integrate data-driven insights into programming education responsibly.

Results

The data-gathering phase initiated the development of a data-driven approach in the form of a forecasting model aimed at identifying programming concepts that present challenges for Grade 10 learners. This model was built using historical examination data on a range of programming concepts, as shown in Table 1, to ensure comprehensive coverage of the syllabus.

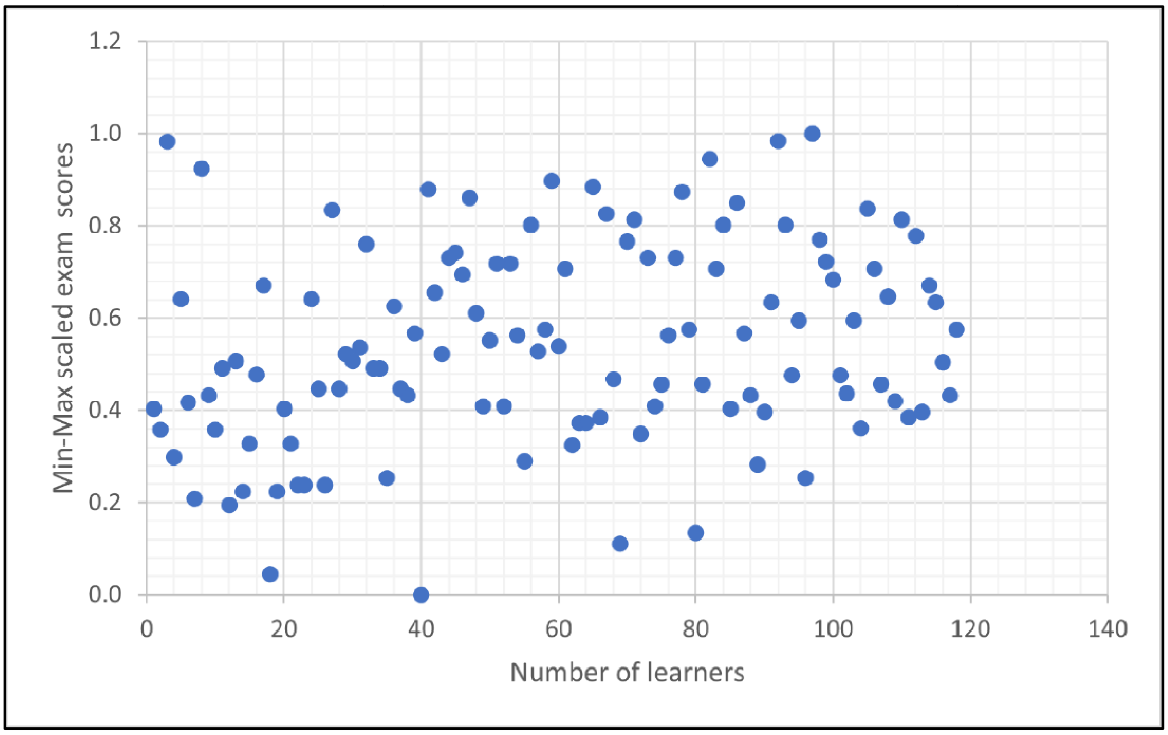
Following data collection, we conducted a variable selection process to focus the model on the most relevant indicators of learner performance. The choice of variable selection as a technique was made to enhance model accuracy by narrowing down to the most impactful features, such as anonymised examination scores, time spent on tasks, and coding excerpts. This selection helped reduce noise in the data, allowing the model to produce reliable insights specific to programming challenges. Figure 2 outlines the procedure used to finalise the variables that were instrumental in identifying performance metrics in programming.

Figure 2
Variable selection procedure for the LA model


Start
Potential variables
Data quality
Remove irrelevant variables
Predictive power
Data collection
Variable selection procedure
End: Final list of variables

