OPEN ACCESS

Manuscript ID:
EDU-2025-14019542

Volume: 14

Issue: 1

Month: December

Year: 2025

P-ISSN: 2320-2653

E-ISSN: 2582-1334

Received: 26.09.2025

Accepted: 01.11.2025

Published Online: 01.12.2025

Citation:
Cuniah, C., Panchoo, S., & Jaillet, A. (2025). Error-Grid Framework: A Pedagogical Approach for Diagnosing Programming Mistakes and Enhancing Learning Outcomes in Secondary Education. *Shanlax International Journal of Education, 14*(1), 49–58.

DOI:
https://doi.org/10.34293/education.v14i1.9542

Error-Grid Framework: A Pedagogical Approach for Diagnosing Programming Mistakes and Enhancing Learning Outcomes in Secondary Education

Canayah Cuniah
University of Technology, Mauritius
https://orcid.org/0000-0003-4764-3946

Shireen Panchoo
University of Technology, Mauritius

Alain Jaillet
University of Cergy-Pontoise, France

Abstract
Programming in secondary schools is still a difficulty because learners are transitioning to text-based languages; this is the case of programming. The dropping pass rates in Mauritius are indicative of endemic conceptual and procedural discontinuities. This study constructs and tests the Error-Grid Framework, a low-cost classroom intervention that uses errors as a kind of diagnostic message to inform specific instruction. Content analysis of 30 Grade 11 students (n=90 handwritten scripts; 2,987 lines) was conducted, providing a code of recurring errors in 11 concepts of core programming. Reliability was studied among four educators using Krippendorff's alpha, and effectiveness was studied with the help of a pre/post design with two groups of students (n=11; n=13). Classroom utility was measured using survey data from 39 teachers across 20 schools. The findings indicated moderate inter-rater reliability (1=0.67; 2=0.64), substantial error decreases (p<0.001), and continued difficulties in loops, arrays, and functions. Teachers showed dense usability and diagnostic worthiness in the study. The framework facilitates differentiated teaching and timely feedback. In the future, this work should be extended to other languages, where automation is introduced to provide scalability and investigate long-term effects.
Keywords: Programming Education, Error Analysis, Secondary Education, Diagnostic Tools, Teacher Support, Computational Thinking

Introduction

This is because programming has been a marketable skill in the current technology-oriented society to achieve a career in computing, automation, data processing, and problem-solving. At the secondary level, it is assumed that students acquire cognitive thinking skills and logical reasoning, although learning and teaching are difficult. Novices are forced to learn abstract concepts, bizarre grammar, and complex thinking, which often leads to frustration and ineffective performance (Izu & Mirolo, 2024). Programming exams have a declining pass rate in Mauritius, suggesting knowledge and teaching gaps. Other languages, such as Scratch and Alice, have made entry more accessible by simplifying syntax at the expense of seldom helping to eliminate conceptual mysticism that can occur when using text-based languages. Beginners have the most difficulty with compilers that identify syntax errors but not logical errors. Teachers also lack regularised ways of studying errors and organising particular interventions, which leads to an increase in the number of mistakes and the development of a lack of confidence (Demirdag, 2015). While more emphasis is placed on learning

through errors, few tools allow teachers to systematically record the programming errors committed in classrooms, interpolate them, and act on them. This is particularly evident in Mauritius, where programming is taught primarily through handwritten scripts. This need is addressed by the present study by developing and evaluating The Error-Grid Framework, a simple and inexpensive framework that allows teachers to transform the errors made by the students into diagnostic data and helps learners reflect on reasoning.

The Error-Grid Framework described in this paper is a framework; it is a paper-based diagnostic tool that could be utilised by teachers to map the errors of the students within the framework of the major concepts of programming and allow them to focus on the aspects of support and think about the common misconceptions. One attempt to fill this gap is the Error-Graph Framework, which offers a cost-effective, accessible, and reliable method of obtaining a visual depiction of tendencies and taking action against errors made by teachers.

