Exploring the holistic impact of Adaptive Formative Assessment for Novice Programmers

Jagadeeswaran Thangaraj

M.Eng., MBA, M.Sc.

Supervisors:

Dr. Monica Ward

School of Computing, DCU

& Dr. Fiona O'Riordan

CCT College Dublin

A Dissertation submitted in fulfilment of the requirements for the award of Doctor of Philosophy (PhD)

School of Computing

Dublin City University

January 2026

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work, and that I have exercised reasonable care to ensure that the work is original and have conformed to the regulations on the use and declaration of Generative AI, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed:

Jagadeeswaran Thangaraj

ID No.:A20216568

Date: 07/01/2026

Declaration on the use of Gen AI

I hereby declare that I used Google Gemini 2.0 Flash and Microsoft Copilot between Jan 2025 and Sep 2025. I mainly used these tools to compare the adaptive formative assessment questions with the framework that I developed in this thesis. I asked these tools with the prompts as follows: 'Can you generate adaptive formative assessment questions with customised feedback on the fundamental topics of introductory programming?'. The outcomes are discussed in Chapter-7 of this thesis.

Also I used them to fix the errors in Latex during thesis writing. I asked these tools with the prompts as follows: 'How to fix the package pgf Error: as error message, Unknown arrow tip kind 'Stealth'?'. In addition, I asked alternative words for describing some sentences in the future Work.

Signed:

Jagadeeswaran Thangaraj

ID No.: A20216568

Date: 07/01/2026

Dedication

To my beloved wife, Mrs.**Bharathi Jagadeeswaran**, whose love, patience and belief in me carried me through the toughest days of this PhD. To my mom, Mrs.**Vasantha**, and my dad, Mr.**Thangaraj**, whose prayers, guidance and unconditional support have always been my foundation. And my children, for their heartfelt wishes and prayers that gave me strength along the way.

Acknowledgements

Firstly, I would like to thank my supervisors **Dr. Monica Ward** and **Dr. Fiona O'Riordan** for providing me with the opportunity to carry out this research at Dublin City University (DCU). Your guidance and knowledge throughout the process was invaluable.

Over the course of my doctoral studies I benefited from the advice and support from so many colleagues at DCU. I would like to thank the School of Computing team and leadership (**Dr. Martin Crane** and **Dr. Andrew McCarren**) for their amazing support. Thanks to my colleagues **Dr. Stephen Blott, Dr. Paul M. Clarke, Dr. Brian Davis** and **Dr. Liang Xu (Charlie)** who were a huge support throughout the last number of years. I would like to thank the School of Computing at DCU for funding this research. In addition, I would like to acknowledge the amazing participants of this study.

Finally, I am profoundly thankful to my wife (**Bharathi**) and my children (**Bhavesh** and **Bhavya**) for their love and support throughout my studies. Their belief in me has been a constant source of motivation and strength.

Contents

Declaration i

Declaration on the use of Gen AI iii

Dedication v

Acknowledgements vii

List of Figures xiii

List of Tables xv

List of Abbreviations xvii

Publications xviii

Abstract xxii

1 Introduction 1
- 1.1 Background . . . 3
  - 1.1.1 Formative assessment of Introductory Programming . . . 5
  - 1.1.2 Adaptive assessment . . . 6
- 1.2 Research goal . . . 7
  - 1.2.1 Hypothesis and Research questions . . . 8
- 1.3 Report Structure . . . 8

2 Systematic Review 11
- 2.1 Introduction . . . 11
- 2.2 Systematic review of Programming assessment systems . . . 12
  - 2.2.1 Systematic Literature Review Research Question . . . 12
- 2.3 Systematic Literature Review - Research Methodology . . . 14
  - 2.3.1 Data Sources and Search Strategies . . . 14
  - 2.3.2 Inclusion and Exclusion Criteria . . . 15
- 2.4 Data Coding of Systematic Literature Review . . . 16
  - 2.4.1 Formative Feedback Purpose (SRQ-1) . . . 16
  - 2.4.2 Nature of Formative Feedback (SRQ-2) . . . 18
  - 2.4.3 Support for Novice programmers (SRQ-3) . . . 18
- 2.5 Discussion of Systematic Literature Review . . . 18
  - 2.5.1 Use of GenAI in Formative Feedback . . . 28
- 2.6 Summary of the systematic review . . . 29

2.6.1 Purpose of Formative Feedback (SRQ-1) . . . 29
2.6.2 Feedback Strategies (SRQ-2) . . . 30
2.6.3 Novices' Support (SRQ-3) . . . 30
2.7 Systematic Literature Review Findings . . . 32
2.7.1 Literature Review Finding-1: Customised feedback . . . 32
2.7.2 Literature Review Finding-2: Formative feedback for Novices learning and Introductory programming . . . 33
2.7.3 Literature Review Finding-3: Adaptive formative feedback . . . 34
2.8 Conclusion . . . 36

3 Development of Formative Assessment Framework 37
3.1 Introduction . . . 37
3.2 Formative assessment of Python programming . . . 37
3.3 Exploring Formative Assessments' Potential to Improve Programming Learning . . . 39
3.3.1 Enhancing Programming skills through Learning from Errors . 39
3.3.2 Enhancing Formative assessment with adaptive strategy . . . 41
3.3.3 Encouraging Learning and Proficiency in programming through Adaptive Formative assessment . . . 42
3.4 Development of a Formative assessment framework . . . 43
3.4.1 Framework implementation . . . 44
3.4.2 Non-adaptive model . . . 45
3.4.3 Adaptive models . . . 45
3.4.4 Summary . . . 52
3.5 Research questions . . . 54
3.5.1 Addressing techniques . . . 56
3.6 Conclusion . . . 57

4 Research Methodology and Data Collection Strategy 59
4.1 Introduction . . . 59
4.1.1 Research design . . . 59
4.1.2 Data collection strategy: A mixed methods approach . . . 60
4.1.3 Survey questions . . . 62
4.1.4 The Population . . . 62
4.1.5 Participants . . . 63
4.2 Ethical Considerations . . . 67
4.3 Summary . . . 67

5 Preliminary experiments of Formative Assessment 69
5.1 Introduction . . . 69
5.1.1 Quiz implementation . . . 70
5.2 Phase-1: Validating the quizzes with Non-adaptive model . . . 71
5.2.1 Cohort-C1 (Both/U) . . . 71
5.2.2 Cohort-C2 (Non-CS/S) . . . 73
5.2.3 Faculty feedback . . . 77
5.2.4 Phase-1: Conclusion . . . 78
5.3 Phase-2: Adaptive strategies . . . 78
5.3.1 Adaptive model-1 . . . 79
5.3.2 Adaptive model-2 . . . 80

5.3.3 Adaptive model-3 . . . 81
5.4 Survey questions . . . 81
5.5 Survey results . . . 86
5.5.1 Statistical analysis . . . 87
5.6 Conclusion . . . 93

6 Adaptive Formative Assessment Framework 95
6.1 Introduction . . . 95
6.2 Phase-3: Main study with Adaptive Model-3 . . . 96
6.2.1 Questions Development . . . 96
6.2.2 The Population . . . 98
6.3 Phase-3 Survey Results . . . 99
6.4 Phase-4: Enhanced main study . . . 105
6.4.1 The Population . . . 105
6.5 Phase-4 Survey Results . . . 105
6.6 Addressing Research Questions . . . 111
6.6.1 Research Question-1 . . . 112
6.6.2 Research Question-2 . . . 115
6.6.3 Research Question-3 . . . 118
6.7 Conclusion . . . 120