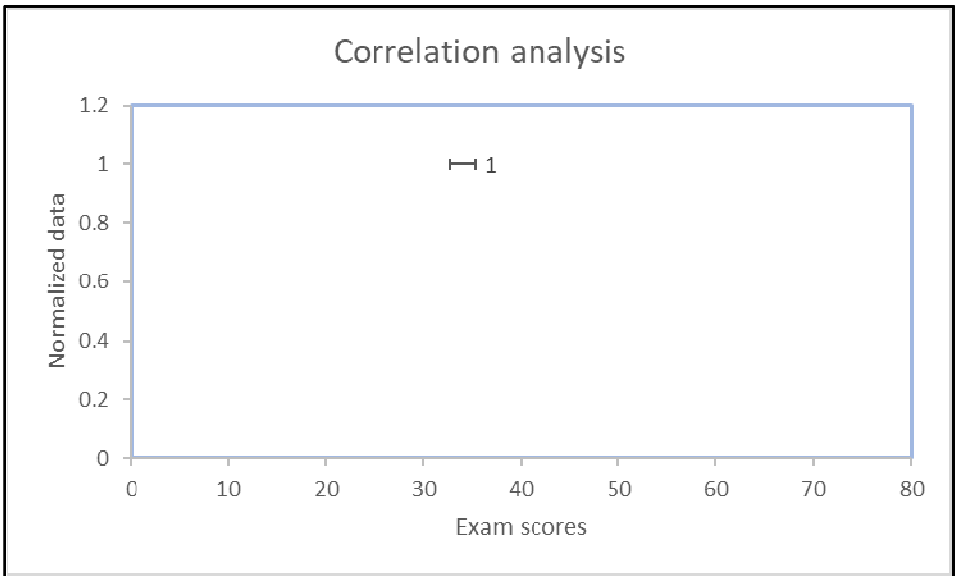
Once the variables were selected, normalisation was applied using the min-max scaling approach that rescaled each variable to a range between 0 and 1. This normalisation step was essential because it allowed variables on different scales to be compared meaningfully. The resulting scatter plot in Figure 3 shows the normalised distribution of examination scores.

Figure 3
Min-Max scaling visualisation



After normalisation, statistical analyses were performed to examine relationships within the data. As shown in Figure 4, the correlation analysis yielded a perfect correlation coefficient of 1 between examination scores and normalised data, indicating a very strong linear relationship between the variables. This result suggests a high degree of consistency between examination scores and the variables selected and supports the validity of the model (Veerasamy et al., 2022).

Figure 4
The correlation between examination scores and normalised data for identifying challenging programming concepts.

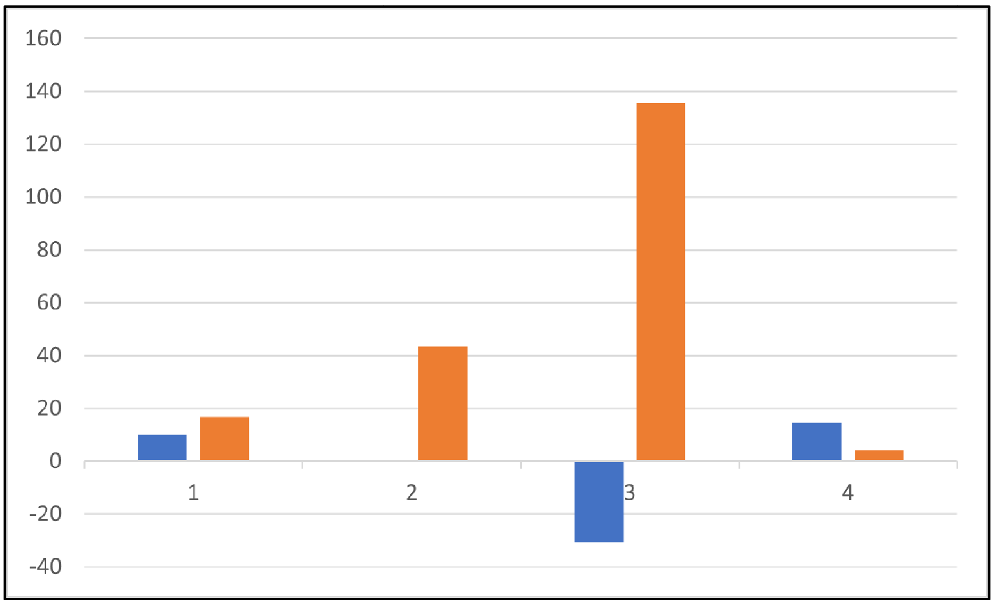


A regression analysis was then conducted to predict learner performance based on four specific code features: code length; cyclomatic complexity; code readability; and code constructs. Regression was chosen for its effectiveness in quantifying the effect of individual predictors on examination performance that would provide insights into each feature's impact. The results showed:

1. **Code length** – Each unit increase in code length led to an approximately 9.94-point increase in examination results
2. **Cyclomatic complexity** – The coefficient range from 0 to approximately 43.42 showed a negative impact on examination scores
3. **Code readability** – A notable coefficient from -30.84 to 135.67 demonstrated that improvements in code clarity correlated positively with examination scores.
4. **Code constructs** – Each unit increase in the use of effective code constructs correlated with a 14.56-point rise in scores.