The research question that will be considered as the focal one and used in the research will be as follows: How effective can the Error-Grid Framework be applied to the detection and avoidance of programming errors among secondary school learners, and how do educators perceive the potential of the framework to be implemented in classes and whether they find it valuable and useful in their teaching process.

Two hypotheses are proposed.

- $\mathbf{H_1}$: The use of the Error-Grid Framework will result in a statistically significant reduction in the number of program errors.
- $\mathbf{H_2}$: Educators will report that they have positive attitudes towards the efficacy, usability, and applicability of the framework in various classroom activities.

The study has the following three objectives:

- To determine and categorise the general programmer errors committed by students in a systematic manner using the Error Grid Framework.
- To establish the consistency of the framework among the various educators regarding inter-rater consistency.
- To determine the effectiveness of the framework in eliminating mistakes and helping the student understand the information with the assistance of pre-grid analysis and post-grid analysis.

This study is intended to be an initial study. It is founded on a classroom style and has a small number of participants, and is primarily aimed at Visual Basic as a target language. Therefore, the findings are provisional and not conclusive for generalised use. Nevertheless, it also contributes to the literature because it demonstrates that the implementation of error-based pedagogy can be reproduced in the realm of programming, as well as to teachers who have limited opportunities most of the time.

It is also particularly small-scale: it is not intended to cover all the problems in the sphere of programming education but to verify whether a systematic diagnostic tool might contribute to raising the emphasis of the teaching strategy and the learning outcomes. The evidence generated provides a basis for additional expansions, such as into other languages, larger cohorts, and the addition of digital or AI technology to offer scalability.

In general, this introduction preconditions the context of the justification of the creation of the Error-Grid Framework, defines its objectives, and locates it in the framework of more comprehensive discussions about the pedagogy of programming. This framework will transform programming education into a stronger, more reflective, and effective learning experience by applying mistakes as helpful information rather than as a loss.

Literature Review

Programming Education and Learning Challenges

Although teaching programming in secondary schools has always been reported to be difficult, the fact that it involves abstract thinking in addition to technical accuracy makes it a challenge. Novices must learn syntax, program structure, and data structure in addition to familiarising themselves with problem-solving and design. Unlike other topics that can be easily learned through memorisation, programming requires the development of a logic sequence of instructions, a task that consumes working memory and requires higher-order thinking. Researchers

and educators have determined that misconceptions are one of the major obstacles in research studies: students are inclined to misunderstand loop execution, stateful variables, and conditional execution. This may lead to frustration, loss of motivation, and poor performance, a trend which has been witnessed in Mauritius over the past years, where the pass rate in programming has decreased.

According to researchers, these challenges are clarified by several factors: poor scaffolding, poorly adjusted pedagogies, and the professionalisation of teachers themselves. Visual languages like Scratch and Alice have fewer barriers to entry; however, they do not prepare text-based languages among learners. Thus, the shift to textual programming, in contrast to block-based tools, may be followed by immensely steep learning curves and conceptual errors that cannot be detected immediately (Larrain and Kaiser, 2022).

Programming Errors as Learning Opportunities

The alternative perspective is error analysis which does not judge mistakes as failures but as diagnostic features of student thinking. According to pedagogical research, through a systematic investigation of misconceptions, one can determine the mental models of students and then guide them on corrective teaching methods. In computer programming, an error in programming, such as an undeclared variable, a loop, or a logical misunderstanding, is found everywhere, regardless of the country of operation, be it Europe, Asia, or Africa. The identification of these patterns will help teachers identify the areas in which students are expected to perform poorly and develop specific remedies (Hadjerrouit, 1999; Ihantola & Kihn, 2011; Demirdag, 2015).

Research has also shown that pedagogy rooted in errors promotes reflection and resilience. By analyzing their errors, there is a higher chance that students will narrow down their arguments and build long-term knowledge. For teachers, error analysis provides first-hand insight into how students think and where modifications in teaching are required. Despite its potential, error analysis has not found widespread use in classroom practice due to time limitations, lack of structured tools, and use of compiler feedback which focuses on syntax and not on logic.