7 Conclusion and Future work 123
7.1 Conclusion and Key findings . . . 123
7.2 Contributions . . . 127
7.2.1 Student-centred . . . 128
7.2.2 Programming language independent . . . 129
7.2.3 Suitable to other subjects . . . 129
7.2.4 Flexibility and adaptability . . . 130
7.2.5 Scalability . . . 132
7.2.6 Interface language independent . . . 132
7.2.7 Quality of Assessment & Feedback: Better Than AI . . . 133
7.3 Limitations . . . 134
7.4 Future Work: Possible directions for Adaptive formative assessment . . . 137
7.4.1 Formative Assessment using AI . . . 138
7.4.2 Formative feedback using SMT solvers . . . 138
7.4.3 Categorise the difficulty levels . . . 139
7.4.4 Enhancing adaptivity in Formative Assessment . . . 139
7.5 Closing statement . . . 140

A Plain statement and Sample quiz questions 175
A.1 Plain statement . . . 175
A.2 Sample Quiz Questions . . . 177
A.2.1 Question-1 with Aligned Feedback . . . 177
A.2.2 Question-2 with Aligned Feedback . . . 178
A.2.3 Question-3 with Aligned Feedback . . . 179
A.2.4 Question-4 with Aligned Feedback . . . 180
A.2.5 Question-5 with Aligned Feedback . . . 181

List of Figures

3.1 Non-adaptive model 45
3.2 Adaptive model-1 47
3.3 Adaptive model-2 48
3.4 Adaptive model-3 50
3.5 Development process of Adaptive model framework 51

4.1 Flow chart of the research 66

5.1 A sample question in Google Forms 70
5.2 A sample 'Kahoot' question 74
5.3 ICT camp students' feedback 75
5.4 Overall recommendation 76
5.5 All cohort's feedback on learning the Python basics 85
5.6 All cohort's feedback on self-confidence 86
5.7 All cohort's feedback on understanding errors 86
5.8 Mean difference between adaptive models 89
5.9 Comparison of correct answers between attempts of Model-2 91
5.10 Comparison of correct answers between difficulty levels of Model-3 . . 92

6.1 Graph of phase-3 cohorts' responses for S1 100
6.2 Graph of phase-3 cohorts' responses for S2 101
6.3 Graph of phase-3 cohorts' responses for S4 103
6.4 Graph of phase-3 cohorts' responses for S5 104
6.5 All students' feedback (in %) on understanding errors 108
6.6 Average confidence ratings for each of the 16 concepts, segmented by gender 108
6.7 Mean difference on self-confidence 115
6.8 All students' self-confidence responses on all concepts 117
6.9 Self-confidence evolving pattern over concepts 117
6.10 Correct responses percentage by Difficulty levels across quizzes 118
6.11 Correct responses distribution by Difficulty and Cohorts 119
6.12 Difference in exam scores based on quiz participation and gender . . . 120

7.1 A sample question (Python & Java) 130
7.2 A sample question in databases 130
7.3 A sample question (Irish and Spanish) 133
7.4 AI generated questions 134

List of Tables

2.1 Key Search Terms 14
2.2 Inclusion and Exclusion Criteria 16
2.3 Purpose of formative feedback 30
2.4 Nature of formative feedback 31

3.1 Summary of programming topics for intervention 38
3.2 An example question with possible answers and accompanying feedback 44
3.3 Block model for programming 48
3.4 Summary of print statement question in adaptive model-3 50
3.5 Sample questions in Adaptive models 53
3.6 Research questions summary 56

4.1 Summary of interventions 65

5.1 Phase-1 intervention summary 71
5.2 Summary of quiz topics for Cohort-C1 72
5.3 Data of Cohort-C1 responses for S1 72
5.4 Data of Cohort-C1 responses for S2 73
5.5 Summary of ICT summer-camp participants 74
5.6 Phase-2 intervention summary 79
5.7 Summary of quiz topics for Cohort-C3 80
5.8 Summary of quizzes for Cohort-C4 81
5.9 Data of Cohort-C3 responses for S1 82
5.10 Data of Cohort-C4 responses for S1 82
5.11 Data of Cohort-C3 responses for S2 83
5.12 Data of Cohort-C4 responses for S2 83
5.13 Data of Cohort-C3 responses for S3 84
5.14 Data of Cohort-C4 responses for S3 84
5.15 Data of Cohort-C3 responses for S4 85
5.16 Descriptive Statistical analysis of phase-2 experiment 88
5.17 ANOVA effect between the adaptive models 88
5.18 Cohort-C4 correct answers for adaptive model-2 quiz 90
5.19 Cohort-C4 correct answers for adaptive model-3 (A blank indicates no questions were asked) 91

6.1 Summary of questions of each quiz 97
6.2 Summary of quiz topics for Cohort-C5, C8 & C9 98
6.3 Summary of quiz topics for Cohort-C6 98

6.4 Summary of quiz topics for Cohort-C7 99
6.5 Statistical results of all cohorts' S1 responses 100
6.6 Statistical results of all cohorts' S2 responses 101
6.7 Students' feedback for S3 102
6.8 Statistical results of phase-3 cohorts' S4 responses 103
6.9 Statistical results of Phase-3 cohorts' S5 responses 104
6.10 Novices Likert scale responses on Python basic concepts 107
6.11 Descriptive analysis of C8 and C9 responses on Python common errors 109
6.12 Descriptive statistics for students' rate on quiz difficulty 109
6.13 Pearson correlation matrix between Quiz Rate and Confidence Feedback of C8 and C9. **Correlation is significant at the 0.01 level (2-tailed). 110
6.14 Descriptive Statistics for S1–S4 by Cohorts (C5-C9) [C5 (Non-CS/U), C6 (CS/U), C7 (CS/S), C8 (Non-CS/U), C9 (CS/U)] 113
6.15 ANOVA effect for Survey Questions S1–S4 Across Cohorts 114
6.16 Comparison of the novices' confidence before and after quiz attempts 116
6.17 Correct responses percentage by Cohort (CS vs Non-CS) for Each Quiz and Difficulty Level 118
6.18 Mean difference between quiz participants and non-participants . . . 119

7.1 Comparison with AI tools 135

A.1 Answer choices and adaptive feedback for Question-1 177
A.2 Answer choices and adaptive feedback for Question-2 178
A.3 Answer choices and adaptive feedback for Question-3 179
A.4 Answer choices and adaptive feedback for Question-4 180
A.5 Answer choices and adaptive feedback for Question-5 181