Figure 5 provides a visual summary of the regression analysis results, illustrating the distinct effects of each code feature on examination scores.

Figure 5
Regression analysis results of code features on examination scores



While the combination of these analysis techniques and visualisations aimed to provide an understanding of the relationship patterns within the datasets, there was a need to test the model's performance and reliability. Validation acted as the stress test that assessed the forecasting model's applicability based on its code features. A hold-out validation technique was implemented to divide the dataset into training and testing sets. As shown in Figure 6, the training set was used to develop and refine the forecasting model. In contrast, the testing set was kept separate to validate the model's applicability to new data.

Figure 6
Hold-out validation technique

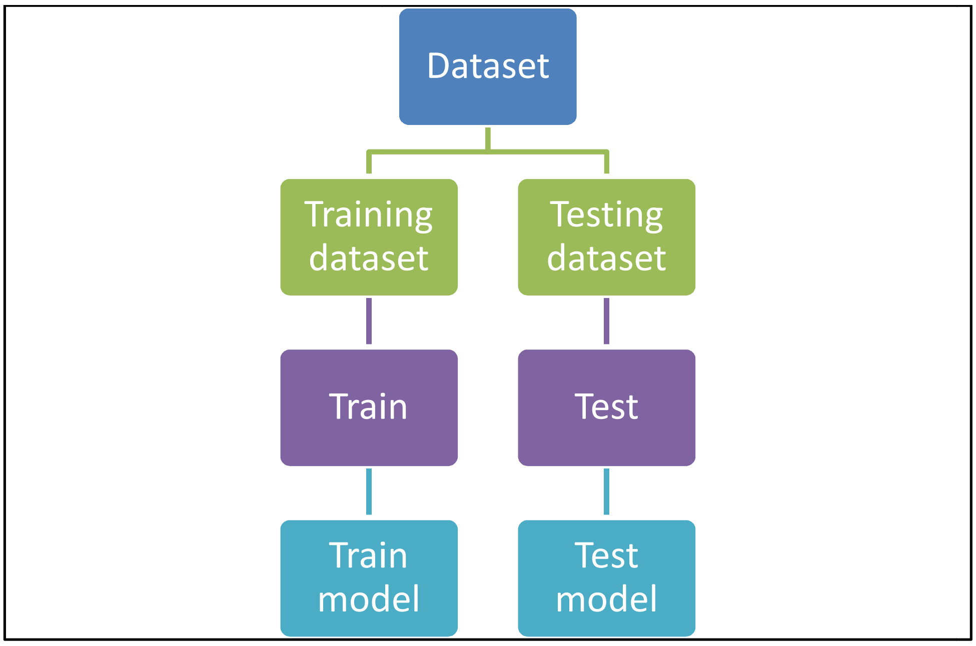
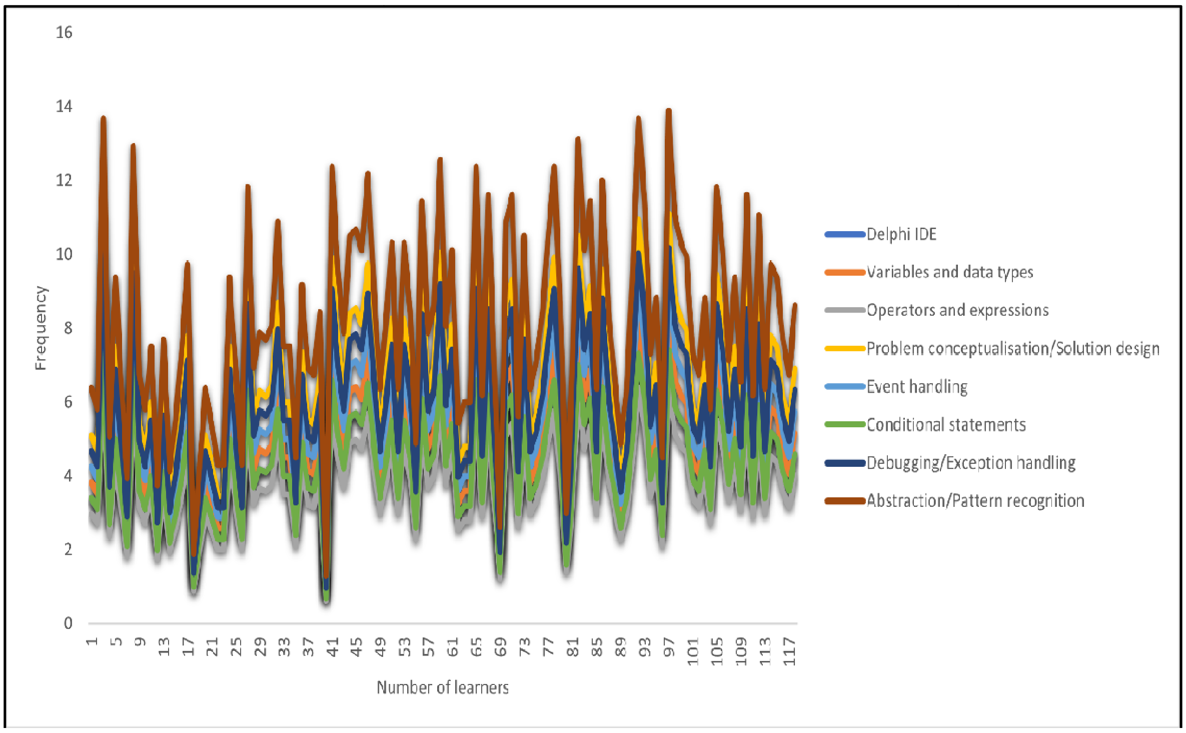