Current Evaluation Methods in Programming Education

Assessment techniques determine the effectiveness of students approach to programming. Compilers, syntax checkers, and online judges are all real-time feedback systems that make it easy to correct and facilitate the process of learning. However normally , they mark superficial errors without engaging with the underlying conceptual errors which are the root causes of superficial errors. More cognitively informative delayed feedback approaches, such as teacher script review and post-task reflection, are also labour-intensive and impossible to scale (Kaufmann et al., 2023).

Grover and Pea (2018) in their turn suggested a systematic study of programming errors, particularly in the situation of transitioning to the visual text environment, where errors are more abstract and conceptually challenging. Yoshizawa and Watanobe (2018) created automated methods for identifying logical errors, such as rule-based classifiers or structure-pattern matching algorithms. These tools seek to expose falsehoods in real time, although it can be challenging to supply them with the technical resources needed to support them, and they are not readily scalable to resource-constrained classrooms (Kim and Lee, 2024). Meanwhile, Bloom's taxonomy still informs the structure of programming activities, as verbal learning assessments progress through memorising syntax, synthesising algorithms, and so forth. Although Bloom's framework is successful in organising assessment, it does not provide a way to record or analyse student errors.

Gaps in the Literature

However, gaps remain in the literature. First, compilers and automated systems are feedback providers but seldom attract the scope of logical and conceptual fallacies committed by novices. Second, current studies in the field of programming pedagogy are more likely to focus on environmental or curriculum changes rather than classroom aids. Third, few articles focus on systematic and accessible frameworks, especially where digital

resources are scarce (Lobanov et al., 2021). This means that teachers are left without any sure means of categorising the errors, discovering patterns, and converting the same to actionable information.

Towards an Error-Based Pedagogical Framework

These loopholes indicate the necessity of a systematic, classroom-prepared method that would render students' mistakes visible and useful. This type of framework would help diagnose and intervene by making mistakes the central pieces of information. It would enable teachers to see trends within a cohort and recognise chronic IA problem areas, such as loops or arrays, and plan differentiated instruction. This would make students think about errors and solve problems adaptively.

The Error-Grid Framework addresses this requirement by providing a low-cost paper-based framework for systematically recording programming errors over core concepts. It is not new to automation but offers teachers a very simple and yet structured diagnostic tool that does not disrupt the established practice. It satisfies the theoretical and practical requirements of programming education by bridging the gap between research on error-based pedagogy and the realities of classroom settings in secondary schools.

Methodology

Study Design and Rationale

To test the feasibility and usefulness of the Error-Grid Framework in secondary schools, this study used a mixed-methods approach. The data sources were 30 Grade 11 students enrolled in a Computer Science course at two state secondary schools in Mauritius. The students were given three curriculum-based programming tasks, and each was required to create 90 handwritten programming scripts (2987 lines of Visual Basic codes).

The small size of the cohorts and the scheduling demands of Mauritian secondary schools rendered the use of a control group impossible. Rather, a pre- and post-design was selected to compare the patterns of errors prior to and after the explicit use of the framework in the classroom setting. These types of designs are used when an exploratory study is conducted in educational research, and random allocation is not feasible, yet internal validity is of concern (Kaufmann et al., 2023).

Participants

The participants were 30 Grade 11 (School Certificate) students (16-17 years old; 16 boys and 14 girls) studying Computer Science in two state secondary schools in Mauritius. The schools are based on the Cambridge International Education (CIE) computer science curriculum and equip students for the Higher School Certificate (HSC) examinations. Participants were identified because of the transition stage between visual technology and block-based technology to text-based programming in Visual Basic. It is always observed that false impressions are likely to be raised at this stage. The students had taken at least one year of introductory computing in Grade 10 and had not yet been taught systematically to analyse and categorise programming errors. The involvement was voluntary and endorsed by the school administration, and parental consent was obtained as per institutional ethics.