Abbreviations

2TSW Gamified Web Based System

AAF Automated Assessment and Feedback

ACM Association for Computing Machinery

AI Artificial Intelligence

AIF Adaptive Instant Feedback

Algo+ Assessment Tool for Algorithmic Skills

ALS Adaptive Learning System

ATF Automated Testing and Feedback

C1-C4 Cohorts - Preliminary experiments

C5-C9 Cohorts - Main experiments

C1 (Both/U) Both CS and Non-CS major/University

C2 (Non-CS/S) Non-CS major/School

C3 (Non-CS/U) Non-CS major/University

C4 (Non-CS/U) Non-CS major/University

C5 (Non-CS/U) Non-CS major/University

C6 (CS/U) CS major/University

C7 (CS/S) CS major/School

C8 (Non-CS/U) Non-CS major/University

C9 (CS/U) CS major/University

CAS Computing Attitudes Survey

CI Confident Interval

CompEd Computing Education Conference

CS Computer Science

CS1 Introductory Computer Science

DCU Dublin City University

DP Drop Project

EPFL Ecole Polytechnique Fédérale de Lausanne

ERIC Education Resources Information Center

HE Higher Education

ICPEC International Computer Programming Education Conference

ICT Information and Communications Technology

IEEE Institute of Electrical and Electronics Engineers

IRT Item Response Theory

ITiCSE ACM conference on Innovation and Technology in Computer Science Education

K-12 Kindergarten and 12 or 12th grade

KST Knowledge Space Theory

LMS Learning Management Systems

M Mean

MCQ Multiple-Choice Question

MMA Mixed Methods Approach

Model Adaptive model

MOOCs Massive Open Online Courses

MULE Maynooth University Learning Environment

N Number of Responses

Non-CS Non-Computer Science

RQ Research Question

S1-S9 Survey Questions

SD Standard Deviation

SIGCSE Special Interest Group on Computer Science Education

SRQ Systematic Literature Review Research Question

TA Teaching Assistants

UKICER The UK and Ireland Computing Education Research

VPL Virtual Programming Lab

List of publications

The research presented in this thesis has led to the following peer-reviewed publications and submissions:

1. **Thangaraj, J.** (2021). Automated Assessment & Feedback System for Novice Programmers. In Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 2 (ITiCSE '21). Association for Computing Machinery, New York, NY, USA, 672–673.

   (*This publication presents the background and motivation of assessment and feedback systems, addressing the challenges in novices learning programming, as discussed in Chapter 1 of this thesis.*)

2. **Thangaraj, J.**, Ward M., O'Riordan F. (2022). Ed-Tech tool for formative assessment of computer programming module. In: ILTA EdTech Conference 2022, 26-27 May 2022, Cork, Ireland.

   (*Forms the basis of Chapter 1 of this thesis, detailing the Educational Tech tool for formative assessment development.*)

3. **Thangaraj, J.**, Ward M., O'Riordan F. (2022). 'Use of Assessment and Feedback systems for Introductory Computer Programming modules of Higher Education: A comparative study.' In: 8th International Conference on Higher Education Advances (HEAd'22), 14 – 17 Jun 2022, Valencia, Spain. ISBN 978-84-1396-003-6.

   (*Forms the basis of Chapter 2 of this thesis, detailing different formative assessment tools for computer programming.*)

4. **Thangaraj, J.** (2022). 'Formative Assessment as a Learning Method for Introductory Programming'. In Proceedings of the 2022 Conference on United Kingdom & Ireland Computing Education Research (UKICER '22). ACM Article 22, 1–2.

   (*Forms the basis of Chapter 2 of this thesis, focusing on research gaps in formative assessment systems for computer programming.*)

5. **Thangaraj, J.**, Ward M., O'Riordan F. (2023). A Systematic Review of Formative Assessment to Support Students Learning Computer Programming. In 4th International Computer Programming Education Conference (ICPEC 2023). Open Access Series in Informatics (OASIcs), Volume 112, pp. 7:1-7:13, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2023).

   (*Forms the basis of Chapter 2 of this thesis, detailing the result of systematic review of formative assessment systems for computer programming.*)

6. **Thangaraj, J.** (2023). 'Adaptive Formative Assessment For Teaching Novices in Introductory Programming.' In Proceedings of the 2023 Conference on United Kingdom & Ireland Computing Education Research (UKICER '23). ACM Article 30, 1.

   (*Forms the part of Chapter 3, 4 & 5 of this thesis, establishing the earlier intervention design of formative assessment systems for introductory programming.*)

7. **Thangaraj, J.** (2023). 'The Impact Of Formative Assessment In Online Programming Learning For Novices.' EDEN 2023 Annual Conference, Dublin, Ireland.

   (*This publication presents the initial intervention results of assessment and feedback systems motivating novices learning programming, as discussed in Chapter 4 of this thesis.*)

8. **Thangaraj, J.**, Ward M., O'Riordan F. (2024). The impact of using Formative Assessment in Introductory Programming on Teaching and Learning. In: 10th International Conference on Higher Education Advances (HEAd'24). Valencia, 18-21 June 2024.

   (*Forms the part of Chapter 3, 4 & 5 of this thesis, detailing the preliminary intervention results of formative assessment systems using three different adaptive models of formative assessment for introductory programming.*)

9. **Thangaraj, J.**, Ward M., O'Riordan F. (2024). 'An Experience with Adaptive Formative Assessment for Motivating Novices in Introductory Programming Learning.' In 5th International Computer Programming Education Conference (ICPEC 2024). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2024.

   (*Forms the part of Chapter 6 of this thesis, detailing the main intervention results of formative assessment systems using adaptive formative assessment (model-3) for introductory programming.*)

10. **Thangaraj, J.**, Ward M., O'Riordan F. (2025). 'The Effects of Difficulty-Based Question Adaptivity in Formative Assessment on Introductory Programming Learning.' In Proceedings of the 33rd International

Conference on Computers in Education (ICCE 2025). Asia- Pacific Society for Computers in Education. ISSN: 3078-4360.

(*Forms the part of Chapter 6 of this thesis, detailing the enhanced main intervention results of formative assessment systems using adaptive formative assessment (model-3) for introductory programming.*)

11. **Thangaraj, J.** (2025). Potential approaches for Adaptive Formative Assessment to motivate Novices in Introductory programming. In Proceedings of The United Kingdom and Ireland Computing Education Research conference (UKICER 2025). ACM, New York, NY, USA.

(*Forms the part of Chapter 7 of this thesis, focusing on possible ideas for future work using adaptive formative assessment for introductory programming.*)

Exploring the holistic impact of Adaptive Formative Assessment for Novice Programmers

Jagadeeswaran Thangaraj

Abstract

Introductory Programming courses facilitate the understanding of the fundamental concepts of programming. Inability to address errors while programming can lead novices to lose interest, making the deliberate introduction of common code errors—alongside strategies to increase motivation and confidence — a key element in supporting their comprehension in programming courses. Formative assessment is one of the approaches for effective programming learning. Most programming education research focuses on either formative assessment or adaptive learning, but not on the intersection. Inspired by this approach, this study proposes the use of adaptive formative assessment as a pedagogical intervention to enhance student confidence and support learning in Introductory Programming courses.

This study introduces an adaptive formative assessment framework that leverages common programming errors as structured learning opportunities. This approach is novel in that it employs a progressive mechanism for assessing and providing feedback on independent programming skills, dynamically adapting question difficulty according to the nature and recurrence of learner errors. This study investigated how effectively these assessments helped students understand, learn and develop a sense of their ability to program computer programs by introducing common programming errors. It investigated the impact of adaptive formative assessment over novices' self-confidence on their comprehension of fundamental programming concepts and their capacity to recognise and correct common errors in programming after engaging with the quizzes. Findings from students' self-rated surveys and analysis techniques indicate that novice students' self-confidence and comprehension of independent programming concepts have significantly increased as a result of the use of difficulty-based adaptive strategies and the introduction of numerous errors in a formative assessment. This study also found that there was a statistically significant difference between the pre and post self-confident values, test scores and completion time. This demonstrates how adaptive formative assessment helps novices learn programming and this approach would be recommended to others working in the field of computer science education.