Figure 7 depicts the forecasting model identifying the challenging programming concepts for Grade 10 learners.

Figure 7
Forecasting model



The model's performance was assessed through a confusion matrix and key evaluation metrics that provide a comprehensive view of its predictive capacity. Tables 2 and 3 summarise these metrics and reinforce the model's reliability in identifying challenging concepts.

Table 2
Confusion matrix for the forecasting model

		Actual values	
		Positive	**Negative**
Predicted values	Positive	TP (101)	FP (0)
	Negative	FN (6)	TN (11)

Table 3
Results of the evaluation metrics

Accuracy	0,949153
Precision	1
Recall	0,943925

Validating the model yielded an understanding of its reliability and efficacy and underscores the benefits of data-driven approaches in programming education, in the following section, we discuss how these insights can inform targeted instructional improvements and support learners effectively.

Discussion

The data-driven approach in this study culminated in a forecasting model that identified programming concepts that challenge Grade 10 learners across four South African schools. This model opens the door to tailored interventions designed to improve learning outcomes and engagement. The model was tested for its capacity to recognise challenging programming concepts and predict challenges for learners on unseen data by splitting the data into training and testing sets (as seen in Figure 6). Tables 2 and 3 quantify Figure 7's model prediction in the testing set using the evaluation metrics of accuracy, precision and recall. The ratios for accuracy were 0.949153 ≈ 94.9%, precision equalling 1, and recall was 0.943925 ≈ 94.4%. The high precision shows that there were no false positives and that every topic the model identified as challenging was complex. As a result, the recall suggests that the model correctly detected 94.4% of the challenging concepts while missing a minor fraction of FP. With an overall accuracy, the model's predictions appear to be accurate 94.9% of the time.

The ensembled forecasting model in Figure 7 revealed key programming concepts that consistently challenge learners: including conditional statements; problem conceptualisation/solution design; debugging/exception handling; abstraction/pattern recognition; and class/object differentiation. At the Grade 10 level, learners struggle to grasp the numerous linkages and interactions between key concepts that were identified in the forecasting model (Khudhur et al., 2023). For instance, programming requires abstraction and pattern recognition which are associated with cognitive skills that learners may not have fully developed at the outset (Telesko et al., 2023). Consequently, concepts such as sorting algorithms and iteration constructs can be especially challenging since learners are often unfamiliar with these abstract structures. Furthermore, programming is inherently multidisciplinary in demanding skills in both problem conceptualisation and solution design, which novice learners may find challenging to develop (Tikva & Tambouris, 2021).