Data Collection and Procedures

The study unfolded across four linked stages:

Script Analysis: A total of 2,987 lines of code (90 scripts) were analysed using content analysis to determine common mistakes. These were organised into 11 programming concepts: variables, conditionals, loops, arrays, and functions. The tests involved students writing some programs, such as a grade calculator, a multiplication table generator, and a menu-based system, which is similar to standard examination conditions. All programming was manually coded according to the national examination patterns. This strategy did not use auto-completion of the compiler or even the automatic display of errors, thus depicting a better picture of the areas in which students are prone to have trouble.

Framework Development: The types of errors were divided into a grid with a structured colour-coded grid that visualised the error against its concepts. This constituted the Error-Grid Framework which could be utilised by teachers to identify patterns at both the class and individual levels.

Test Reliability: Four teachers tested the framework using the same sample scripts.

Krippendorff alpha was used to compare their classifications to determine consistency.

Experimental Study: Two groups of students were allocated programming tasks prior to and after introducing the framework. The number of errors was documented and compared using paired t-tests to determine whether exposure to the error grid decreased mistakes.

Educator Survey: Thirty-nine teachers in 20 schools were given a questionnaire that discussed their views on the tool's usefulness, effectiveness, and classroom usefulness. Open-ended responses were coded thematically, whereas closed questions were coded descriptively.

Error Coding and Reliability

This study combined both qualitative and quantitative analyses to assess the framework. In the initial procedure, we conducted a content analysis of the handwritten programming of the students to discover and list common program errors. Based on this analysis, a diagnostic grid was created which categorised errors into 11 fundamental programming concepts. The instructional design used to test whether the use of the grid minimised student errors and changes was statistically significant pre-grid/ post-grid.

In a test of consistency, the grid was applied to a sample of scripts by four seasoned educators operating independently, and inter-rater agreement was determined. The values obtained were within the acceptable levels of reliability for educational instruments. Finally, a questionnaire was created to collect the teachers' perceptions regarding the usefulness of the grid in the classroom, application ease, and value as a diagnostic tool.

Effectiveness Testing

The contribution of the Error-Grid Framework to the performance of the students was analysed with the help of paired samples t-tests which were conducted to compare the number of errors prior to and after the introduction of the grid.

Table 1 Pre and Post Grid Error Counts and Effect Sizes

Group	N	Pre-mean (SD)	Post-mean (SD)	t(df)	p	Cohen's d
1	11	17.54 (3.12)	8.36 (2.87)	9.12 (10)	<.001	1.22
2	13	11.53 (2.65)	3.54 (1.97)	10.05 (12)	<.001	1.39

These findings indicate that programming error reductions of practical importance (not just statistically significant differences) were obtained through exposure to the grid. The magnitude of the effect indicates that despite the lack of a control group, the framework has the definite possibility of enhancing classroom results in the future.

Teacher Survey

On behalf of the error-grid framework, we invited 39 computer science teachers from 20 Mauritian secondary schools to share their experiences with the error-grid framework. The survey combined closed-ended questions on a Likert scale (five points, from strongly disagree to strongly agree) with open-ended questions. The rating items included questions regarding how easy the tool was to use, the clarity of the categories of errors, appropriateness to current instructional practices, and the general value of instruction. Open questions helped the teachers explain how the grid was applied in their classrooms, what advantages or disadvantages they observed, and how it could be improved.

The Likert answers were analysed descriptively (frequencies and percentages) to determine overall acceptance. The open remarks were thematically codified to appear as common themes such as usability, diagnostic clarity, and recommendations on digital or AI-based extensions. Such a combination of numerical tendencies and narrations by teachers provided a ground-based image of how the grid worked in the classroom and what priorities were to be followed for further development.