Chapter 1

Introduction

Any course in a higher education institution worldwide that is concerned with software development requires introductory programming modules. In Ireland, there are a number of programming modules with different names, although many of them have similar content (Becker, 2019). These modules are primarily provided during the first two years of study with different names: E.g. Introductory programming, Introductory Computer Science (CS1) (Hertz, 2010). By introducing syntax and semantics, these modules aim to impart fundamental knowledge of programming languages (Xinogalos et al., 2019). Computer programming modules are crucial for students to feel confident in their study in Computer Science (CS) (Rum & Ismail, 2017). Novice programmers are those taking their first computer programming courses or those with no prior programming experience. For example, this includes first year CS degree students, Secondary school students such as Junior or Senior cycle years. According to a 2016 survey, 66% of first-year students said they had no prior programming experience before starting their third-level course in Ireland (Strong et al., 2017). Research shows that novice programmers find difficult with programming can be divided into three categories: i) having trouble grasping the fundamental concepts of programming structure; ii) learning programming language syntax; and iii) identifying errors and troubleshooting program code (Kadar et al., 2021; Piteira & Costa, 2013; Rosminah & Ali, 2012). Novice programmers often face challenges interpreting program code and have a

lack of understanding of programming principles (Koorsse et al., 2015). Students who are learning their introductory programming must become familiar with the often-hard syntax of the language, numerous data types and its operations, the effects of various statements on variables, and control flow (Islam et al., 2019). Understanding syntax, writing programs, and debugging are examples of independent components (Qian & Lehman, 2019). The cold start problem is the challenge that students encounter when they begin learning a new subject without having enough background information, experience, or context to interact with it successfully (Velde et al., 2021). In programming education, it arises when novice learners are presented with learning materials or assessment tasks without sufficient prior knowledge, leading to disengagement and poor performance. The main challenge is identifying syntactic and semantic errors in programming languages (Bosse & Gerosa, 2017; Kadar et al., 2021). These independent components of programming fundamentals will make programming more difficult for novices in the early stages due to their lack of programming experience and inability to comprehend program code (Luxton-Reilly et al., 2017; Qian & Lehman, 2019; Renske Weeda & Barendsen, 2023). Due to these factors, novice programmers take twice as long to complete programming tasks as more experienced programmers (Bowman et al., 2019). Although the computer science courses are in high demand, introductory programming modules frequently have dropout and failure rates as high as 50% (Malik & Coldwell-Neilson, 2017; Margulieux et al., 2020). These modules could play an important role in making them comfortable in continuing their education in computing (Rum & Ismail, 2017). Their interest in programming will rise once pedagogical methods motivate their confidence, and dropout rates will reduce (Margulieux et al., 2020). Providing students access to more activities to help them grasp basic concepts and common errors may help them learn programming (Singh, 2022).

1.1 Background

Researchers are investigating several approaches to proficiently teach programming and novice programmers in programming modules (Figueiredo & García-Peñalvo, 2020; Ling et al., 2021; Lohiniva & Isomöttönen, 2021; Nunes et al., 2021). There are a number of activities introduced to motivate novice students in programming modules: For example, intelligent tutoring, real-time problem solving, competence-based, etc (Kalelioğlu & Gülbahar, 2014; Shareghi Najar & Mitrovic, 2012). Recommended pedagogical approaches are live coding, pair programming, peer instructions, collaborative coding and assessment & feedback systems (Brown & Wilson, 2018). The traditional approach of teaching programming through pre-coded operating code examples whereas live-coding involves the instructor writing actual code from the beginning without the use of boilerplate or skeleton code (Raj et al., 2018). Since the instructor intentionally writes bug-filled code and makes corrections in real time while the students watch, live-coding can be an effective method of teaching and fixing frequent introductory programming errors (Rubin, 2013). Despite its potential to improve students' coding skills and raise their results, live coding has many disadvantages, including being quite time-consuming. It is challenging for students to take notes and to keep up with the programming's speed (Selvaraj et al., 2021). The collaborative coding and pair programming approaches enable two programmers to work on one task at their own pace. In which one writes code and the other reviews over each line of code. Compared to individual and live coding, these approaches have the potential to create better-designed solutions, fewer errors, and better-quality code (Athinaiou, 2023). However these approaches might be time and resource-intensive, frequently require dedicated scheduling and classroom management (Williams et al., 2008).

An essential component of education that promotes learning is assessment (Schellekens et al., 2021; Shanley et al., 2022). Assessment is an important part of education which leads to learning (Carless, 2014). The multiple ways of

representation, action, and engagement are addressed through a curriculum that includes goals, resources, instruction, and assessment (Boud & Dochy, 2010). Assessment encourages self-regulated learning, which supports the development of adaptable learning skills that can be used in a variety of settings to help students become more motivated, metacognitive, and capable of taking strategic action (Brenner, 2022). It enables learners to determine their own learning requirements, keep track of their own development, and control and maintain their interest, effort, and tenacity during a learning process (Ralabate, 2011). The goal of assessment for learning is to extend learning, foster achievement, and create chances for the growth of self-regulated learners, initiative, and reflective practice (Andrade & Heritage, 2017). Traditional assessment justifies and assesses the students' effort (Ghosh et al., 2020). It simply evaluates what they already know and what they are capable of, not how much behind they are (Asamoah, 2019; Lombardi, 2022). Research to date indicates that the traditional assessment system does not let students know where they fall short or what kinds of mistakes they made (Asamoah, 2019; Lombardi, 2022). It justifies and grades the student's work in the traditional assessment. In addition, feedback plays a major influence on learning and assessment. Automated assessment system evaluates and gives the feedback immediately (Qian & Lehman, 2019). Feedback, self-regulated learning, and adaptive formative assessment are some of the teaching strategies that effectively support students in building confidence and mastering core concepts (Barana et al., 2020). Using this method, learning gains could be improved quantitatively (McSweeney, 2012). The assessment process enhances student learning results in a number of areas, such as coding standards compliance, collaborative learning, and programming abilities (Y. Wang et al., 2012). Formative assessment, used in schools or higher education to detect learning needs and modify instruction, refers to routine, interactive evaluations of student development and comprehension (CERI et al., 2012). Formative assessment aims to increase student understanding, instructor instruction, and learning by providing feedback on students' progress

(Dietrich et al., 2020). It also enables the teacher to assess students' performance as well as their teaching process. Therefore, it should follow a common pattern to avoid risk of variance between different students. There are a number of studies that have been conducted in automated assessment systems in the field of computer science (programming) (Cardoso et al., 2020; Combéfis, 2022; Gordillo, 2019; Ullah et al., 2018; Verkleij, 2019). The literature on adaptive formative assessment has been enhanced significantly throughout this research.