In programming, the ability to solve problems requires an integration of theoretical understanding and practical skills. Previous studies revealed that relying solely on theoretical knowledge or practical models may not be effective for understanding programming (Telesko et al., 2023). The forecasting model in Figure 7 supports this by showing that learners struggle with key concepts, including syntax, debugging, and exception handling—areas in which both theoretical understanding and practical skills are needed to achieve competency. Simply having programming abilities without a solid theoretical basis can lead to poor program design; this highlights the importance of integrating both aspects. Moreover, programming is a multitasking endeavour that demands concurrent concept development, syntax learning, and code creation (Santos et al., 2022). This multitasking can lead to cognitive overload and stress for learners, in particular, as reflected in low marks and output quality in their programming models (Telesko et al., 2023). When learners do not achieve the desired programming outputs, they risk becoming demotivated and this accentuates the need for instructional strategies that manage cognitive load and sustain engagement (Koorsse et al., 2015).

This study's findings are consistent with those of previous research which indicate that syntax and debugging/exception handling present challenges for many novices, including Grade 10 learners. Such difficulties are often rooted in the inherent complexity of programming languages and the process of error identification and correction. For example, Kadar et al. (2021) identified that common errors among novices include design issues, basic structural errors, and syntax errors such as missing semicolons and curly brackets. Consistent with these findings, this study also observed that learners struggled with syntax errors, including failing to define variables and incorrectly using Boolean statements. This leads them to face challenges in interpreting compiler debug messages, which hinders their ability to independently resolve coding errors.

In addition to identifying challenging programming concepts, the study assessed the code characteristics in association with examination scores. These code characteristics included code length, cyclomatic complexity, code readability, and code constructs that programming teachers can use to tailor instruction and provide targeted interventions to address learning gaps. In Figure 5, a positive correlation was identified between code length and examination scores, with a 9.94-point increase per unit length, suggesting that longer code may reflect thorough problem-solving approaches. However, code length alone does not equate to effective programming since well-structured and concise code can reflect greater understanding and better coding practices (Pelánek & Effenberger, 2022). Teachers must distinguish between lengthy, well-organised code that reflects learners' understanding of programming concepts and verbose solutions that may be ineffective or overly complex. This emphasises the importance of teachers providing learners with coding activities that foster depth of understanding and problem-solving over aiming for longer code (Santos et al., 2022).

Research supports the role of solution length as an indicator of learner competency since it allows teachers to assess coding proficiency and offer targeted support (Woo & Falloon, 2022). In a study by Pelánek and Effenberger (2022), it was found that teachers can establish standards or limitations with greater understanding by analysing the distribution of solution lengths that promote clear and efficient coding procedures. Moreover, their analysis posits that the areas where learners write inefficient code most often are the programming areas that need targeted support including scaffolding, recommendations, or feedback. Fagerlund et al. (2021) similarly found that learners who produced excessively lengthy responses to simple problems often exhibited underlying misconceptions and recommended support for them in programming constructs of loops, conditional statements, and functions. Consequently, in identifying learners who provide lengthy solutions, targeted support may be implemented to assist them in improving their abilities to code using these programming constructs.

We found that cyclomatic complexity was a code characteristic that correlated negatively with examination scores. Cyclomatic complexity quantifies the number of distinct paths in a program's source code to indicate its structural complexity (Hughes, 2021). In other words, it captures the innate complexity of a piece of code by considering components like nesting levels and decision points (if-then-else constructs, loops). Negative cyclomatic complexity often translates to code that is challenging for learners to follow and reason about because of intricate control flow structures and a great number of execution paths, and is more complex to debug since each decision point introduces potential sources of bugs, and this makes the process more time-consuming and error-prone (Hughes, 2021). The regression analysis in Figure 5 resulted in a coefficient ranging from 0 to approximately 43.42 for cyclomatic complexity translating into lower examination scores reflecting these observations. Similarly, in their review of coding and computational thinking, Mills et al. (2024) found that it can be challenging for learners to understand and execute the logic of code with many branches and conditional statements. This can make it challenging for them to complete their programming tasks to achieve the desired learning outcomes. Therefore, learners working with code with negative cyclomatic complexity may score lower owing to the inherent challenge of understanding the logic and debugging errors inside those code structures.