Ethical Considerations

All participants were informed of the purpose of the research and signed an informed consent form prior to participating in the research. The students' scripts were anonymised by replacing their names

with codes. The teachers' involvement was on a voluntary basis, and the responses were handled privately. The research was conducted with the approval of the University of Technology, Mauritius, and in line with the ethics of carrying out research in education in the country.

Results

Categorisation of Programming Errors

The 90 student scripts (2,987 lines of code) were analysed, and 11 programming concepts had recurring errors. The error grid framework is an error-coding system. The most common problems were encountered in arrays, loop constructs, and functions/procedures, with variables and data types becoming less problematic. This trend shows that students usually acquire the ability to learn simple syntax without any problems with high-level logic and abstraction.

The errors are categorised in Table II, Appendix A. Complex constructs were shown to create disproportionate errors, which is why specialised instructional support is necessary, as confirmed by visualisations.

Application of the Error-Grid Framework

Student scripts annotated by students were subjected to the Error Grid, generating binary data on the presence or absence of each type of error. The process visualised the error trends at the class and personal levels. The colour-coded structure (red means errors, green means corrections) allowed for quick diagnosis in the classroom setting.

Figures 1 and 2 of error frequencies with bar charts show that concepts of conditional statements (85%), loops (85%), arrays (89%), Functions and Procedures (85%) continue to create the largest portion of errors, with the input/output and data types being less troublesome. Such visual products justified the usefulness of the grid in clarifying student problems and provided evidence to teachers concerning differentiated instruction.

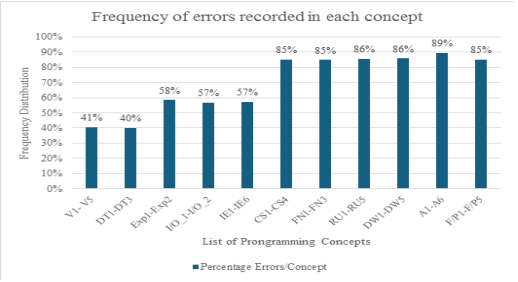

Figure 1 Frequency of Programming Errors Recorded per Concepts

Implications for Teaching

Findings point to several pedagogical insights:

- More specialised instructional design is needed for higher-level subjects, such as arrays and control flow. These areas can be consolidated by using structured practice and activities that have scaffolds.
- Reflection is improved through error awareness. The framework promotes metacognition and remedial measures among students by demonstrating the location of their mistakes.
- Teacher planning benefits. The grid provides an educator with a systematic understanding so that the lesson can be amended based on the actual performance of the students as opposed to speculations.

Reliability of the Framework

To determine consistency, four teachers were asked to use the Error-Grid independently on a set of scripts. The inter-rater agreement calculated by Krippendorff's alpha was 0.67 in total and 0.64 in categories, indicating moderate reliability. Although imperfect, these values indicate that the framework can be used to provide reproducible results among teachers, which contributes to its use as a diagnostic tool.

Effectiveness: Pre- and Post-Grid Analysis

The pre- and post-tests were conducted on two groups of students (n=11 and n=13). The findings showed a statistically significant decrease in errors.

- Group 1: The mean errors decreased from 17.54 to 8.36.
- Group 2: The mean errors decreased from 11.53 to 3.54.

These means were found to be significant ($p < 0.001$) by paired-samples t-tests that reported not only the presence of errors but also an improvement in these errors when the framework was used.

Educator Perceptions

The answers to 39 surveys from 20 schools provided insight into classroom usage. The usability of the grid was found to be high, with the teachers liking its simplicity and compatibility with current practices. They emphasised its diagnostic quality, pointing out that it determined recurring misconceptions better than compiler feedback alone. Its pedagogical value as a lesson planner, scaffold, and formative assessment has been valued by many. Some improvements have been proposed, such as combining error classes and creating electronic or automated systems to work effectively. The open-ended remarks emphasised the importance of the tool in resource-constrained environments where a digital feedback system is nonexistent.