1.1.1 Formative assessment of Introductory Programming

As stated earlier, the 'Introductory Programming' module is the first step in the software development courses (Becker, 2016). The learning outcomes are fundamental concepts of programming such as basic computation, simple input/output, conditional and iterative structures and usage structures. In assessment of fundamental programming concepts, the examiner asks students to design, implement and test a project which is typically complex (Luxton-Reilly et al., 2017; Qian & Lehman, 2019). The modular or independent nature of programming components can significantly increase the cognitive load for novice programmers, making it more challenging for them to learn concepts and solve programming problems effectively (Ettles et al., 2018). Numerous approaches have been developed to simplify assessment of programming concepts and provide students with feedback regarding their errors or areas of difficulty. One area for improvement in the assessment and feedback system of introductory programming modules is ensuring that assessments address the granular components of program and give students clearer guidance on where they struggled in achieving correct results.

Formative programming assessment system evaluates student program submissions and provides timely feedback (Qian & Lehman, 2019). The impact of feedback on learning and assessment is significant (Van der Kleij et al., 2012). Additionally, assessment and feedback are crucial for assisting and motivating their

programming abilities (Dawson et al., 2018; X.-M. Wang et al., 2017). Formative feedback is information given to a student with the goal of changing their way of thinking or acting to enhance learning (Shute, 2008). Formative assessment is one of the approaches for effective programming learning (Sun et al., 2019). To ensure that students receive the correct results, the assessment and feedback systems must examine the programs' individual elements and point out any areas where mistakes were made by the students. Beyond assessment, it increases novice programmers' self-confidence and meta-cognitive awareness (Lishinski & Yadav, 2021; Prather, 2018; Sharmin et al., 2019).

1.1.2 Adaptive assessment

Each learner has strengths, limitations, and favourite subject areas within the framework of the learning environment. Adaptive assessment process assesses the students with different abilities with different sets of questions. In this process, a list of questions is classified in three cognitive levels based on the complexity (like easy, moderate and difficult). Here easy questions assess the basic concepts, moderate questions assess comprehensive knowledge and difficult questions do the applications of the knowledge. In adaptive assessment, different models are utilised depending on the goal (Vie et al., 2017). These models are gaining attention recently and these provide a wide range of advantages in teaching diverse students (Louhab et al., 2018). These allow academics to design flexible methods of engagement in learning; in particular, these provide options for expression and communication (Barana et al., 2020). These models allow supporting the students with feedback to make them understand the concepts of computer sciences and prevent from the cold start problem (Velde et al., 2021). According to the learner's response pattern, adaptive systems allow them to learn new concepts in real time by modifying questions across different levels (Marwan et al., 2020; Vesin et al., 2022). This allows them to overcome cold start problems and gradually refine the difficulty profile and personalize the learning pathway (Pankiewicz, 2021; Velde et al., 2021).

By continuously aligning task complexity with learner readiness, the system fosters confidence, sustains engagement, and supports gradual mastery of programming concepts.

1.2 Research goal

Although adaptive formative assessment is in use in the education sector, it has got more attention in recent years. A range of formative assessments have been investigated to support programming learning goals. While many formative assessment systems are effective for improving performance, some studies suggest they may fail to sufficiently motivate novice programmers, particularly when feedback is generic, tasks are misaligned with students' current level, or when there is limited support for understanding and improving from mistakes (Ullah et al., 2018)

The research problem in formative assessment systems is that these systems fail to motivate the novices to learn programming skills. The aim of this research is to explore current formative assessment processes for novice programmers and to design and develop an alternative approach to help new programmers acquire programming abilities. The novelty of this research is to introduce a progressive approach to assess and feedback the independent component of programming by using adaptive formative assessment. There is a lack of research on the effects of adaptive formative assessment interventions on novices' motivational changes, as well as a lack of published work on adaptive formative assessment of introductory programming in higher education. Along with the advancement of adaptive formative assessment (Marwan et al., 2020; Vesin et al., 2022), new frameworks and models have been introduced (Choi & McClenen, 2020; A. C. Yang et al., 2022). To find out how formative feedback may be developed for motivating, and captivating programming experiences of novices with different abilities, more research is particularly required. While some systematic reviews have been undertaken on adaptive formative assessments in computer programming, this

research is interested in knowing whether adaptive formative assessment is used to scaffold or encourage novice programmers (Marwan et al., 2020; Vesin et al., 2022; A. C. Yang et al., 2022). The objectives underpinning the aim of the research are to explore: The extent to how an adaptive formative assessment helps to build self-confidence in novice programmers by understanding common programming errors as a resource in their programming learning journey.

1.2.1 Hypothesis and Research questions

Given that an adaptive formative system can indicate some of the most common programming errors, the general hypothesis of this research is that **"Using an adaptive approach to formative assessment can increase novices self-confidence in terms of comprehending fundamental concepts and errors in introductory programming"**. In order to either prove or disprove this hypothesis, the following Research Question (RQ)s were the focus of this study since it intends to explore the scaffolding and support that adaptive formative assessment may offer for students learning to program.

RQ-1: Can adaptive formative assessment improve the self-confidence of novice programmers in:

a) accurately predicting the outcomes of fundamental programming concepts?

b) effectively identifying and correcting errors in Python programming?

RQ-2: How effective is adaptive formative assessment in facilitating novices' understanding of distinct conceptual components in programming?

RQ-3: To what extent can adaptive formative assessment encourage novice learners to improve their programming skills?

1.3 Report Structure

The report is structured as follows: Chapter 2 provides a systematic review of formative assessment for computer programming; Chapter 3 outlines the continuous

development of formative assessment framework and research questions that this thesis addresses; Chapter 4 provides research methodology and questions that this thesis addresses; Chapter 5 describe reviews and outcomes for each of the pilot studies; Chapter 6 discusses the research outcomes from main study; Chapter 7 provides the conclusion of this thesis and further possible directions to apply adaptive formative assessment.

Chapter 2

Systematic Review

> Assessment has most effect when —feedback is informative and supportive and facilitates a positive attitude to future learning
>
> *(Boud & Dochy, 2010)*

2.1 Introduction

Formative assessment has recently increased in programming evaluation especially in Higher Education (HE) (Jha et al., 2023). This chapter provides a systematic literature review of the research conducted on formative assessment in programming. The review is primarily concerned with two topics: computer programming and formative assessments. The types of feedback utilised in formative programming assessment and its use in various techniques emphasising the significance of motivation are covered in section 2.4.1. Nonetheless, because this study places a high value on self-confidence building and motivation, various formative feedback are looked at in section 2.4.2 to see if they aid in novice motivation. This chapter provides a systematic review which provides an overview of the variety of formative assessments used in programming courses.

2.2 Systematic review of Programming assessment systems

Recent systematic literature reviews on assessment systems for programming courses tend to concentrate on how useful they are for automatic assessment techniques (Gupta & Gupta, 2018; Kallia, 2017; Pettit et al., 2015) or on the kind of feedback generated by assessment tools for evaluating programming languages or programming paradigms (Combéfis, 2022; Keuning et al., 2018; Le, 2016). Furthermore, other research reviews examined the assessment process, finding that it was primarily concentrated on advanced courses (Marchisio et al., 2020) or other computing subjects (González, 2017). Studies that use automatic assessment programming as a pedagogical strategy typically focus solely on teaching and learning and do not examine findings on inspiring or motivating novices.