Our findings stress the importance of promoting code that is not only functionally correct but also exhibits positive cyclomatic complexity and high readability since these factors were positively associated with higher examination scores among the learners analysed. Key components contributing to readable code, as shown in the study, include the use of whitespace to organise sections, and the meaningful naming of classes, functions, and variables (Messer et al., 2024). Figure 5 illustrates a coefficient range for code readability from -30.84 to 135.67, suggesting that improvements in code readability correlate with higher examination scores. This suggests that learners are likely to perform well on examinations if they write code that is understandable to themselves and others.

Furthermore, the study showed that despite the significance of reducing cyclomatic complexity through well-structured code which includes the effective use of space, indentation, formatting, layout, line length, and comments (Oliveira et al., 2023), many learners struggled to apply these concepts. As illustrated by Woo and Falloon (2022), computational thinking involves breaking down complex problems into manageable steps, emphasising problem decomposition, abstraction, pattern recognition, and algorithm development. However, the findings revealed that these elements often did not align with learners' problem-solving methods. Learners frequently neglected to define problems in terms of inputs, outputs, and constraints, exhibited a lack of abstraction, failed to recognise patterns, and focused more on code translation than algorithm development. This points to the necessity for targeted instructional methods that will assist novice learners in acquiring the computational thinking skills required for code readability (Messer et al., 2024; Mills et al., 2024; Tikva & Tambouris., 2021).

Studies in programming education have shown that focusing on code quality and readability improves learning outcomes. For example, Karnalim & Simon (2021) developed an automated feedback tool to improve learning code quality by providing code and comment recommendations and concepts. Their approach focuses on uniformity in naming conventions, detecting misspelt words, and assessing the importance of comments. Evaluations of this tool have been beneficial, particularly in reducing human error and ensuring adherence to code quality standards (Messer et al., 2024). Similarly, in this study, code constructs exhibited a positive correlation with examination scores with each unit such as the use of variable names, indentation, appropriate use of control flow structures, and overall code organisation increased 14.56-point rise in examination scores. This corroborates previous findings that the use of these code constructs results in readable code, reduces errors, and improves debugging efficiency (Messer et al., 2024; Oliveira et al., 2023). Thus, concerted efforts to fortify foundational code constructs of challenging programming concepts and improve code readability are key to fostering effective programming skills and achieving desired learning outcomes in challenging programming concepts.

Conclusion

In conclusion, this study leveraged LA to shed light on programming challenges with which Grade 10 learners grapple. We used a data-driven approach to identify programming areas of challenge and learning performance by analysing examination results and code characteristics gathered from code excerpts. Statistical techniques such as variable selection, normalisation, correlation, and regression were employed to identify patterns. The model's generalisation was assessed using a hold-out validation approach that divided the data into training and testing sets. The LA-powered forecasting model identified conditional statements, problem conceptualisation/solution design, debugging/exception handling, abstraction/pattern recognition, and class/object differentiation as programming concepts that correlated with lower examination scores. Beyond identifying challenging programming concepts, the findings revealed code characteristics that were associated with lower examination scores, including code length, readability, and complexity, that indicated learners' difficulty. Teachers can use these findings to provide targeted interventions that result in readable code, reduce errors, and improve debugging efficiency by fostering computational thinking skills and breaking down complex coding structures through problem-solving (Messer et al., 2024; Mills et al., 2024; Telesko et al., 2023).

Despite the study having provided a foundation for understanding programming challenges, we acknowledge its limitations. The model's efficacy is highly dependent on the quality and representativeness of the training data. This study relied on data from 120 learners across four schools, with a primary focus on examination scores and specific code characteristics from code excerpts. Future research could enhance the model's capabilities by incorporating data that captures broader aspects of how learners engage with programming. Furthermore, examining the long-term impact of LA-driven interventions and assessing the scalability of these approaches across varied educational settings would provide a nuanced understanding of their potential to improve programming instruction. The ongoing development and application of LA using comprehensive data can further support teachers in understanding their learners and fostering effective and engaging programming classrooms in South Africa and beyond.

Declaration of interest statement

The authors have no conflicts of interest to declare.