Summary of Results

The Error-Grid Framework was effective in leading to the classification of common programming errors, visualisation of error patterns, and teacher diagnosis and student reflection. The framework had moderate inter-rater agreement, as revealed by reliability analysis and statistically significant error reduction after the experimental study. The usefulness of the teacher surveys and their classroom relevance were further justified, and a means to improve them was identified. Collectively, these findings indicate that the framework offers a repeatable, available, and powerful channel for enhancing programming learning in high schools.

Discussion and Critical Evaluation

Interpretation of Findings

This study demonstrates that the Error-Grid Framework can serve as a low-cost, practical diagnostic instrument for secondary programming learning. The grid makes the areas where students have the most difficulties apparent through the systematic classification of errors into 11 programming concepts. The findings we have are in line with global trends: students tend to master simple syntax, such as variables and data types, but continue to struggle with control flow, arrays, and functions, which involve more abstract thinking and multi-step reasoning.

The best aspect of the framework is its ability to transform these challenges into practical teaching considerations. According to teacher reports, the grid assisted them in identifying misconceptions that were not considered by the compilers and used actual student performance to shape the lesson and not speculation. Students, in turn, became more conscious of common mistakes and more introspective about themselves in terms of problem-solving. The inter-rater reliability was moderate (Krippendorffs 0.67 0.64), indicating that the grid can be used with a reasonable degree of consistency by different teachers.

This picture is reinforced by the pre/post analysis. The error decreases in both groups of learners, large, statistically significant, 17.54 to 8.36 ($d = 1.22$) in Group 1 and 11.53 to 3.54 ($d = 1.39$) in Group 2. Such large effect sizes indicate that the impact of the tool on a classroom should not be considered merely reliable statistically, but meaningful education-wise, as it assists novices in producing a cleaner and more accurate piece of code.

Pedagogical Implications

There are a number of practical implications that can be made to classroom practice:

- Attack higher-level subject areas: Loops, arrays, and control flow are significant pitfalls; therefore, structured practice and scaffolding of these areas should be taught.
- Promoting error awareness: Demonstrating to students the location of their errors promotes metacognitive and resilient learning.
- Evidence-based planning: The grid provides teachers with a straightforward non-tech map of misconceptions among students, which enables them to modify lessons and feedback on the spot.
- Accessible innovation: Because the tool is paper-based, it can be applied in resource-constrained situations where technology to provide automated feedback cannot be used.

Limitations

The current study was limited to Visual Basic and therefore may not be applicable to other languages, such as Python or C++, which present other challenges. The Error-Grid is a manual tool, which would involve teachers marking every script separately, which is manageable in a small classroom but requires automation in a large one, and would leverage scaling feedback. The existing grid also focuses on syntax and logic, which might ignore more underlying conceptual misconceptions. Moreover, the study did not include a control group or prolonged follow-up to make assertions about enduring learning benefits. Lastly, teacher training was not assessed, and structured guidance would enhance uniform usage.

Future Directions

Further research in this direction should be conducted as follows:

- Other languages, such as Python and Java, should be tested to enhance generalisability.
- The addition of error classification based on automation or AI can provide the tool with a larger scale for classrooms and real-time feedback.
- Expanding the number of categories of errors used to represent more conceptual misconceptions and the mental models of students, not just syntax and logic.
- Longitudinal and controlled research studies should be conducted to determine retention and transfer learning results in the long term.
- Teacher training materials and professional learning modules should be developed to ensure that the framework is used consistently and effectively in various school contexts.

Summary

The Error-Grid Framework is an effective, reliable, and practical method for diagnosing programming errors in secondary schools. It fills the gap between the pedagogy of errors and the realities of the classroom by providing teachers with a systematic approach to the interpretation of mistakes and students with a way to reflect and improve. Although constraints are still present, especially in terms of breadth, scalability, and extended validation, the results imply that the framework has a high possibility of being used to make programming education more resilient and reflective.