2.2.1 Systematic Literature Review Research Question

This systematic review aims to explore the literature which look at strategies for increasing self-confidence in programming learning in order to address the following research question: **"How can an adaptive formative assessment help build self-confidence in novice programmers in learning basic concepts of programming?"**.

Novices often perceive programming as challenging (Luxton-Reilly et al., 2018). A number of non-cognitive elements, including self-confidence, self-efficacy, and self-awareness, can be linked to learning outcomes in the context of programming education (Aljowaed & Alebaikan, 2018). In general, self-confidence is a belief that one can accomplish particular objectives by taking the necessary steps (Pfitzner-Eden, 2016). Self-confidence, or the student's assessment of their own performance, is an important factor that affects how well they learn programming (Kővári & Katona, 2023). In addition, motivation, scaffolding, and metacognitive support are other important factors for learning programming

modules (Acosta-Gonzaga & Ramirez-Arellano, 2022). Students with low self-confidence tend to struggle more and may be less willing to engage with challenging programming and algorithmic tasks (Kővári & Katona, 2023). Novice's confidence when learning computer programming can be increased by giving them immediate feedback (Pfitzner-Eden, 2016). The main focus of formative assessment is immediate feedback which aids in student self-regulation and helps them "learn how to learn" (Grover, 2021). Formative feedback includes pointing out both the positive and negative points in a student's submitted program, and offering guidance on how to finish the assignment without offering a solution (Koulouri et al., 2015). Adaptive feedback would not only confirm an answer's correctness but also provide different information for each response, instead of providing feedback like "wrong" and "correct" (Mejeh, 2024). In the programming context, for example, specific information in a feedback could include the position of the incorrect program code, an explanation of the incorrectly applied programming concept, or a specific suggestion for enhancing the student's solution (Le, 2016). Adaptive feedback in educational systems is usually produced by intelligent tutoring systems or algorithms that assess learner input and adjust responses accordingly (Dāboliņš & Grundspenkis, 2013). Students could therefore try the exercises as many times as they wanted and get feedback. Numerous automated assessment systems and tools have been developed for aiding students and teachers (Luxton-Reilly et al., 2023). This review of formative assessments for introductory programming is guided by the following Systematic Literature Review Research Question (SRQ)s:

- SRQ-1: What is the purpose of the formative feedback techniques for programming languages learning?
- SRQ-2: What is the nature of the formative feedback techniques for programming languages learning?
- SRQ-3: Does the assessment method prioritize helping novices explicitly and influencing their programming learning?

2.3 Systematic Literature Review - Research Methodology

More study is necessary to fully understand the formative assessment that takes place in a programming learning environment. In order to better comprehend the current state of research, this study will examine recent articles that were released between 2013 and 2023. This study used this time frame of the last 10 years because the review topic is only applicable to contemporary studies (Meline, 2006). The What Works Clearinghouse Procedures and Standards Handbook, Version 4.0, from the U.S. Department of Education's Institute of Education Sciences served as the process (Maggin et al., 2022). The five stages were as follows: (a) creating the review methodology; (b) locating pertinent literature; (c) screening studies; (d) reviewing articles; and (e) reporting findings. This study follows these stages to systematically review the literature.

Key Concepts	**Search Terms**
Computer Programming	"novice computer programmers" or "computer programming" or "programming skills" or "first year computer programming" or "programming concepts" or "introductory programming" or "automated system" or "Automated assessment"
Formative Assessment	"formative feedback" or "error correction" or "learner confidence" or "learner efficacy" or "adaptive assessment" or "computer programming assessment system"

Table 2.1: Key Search Terms

2.3.1 Data Sources and Search Strategies

The terms "Formative assessment" and "Computer programming" were broadly entered into databases using the Title, Keyword, and Abstract search functions to look for published publications between the years 2013 and 2023 (Cooper et al., 2018). The selection of these search terms is predicated on concepts and words that are closely associated with the research question (Snyder, 2019). Table 2.1 shows

the alternative terms for the key terms in the search. Academic Search Complete, ERIC Library, ACM, IEEE and Science Direct were the combined databases used. This research used search engines like Google Scholar, ResearchGate, and Academia to find manuscripts (Howland et al., 2009). Additionally, core conferences in computer education such as SIGCSE, CompEd, ITiCSE, UKICER and ICPEC are taken into account. 221 articles were found in the initial search results since 1998. Based on the inclusion and exclusion criteria, these papers were evaluated at the title, abstract, and full text levels. 31 articles were included and then 4 further excluded because they concentrated on basic computer science, block-based programming, summative assessment, or gaming techniques rather than clearly focusing on introductory programming, formative assessment, or assessment approaches, 5 articles were eliminated (Goel & Joyner, 2016; Hainey et al., 2022; Qian & D.Lehman, 2021; Stanja et al., 2023; Trumbull & Lash, 2013). Some tools and infrastructure, both publicly available and locally developed, related to programming module assessment techniques are additionally provided for review.

2.3.2 Inclusion and Exclusion Criteria

Each study must meet these screening requirements to be considered for this systematic review: Computer programming and formative assessment are the article's primary foci, and its publication dates range from 2013 to 2023. Its publication type is original research from peer-reviewed journals, and its research methodology includes both quantitative and qualitative approaches with a clear methods section and results presentation. Its language is English, and its emphasis on formative assessments that are being used to support their programming learning. If a research study did not satisfy one or more of the inclusion requirements that shows in Table 2.2, it was excluded.

Criterion	Inclusion	Exclusion
Time-frame	2013-2023	Prior to 2013 the focus in literature was primarily on plagiarism.
Language	English	Non-English
Access	Full-text availability only.	Only titles or abstracts available.
Sample	Novice programmers	non-typed/non-scripted programming languages, Web design, model development, advanced programming courses (E.g. Data analytics).
Type of publication	Peer-reviewed, original research, conference papers.	Content that was not peer-reviewed or original.
Focus of literature	Presenting findings that help design an enhanced formative assessment system for novice programmers (what works and does not work, and why).	Studies that present findings on summative assessment. Systems designed for advanced programmers.

Table 2.2: Inclusion and Exclusion Criteria

2.4 Data Coding of Systematic Literature Review

Content analysis was used to find categorical themes from narrative data used to inform research focus and feedback strategy. In order to draw logical conclusions, content analysis aims to organize and interpret the data collected (Bengtsson, 2016; Kitchenham & Charters, 2007). The data gathering approach across literature search was mostly concentrated on the research publications focusing on formative or automated assessment. All studies were conducted in higher education or secondary school contexts and took place globally, primarily in Northern America and Europe.

2.4.1 Formative Feedback Purpose (SRQ-1)

Using the main subcategories of formative feedback found in the chosen publications, the researcher expands the following categories as they are interrelated in achieving learning goals including, understanding basic principles, learning how to express these concepts syntactically and semantically, developing problem-solving techniques, constructing programs, testing and debugging programs (Acosta-Gonzaga & Ramirez-Arellano, 2022; Becker & Fitzpatrick, 2019).

Scaffold

The term "scaffolding" describes how an assessment plan might lead students through the steps of a larger project, with the teacher acting as an experienced leader who offers advice along the way (Reiser & Tabak, 2014). The design of an assessment can scaffold the steps of a larger project by asking students to complete the steps so they can receive formative feedback in between (Grover, 2021).