References

Brocker, A., Judel, S., Roepke, R., Mihailovska, N., & Schroeder, U. (2022). Gamifying jupyterlab to encourage continuous interaction in programming education. In *Games and Learning Alliance: 11th International Conference, GALA 2022* (pp. 316–322). Springer Nature. https://doi.org/10.1007/978-3-031-22124-8_32

Campbell, J. P., & Oblinger, D. G. (2007). Academic analytics: A new tool for a new era. *EDUCAUSE Review*, *42*(4), 40.

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*, 1–14. https://doi.org/10.30935/cedtech/8247

Choi, W. C., Lam, C. T., & Mendes, A. J. (2023). A systematic literature review on performance prediction in learning programming using educational data mining. *In 2023 IEEE Frontiers in Education Conference (FIE)* (pp. 1–9). Texas, USA. https://doi.org/10.1109/fie58773.2023.10343346

Creswell, J. W., & Plano Clark, V. L. (2018). *Designing and conducting mixed methods research* (3rd ed.). SAGE.

Department of Basic Education. (2011). *Curriculum and Assessment Policy Statement Grades 10-12 Information Technology.* https://www.Department+of+Basic+Education.+(2011).+Curriculum+and+Assessment+Policy+Statement+Grades+10-12+Information+Technology

Department of Basic Education. (2021). *Proposed amendments to the curriculum and assessment policy statement (CAPS) to make provision for coding and robotics Grades R-9. https://www. Department+of+Basic+Education.+(2021).+Proposed+amendments+to+the+curriculum+and+assessment+policy+statement+(CAPS)+to+make+provision+for+coding+and+robotics+Grades+R-9.&rlz*

Dlamini, R., & Dewa, A. (2022). Unplugged teaching: Deepening information technology learning. *Open Journal of Social Sciences*, *10*(4), 476–486. https://doi.org/10.4236/jss.2022.104034

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

Hellings, J., & Haelermans, C. (2022). The effect of providing learning analytics on student behaviour and performance in programming: a randomised controlled experiment. *Higher Education*, *83*(1), 1–18. https://doi.org/10.1007/s10734-020-00560-z

Hughes, E. (2021). Comprehensive review: Key metrics in defect prediction models. *Journal of Science & Technology*, *2*(5), 83–92. https://thesciencebrigade.com/jst/article/view/56

Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Börstler, J., Edwards, S. H., & Toll, D. (2015). *Educational data mining and learning analytics in programming: Literature review and case studies*. Proceedings of the 2015 ITiCSE on working group reports, 41–63. https://doi.org/10.1145/2858796.2858798

Janse van Vuuren, E. C. (2020). Development of a contextualised data analytics framework in South African higher education: Evolvement of teacher (teaching) analytics as an indispensable component. *South African Journal of Higher Education*, *34*(1), 137–157. https://hdl.handle.net/10520/EJC-1e1a0881b0

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. http://dx.doi.org/10.6007/IJARPED/v10-i3/11100

Karnalim, O., & Simon. (2021). Promoting code quality via automated feedback on student submissions. *In 2021 IEEE Frontiers in Education Conference (FIE)* (pp. 1–5). Lincoln, NE. https://doi.org/10.1109/FIE49875.2021.9637193

Khudhur, N. F. (2023). Conceptual level comprehension support of the object-oriented programming source-code using kit-build concept map. *International Journal of Information and Education Technology*, *13* (12), 1858–1867. https://doi.org/ 10.18178/ijiet.2023.13.12.1999

Koorsse, M., Cilliers, C., & Calitz, A. (2015). Programming assistance tools to support the learning of IT programming in South African secondary schools. *Computers & Education*, *82*, 162–178. https://doi.org/10.1016/j.compedu.2014.11.020

Kroustalli, C., & Xinogalos, S. (2021). Studying the effects of teaching programming to lower secondary school students with a serious game: A case study with Python and CodeCombat. *Education and Information Technologies*, *26*(5), 6069–6095. https://doi.org/10.1007/s10639-021-10596-y

Lee, H. Y., Lin, C. J., Wang, W. S., Chang, W. C., & Huang, Y. M. (2023). Precision education via timely intervention in K–12 computer programming course to enhance programming skill and affective-domain learning objectives. *International Journal of STEM Education*, *10*(1), *52*, 1–9. https://doi.org/10.1186/s40594-023-00444-5