Conclusion

This research appraised the Error-Grid Framework as a viable instrument for planning and minimising errors in programming in secondary schools. It mapped errors on 11 core concepts and revealed that novices can cope with simple syntax but cannot cope with loops, arrays, and functions. The tool assisted students in minimising mistakes in pre/post testing and provided teachers with a very cheap solution to visualise trends and focus instruction. Inter-rater reliability demonstrates uniformity in the classroom. The drawbacks of this study are the use of one language, small sample size, and manual coding. Future research should be expanded to other languages, make feedback more automated, and offer training to teachers.

References

Demirdag, S. (2015). Management of Errors in Classrooms: Student Mistakes and Teachers. *International Journal of Humanities and Social Science, 5*(7), 77-83.

Grover, S., & Pea, R. (2018). Computational Thinking: A Competency Whose Time has Come. In *Computer Science Education: Perspectives on Teaching and Learning in School*. Bloomsbury Publishing.

Hadjerrouit, S. (1999). A constructivist approach to object-oriented design and programming. *Proceedings of the 4th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education*.

Ihantola, E. M., & Kihn, L. (2011). Threats to validity and reliability in mixed methods accounting research. *Qualitative Research in Accounting & Management, 8*(1), 39-58.

Izu, C., & Mirolo, C. (2024). Towards comprehensive assessment of code quality at CS1-level: Tools, rubrics and refactoring rules. In *2024 IEEE Global Engineering Education Conference (EDUCON)*.

Kaufmann, O. T., Larsson, M., & Ryve, A. (2023). Teachers' error-handling practices within

and across lesson phases in the mathematics classroom. *International Journal of Science and Mathematics Education, 21*, 1289-1314.

Kim, D. Y., & Lee, W. J. (2024). Predicting rough error causes in novice programmers using cognitive level. In *Generative Intelligence and Intelligent Tutoring Systems*, Springer.

Larrain, M., & Kaiser, G. (2022). Interpretation of Students' Errors as Part of the Diagnostics Competence of Pre-service Primary School Teachers. *Journal for Mathematics Didactics, 43*, 39-65.

Lobanov, A., Bryksin, T., & Shpilman, A. (2021). Automatic Classification of Error Types in Solutions to Programming Assignments at Online Learning Platform. *arXiv*.

Yoshizawa, Y., & Watanobe, Y. (2018). Logic Error Detection Algorithm for Novice Programmers Based on Structure Pattern and Error Degree. In *2018 9th International Conference on Awareness Science and Technology (iCAST)*.

Appendix A

An Extract of the Error-Grid framework reflects the presence or absence of each error per student submission, with '1' indicating that the error occurred at least once, and '0' indicating it was not present. This binary approach helps identify which concepts are most frequently affected but does not reflect how many times each error type occurred.

The Table 1 presents a comparative analysis of programming errors made by Students of Group-1 before and after using the Error-Grid Framework. It categorises errors based on reference codes, programming concepts, and descriptions, displaying the frequency of each mistake for individual students in both pre-grid and post-grid phases. The results highlight a reduction in errors after using the framework, demonstrating its effectiveness in helping learners identify and correct common programming mistakes.