Motivation

A student's motivation is defined as their "willingness, need, desire, and necessity to engage in and be successful in the learning process" (Steinmayr et al., 2019). The two main categories of motivational factors are intrinsic and extrinsic (Adamopoulos, 2019; Ryan & Deci, 2000). Self-motivation is another name for intrinsic motivation, which is the strong desire to learn a subject. Extrinsic motivation occurs when actions are taken to satiate an outside demand or receive an outside-imposed reward. Instead of just enjoying the activity or engaging in it for its own sake, they may be performed for their instrumental benefit.

Engagement

Engagement promotes learning and forecasts students' success (Acosta-Gonzaga & Ramirez-Arellano, 2022). It is a multidimensional meta-construct with elements of behavior, emotion, and cognition. Formative assessment helps students become more engaged in their learning and that makes them more confident in the subject (Scott, 2017).

Metacognitive and Self-efficacy

Understanding how to learn, participating actively, and reflecting on that engagement is defined as metacognition (Conley & French, 2014). An individual's belief in their own ability to do well is known as self-efficacy (Adamopoulos, 2019). For successful programming learning, metacognition and self-efficacy are crucial

abilities (Loksa et al., 2022).

2.4.2 Nature of Formative Feedback (SRQ-2)

This review categorizes studies based on feedback techniques tailored to engage students in learning programming as follows (Hao et al., 2021; Shute, 2008):

Assessment Approach

This defines the assessment approaches to generate the feedback such as, self-assessment, peer assessment, (semi-) automated assessment on programming assignment.

Feedback Types

It defines what type of feedback the tool or system provides such as, standard error messages, customised error messages, testing report or grades.

Feedback Mechanism

It defines how the feedback is generated to notify such as unit testing, test cases, verification and validation, guided instructions or directions of tests.

2.4.3 Support for Novice programmers (SRQ-3)

Using the above mentioned criteria, this review investigates whether these systems offer novice programmers assessment-based support in learning programming and how they facilitate their learning.

2.5 Discussion of Systematic Literature Review

Formative assessment systems offer automatic formative feedback to students who submit their solutions or programs. These systems use various technologies to provide this feedback. Unit testing and verification are two important methods. In

the formative assessment system introduced by (Grawemeyer et al., 2022), students can receive automatic formative feedback based on automated unit and system tests in this system if the code is valid; otherwise, they receive error messages so they can change it. This system tested how well formative feedback connects with students of diverse backgrounds and showed how it facilitated their learning of programming (Grawemeyer et al., 2022). Another system, developed by (Grant, 2017), employed laboratory exercises accompanied by automated test cases created by the lecturer using solely free software testing tools that match industry standards in order to provide students with formative feedback as they were working. The immediate feedback and continued efforts to close the performance gap between actual and expected performance were lauded by the students, and the efficacy was determined to be successful (Grant, 2017).

An Adaptive Instant Feedback (AIF) method was included into a block-based programming environment by another team in order to increase novice programmer engagement and persistence in computer science (Marwan et al., 2020). AIF offers immediate feedback tailored to each student's achievements regarding how well they performed on a particular open-ended programming exercise (Marwan et al., 2022). Given that it increases students' motivation and engagement, it provides positive feedback upon completion (Mitrovic et al., 2013). Example feedback messages are: "You are legit amazing!!, All objectives are done- High FIVE!!, Yay!!!!, you DID IT!!, it's fixed !!, and You are doing great so far!". These positive feedback help novices feel less hesitant about their actions (Mitrovic et al., 2013). In AIF, adaptive pop-up messages have demonstrated promptly well-designed feedback can improve students' performance and learning in programming, which could increase motivation at a crucial moment (Marwan et al., 2020). The study was conducted within a block-based programming environment, which restricts the generalizability of its findings to text-based or advanced programming contexts. Adaptive in this context refers to the sub tasks that are completed to solve a software problem.

The importance of providing meaningful feedback is investigated by another work

that analyses several forms of automated feedback, including simply pointing out errors, highlighting gaps, and providing hints (Hao et al., 2019). It showed that providing only minimal information about failed test cases often led to superficial trial-and-error behavior, whereas adding gap feedback (i.e., differences between expected and actual output) substantially improved student learning and error comprehension. Interestingly, additional hints did not yield significant improvements beyond gap feedback alone. Extending this work, authors found similar results when comparing knowledge of results (KR), knowledge of correct results (KCR, or gap feedback), and elaborated feedback (EF with hints) (Hao et al., 2021). Students who received KCR or EF consistently outperformed those who received only KR, with performance differences widening over multiple assignments. Together, these studies suggest that gap-oriented feedback represents a critical element of effective formative assessment, while elaborated hints offer only incremental benefits unless carefully designed.

The autoCOREctor, a tool for automated student-centered assessment, was created to be easily connected with Learning Management Systems (LMS) (Barra et al., 2020). It encourages the development of a problem-statement scaffold for programming assignments and a straightforward test set with test cases to assist teachers in using it in diverse situations. Regarding the codes they entered, which they must attempt several times to pass, students receive grades and feedback. Because of this, the autoCOREctor's feedback was helpful, easy to understand, guided students to improve their assignments, and increased their motivation. The work's drawback was that it was unclear whether it would be helpful for novice programmers or the introductory programming module.

Because they can accommodate an endless number of students and submissions, Automated Testing and Feedback (ATF) systems were examined to meet the demand (Gabbay & Cohen, 2022). The learning process can be completed by the student by submitting a novel answer and promptly receiving feedback. Feedback can address syntax errors, output accuracy, code performance, and if the code adheres to

instructions exactly. The engagement and learning behaviors of learners in Massive Open Online Courses (MOOCs) are examined in this study. This study (Gabbay & Cohen, 2022) found that code feedback is one of the most crucial aspects of MOOCs for programming and that there might be a positive trend toward ATF users getting better grades.

Incorporating online coding tasks into formative assessment is examined in an article to determine its practicality and efficacy (Dhakshina Moorthy & Dhakshina Moorthy, 2020). Positive results from the experimental investigation back up the use of online coding environments in introductory programming and algorithm courses. It argues that formative rather than summative assessments enhance the learning process for students. Drop Project (DP) is an automated assessment tool built on top of the Maven build automation software (Cipriano et al., 2022). Multiple validations, including unit tests, code quality, execution speed, and memory use, are the basis of formative feedback in DP. By giving prompt feedback on students' work and pointing out their errors, this tool increased their interest in programming classes.

A study looked at two automated assessment methods for online introductory programming courses: Assessment Tool for Algorithmic Skills (Algo+) and Ecole Polytechnique Fédérale de Lausanne (EPFL) (Bey et al., 2018). Based on the discrepancy between the supplied program and the referent solution that is the closest match, Algo+ delivers comments. This distinction clarifies to the learner the processes to follow in order to arrive at the correct response. The feedback given by the EPFL grader is based on test cases and a check style process. The generated feedback educates students about the test case successes and failures by displaying the program's result and the anticipated output. The students' understanding of why the programs do not provide the right responses even though they are syntactically correct is improved by the feedback on the most familiar incorrect programs.