López-Pernas, S., Saqr, M., & Viberg, O. (2021). Putting it all together: Combining learning analytics methods and data sources to understand students' approaches to learning programming. *Sustainability*, *13*(9), *4825.*, 1–18. https://doi.org/10.3390/su13094825

Marimuthu, M., & Govender, P. (2018). Perceptions of scratch programming among secondary school students in KwaZulu-Natal, South Africa. *The African Journal of Information and Communication*, *21*, 51–80. http://dx.doi.org/10.23962/10539/26112

Messer, M. B. (2024). Automated grading and feedback tools for programming education: A systematic review. *ACM Transactions on Computing Education*, *24*(1), 1–43. https://doi.org/10.48550/arXiv.2306.11722

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*, *0*(0), 1–38. https://doi.org/10.3102/00346543241241327

Msweli, N. T., Mawela, T., & Twinomurinzi, H. (2023). Data science education-A scoping review. *Journal of Information Technology Education*, *22*, 263–294. https://doi.org/10.28945/5173

Oliveira, D., Santos, R., Madeiral, F., Masuhara, H., & Castor, F. (2023). A systematic literature review on the impact of formatting elements on code legibility. *Journal of Systems and Software*, *111728*, 1–17. https://doi.org/10.1016/j.jss.2023.111728

Omer, U., Tehseen, R., Farooq, M. S., & Abid, A. (2023). Learning analytics in programming courses: Review and implications. *Education and Information Technologies*, *28*(9), 11221–11268. https://doi.org/10.1007/s10639-023-11611-0

Pelánek, R., & Effenberger, T. (2022). Design and analysis of microworlds and puzzles for block-based programming. *Computer Science Education*, *32*(1), 66–104. https://doi.org/10.1080/08993408.2020.1832813

Pereira, F. D., Oliveira, E. H., Oliveira, D. B., Cristea, A. I., Carvalho, L. S., Fonseca, S. C., & Isotani, S. (2020). Using learning analytics in the Amazonas: Understanding students' behaviour in introductory programming. *British Journal of Educational Technology*, *51*(4), 955–972. https://doi.org/10.1111/bjet.12953

Prinsloo, P., & Kaliisa, R. (2022). Learning analytics on the African continent: An emerging research focus and practice. *Journal of Learning Analytics*, *9*(2), 218–235. https://doi.org/10.18608/jla.2022.7539

Santos, J. S., Andrade, W. L., Brunet, J., & Melo, M. R. A. (2022). A Systematic Literature Review on Predictive Cognitive Skills in Novice Programming. *In 2022 IEEE Frontiers in Education Conference (FIE)* (pp. 1–9). Uppsala, Sweden: https://doi.org/10.1109/FIE56618.2022.9962582

Sun, L., Guo, Z., & Zhou, D. (2022). Developing K-12 students' programming ability: A systematic literature review. *Education and Information Technologies*, *27*(5), 7059–7097. https://doi.org/10.1007/s10639-022-10891-2

Telesko, R., Spahic-Bogdanovic, M., Hinkelmann, K., & Pande, C. (2023). A new approach for teaching programming: Model-based Agile Programming (MBAD). *In Proceedings of the 2023 8th International Conference on Information and Education Innovations* (pp. 13–18). Association for Computing Machinery. https://doi.org/10.1145/3594441.3594445

Tikva, C., & Tambouris, E. (2021). Mapping computational thinking through programming in K-12 education: A conceptual model based on a systematic literature review. *Computers & Education*, *162, 104083*, 1–23. https://doi.org/10.1016/j.compedu.2020.104083

Utamachant, P., Anutariya, C., & Pongnumkul, S. (2023). i-Ntervene: applying an evidence-based learning analytics intervention to support computer programming instruction. *Smart Learning Environments*, *10*(1)*, 37*, 1–30. https://doi.org/10.1186/s40561-023-00257-7

Veerasamy, A. K., Laakso, M. J., & D'Souza, D. (2022). Formative assessment tasks as indicators of student engagement for predicting at-risk students in programming courses. *Informatics in Education*, *21*(2), 375–393. https://doi.org/10.15388/infedu.2022.15

Woo, K., & Falloon, G. (2022). Problem solved, but how? An exploratory study into students' problem solving processes in creative coding tasks. *Thinking Skills and Creativity*, *46, 101193*, 1–16. https://doi.org/10.1016/j.tsc.2022.101193