Table 2 An Extract of the Error-Grid Framework with Recorded Data

			Students group 1 pre-grid											
Ref Code	**Concepts**	**Description**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**No of Errors**
V1	Variable	No Initialisation of variable	0	1	1	0	1	0	0	0	1	1	1	5
V2	Variable	Use of reserved keyword as variable	0	1	1	0	1	1	0	0	1	1	1	4
V3	Variable	Wrong spelling of Keywords	0	1	1	0	1	0	0	0	1	1	0	6
V4	Variable	Space between two words	0	0	1	0	1	0	0	0	1	1	0	7
V5	Variable Declaration	Keyword 'Dim' is missing or wrongly written	0	1	1	0	1	0	0	0	1	1	0	6
DT1	Data Type	Data type wrongly written: e.g. Dooble	0	1	1	0	1	1	0	0	1	1	0	5
DT2	Data Type	Irrelevant use of data type	0	0	1	0	1	1	0	0	1	1	0	6
DT3	Use of constant keyword	The keyword 'const' is missing or wrongly written	0	1	1	0	1	1	1	0	1	1	0	4
Exp1	Expression	Expression wrongly written/ Irrelevant	0	0	1	0	1	0	0	0	1	1	0	7
Exp2	Expression	Assignment operator ' =' missing	0	0	1	0	1	1	1	1	1	1	0	4
I/O_1	Output Statement	Keywords 'console.writeline' wrongly written	0	0	1	0	1	1	1	1	1	1	0	4
I/O_2	Input Statement	Keywords 'console.readline' wrongly written	0	0	1	0	1	1	1	1	1	1	0	4
IE1	If-Else	'Else' written before 'If'	0	0	1	0	1	1	1	0	1	1	0	5

IE2	If-Else	Condition missing	0	0	1	0	1	1	0	0	1	1	0	6
IE3	If-Else	Curly braces omitted	0	0	1	0	1	0	0	0	1	1	0	7
IE4	If-Else	Wrongly placed statements	0	0	1	0	1	1	0	0	1	1	0	6
IE5	If-Else	Irrelevant logical expression	0	0	1	0	0	1	0	0	1	1	0	7
IE6	End-If	Missing Endif	0	0	1	1	1	1	1	0	1	1	0	4

			Students group 1 post-grid											
Ref Code	**Concepts**	**Description**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**No of Errors**
V1	Variable	No Initialisation of variable	1	1	1	1	1	1	1	0	1	1	0	2
V2	Variable	Use of reserved keyword as variable	1	1	1	1	1	0	1	0	1	1	0	3
V3	Variable	Wrong spelling of Keywords	1	1	1	0	1	0	1	0	1	0	0	5
V4	Variable	Space between two words	1	1	1	1	1	1	1	1	1	1	1	0
V5	Variable Declaration	Keyword ‘Dim’ is missing or wrongly written	1	1	1	1	1	1	1	0	1	1	0	2
DT1	Data Type	Data type wrongly written: e.g. Dooble	1	1	1	0	1	0	1	0	1	1	0	4
DT2	Data Type	Irrelevant use of data type	1	1	1	0	1	0	1	0	1	1	0	4
DT3	Use of constant keyword	The keyword ‘const’ is missing or wrongly written	1	1	1	1	1	1	1	0	1	1	0	2
Exp1	Expression	Expression wrongly written/ Irrelevant	1	1	1	0	1	0	1	0	1	0	0	5
Exp2	Expression	Assignment operator ‘ =’ missing	1	1	1	1	1	0	1	0	1	1	0	3
I/O_1	Output Statement	Keywords ‘console.writeline’ wrongly written	1	1	1	1	1	1	1	1	1	1	0	1
I/O_2	Input Statement	Keywords ‘console.readline’ wrongly written	1	1	1	1	1	1	1	0	1	1	1	1
IE1	If-Else	‘Else’ written before ‘If’	1	1	1	1	1	0	1	0	1	1	0	3
IE2	If-Else	Condition missing	1	1	1	1	1	0	1	1	1	1	1	1
IE3	If-Else	Curly braces omitted	1	1	1	1	1	1	1	1	1	1	1	0
IE4	If-Else	Wrongly placed statements	1	1	1	1	1	1	1	1	1	1	1	0
IE5	If-Else	Irrelevant logical expression	1	1	1	1	1	0	1	0	1	1	1	2
IE6	End-If	Missing Endif	1	1	1	1	1	1	1	0	1	1	1	1

Author Details

Canayah Cuniah, *University of Technology, Mauritius, **Email ID**: canayah.cuniah@gmail.com*

Shireen Panchoo, *University of Technology, Mauritius, **Email ID**: s.panchoo@utm.ac.mu*

Alain Jaillet, *University of Cergy-Pontoise, France, **Email ID**: alain.jaillet@cyu.fr*