As tutoring systems, chatbots have been used in a variety of settings. A system

aims to increase student engagement and work completion by using chatbots to generate formative feedback while teaching fundamental CS concepts (Benotti et al., 2018). This system generated formative feedback based on character modeling, programming concepts proficiency, and high level concept mastery while testing two distinct chatbots, one for females and one for males in addition to a game programming tool. This system found that female students were more attracted to chatbots than to game systems. Because they produce formative feedback immediately at the task level and use the input to guide students toward learning programming, chatbots are useful tools for formative feedback. The new advancements in GenAI will make this a more effective methodology (Mahon et al., 2024). Another formative assessment technique is that formative assessment is combined with automatic source code verification and validation feedback introduced in Anfurrutia et al., 2018. With the use of this system, formative assessments will be able to provide feedback on the verification outcomes. When errors are discovered during the automatic verification phase, students will be given a report, enabling them to both fix the errors and gain a deeper understanding of the code.

In computer science, formal methods are essential procedures used to ensure that programs and systems function precisely as intended (Broy et al., 2024). They use the rigorous application of mathematical approaches to verify the accuracy of hardware and software designs. Among the techniques used in verification to identify program defects are static program analysis, model checking, and theorem proving (Leino & Monahan, 2007; Szabó et al., 2017). Error checking is done statically, without executing the program, using two distinct approaches in bug detection: static program analysis and model checking (Vorobyov & Krishnan, 2010). Checking if a system is valid in relation to a formal specification or property is known as theorem proving (Mahmoud et al., 2024). An automated assessment system was designed using static analysis to assess programs with an automated assessment library (Delgado-Pérez & Medina-Bulo, 2020). The lecturer

can personalise exercises, reuse verification, and modify the lesson for each student using the library. It showed how flexible feedback on verification helps students identify inefficient and incorrect code fragments and encourages them to adopt good programming techniques.

Another approach of incorporating programming exercises into Adaptive Learning System (ALS) uses a combination of static and dynamic code analysis to directly classify student code into so-called response classes (Lohr, Berges, et al., 2024). In order to foster self-directed learning, this system uses an automated feedback generator to enable personalised feedback in an adaptive learning assistant that offers learners personalised content. This method makes it possible to provide personalised feedback that keeps motivation strong while reducing the barrier to learning programming. The lack of resources makes it difficult for lecturers to offer personalised support; this strategy solves this crucial gap by offering automated, customised feedback. Here adaptability comes into play when creating content based on student achievement.

Guided inquiry learning is an example of an inductive collaborative learning strategy (Lukose & Mammen, 2018). Students are expected to complete the learning objectives and provide their peers and the instructor comments on the problem (Lukose & Mammen, 2018). It showed how receiving peer feedback improved student programming skills. Another (peer code review) system that used formative assessment based on peer code review caused students' programming abilities to consistently improve (Sun et al., 2019). Peer code review and inspection is an effective strategy to ensure the high quality of a software by methodically examining the source code. With the use of peer feedback, the students were able to identify and correct their errors. Similarly, a study looked at how students in introductory programming courses who are not majoring in computer science respond to evaluation situations (Riese & Stenbom, 2020). Students upload their code into the learning management system in this process, and peer reviews—either by other students or Teaching Assistants (TA)s—evaluate it. It was said that

the TAs assisted and mentored the students with their individual projects, lab assignments, and exam preparation. However, some of the students who were questioned expressed concern about the TAs' lack of pedagogical abilities, claiming that even if they wanted to assist, their capacity to offer instruction was constrained. There were instances where students became friends with TAs and could get in touch with them outside of class hours, and some students would recall specific TAs as being really helpful. The volume of feedback received also left some students feeling a little let down. It was therefore not always feasible to access the manual (TAs') feedback, even when it was useful.

A framework for understanding the what, why, and how of formative assessment of introductory programming in Kindergarten and 12 or 12th grade (K-12) computer science was developed in order to answer the overall need for understanding formative assessment (Grover, 2021). Computer Science research on assessment design and programming learning, particularly student misconceptions, has an impact on the formative assessment questions' design (Grover, 2021). Another study focused on how gamified strategy can give students rapid feedback by using an internet platform called HERA (Orozco-Garcia et al., 2019). It argues that the formative feedback was important and helpful for computational thinking (Orozco-Garcia et al., 2019). According to another study (Jansen et al., 2017), suggestions about unit length, unit complexity, and code duplication were the most beneficial to students. While this feedback does not help with the assignment, it enhances understanding (Jansen et al., 2017).

A study (Lishinski & Yadav, 2021) looked at the role of self-assessment in computer programming. The results of this study indicate that self-assessment may improve students' performance in an introductory programming course. It was found that students who got comprehensive feedback on their learning were more motivated than those who merely got a rubric-based assessment. According to this study (Lishinski & Yadav, 2021), it was found that the self-assessment intervention had a significant impact on students' performance on programming projects. Additionally,

students reported higher perceptions of self-efficacy from the latest project feedback than from earlier feedback. Another study (Sharmin et al., 2019) looked at the effects of open-ended assessment on students learning introductory programming in terms of performance and self-efficacy. Students who routinely completed the open-ended versions had higher average self-efficacy scores and assignment marks, though not by a statistically significant amount.

A number of locally developed tools and infrastructures are available to facilitate automated assessment. Leed's automated assessment and feedback platform system enables students to receive automatically generated, instant and personalised feedback (Evans & Wilson, 2020). It provides formative feedback to students after each learning exercise, with feedback usually provided following formal summative assessment. It is designed for multiple programming languages: C, C++, Python and simulation packages. It allows electronic submission and students receive the automatic grading with textual feedback on their submission.

CodeRunner is another feedback system which was developed at the University of Canterbury (Lobb & Harlow, 2016). It follows the Moodle quiz question type. It runs millions of student quiz question submissions in various programming languages: Python, C, JavaScript, PHP, Octave and Matlab. By far the most common use of CodeRunner is in programming courses where students are asked to write program code to solve programming problems and that code is then graded by running it in a series of tests. Adaptive mode is available in CodeRunner which allows students to write the programs to get immediate test case results. If they find any errors with immediate feedback, they can resubmit the code with correct answers with a small penalty.

The Virtual Programming Lab (VPL) is an extension to Moodle which enables academics to assess and grade programming assignments in many programming languages (Rodríguez-Del-Pino et al., 2012). It allows students to edit, run and test the computer programs in the browser. It also allows the academics to automatically evaluate and check plagiarism. It supports many programming languages: Java,

Python, C#, PHP, JavaScript, Matlab and more. It allows the academics to set up different test cases in order to check student's submissions. Therefore, the students are immediately able to receive the feedback about the errors against the test cases. It helps to manage all the programming assignments on Moodle which makes the students' learning process easier. Moodle and a VPL plugin are used in conjunction with the open-source MCTest application to create and assess parameterised programming language questions (Zampirolli et al., 2020). For the purpose of preventing answer sharing, this system generates questions, administers assessments, and gives feedback to various students using distinct sets of questions. Similarly, Mekterovic et al., 2020 developed Edgar, an open-source automated program assessment tool, to evaluate programming assignments composed of several programming languages to prevent potential fraudulent behaviour.

There are a number of pedagogical coding environments available. Maynooth University Learning Environment (MULE) is an online, browser-based pedagogical coding environment, for delivering, marking and providing feedback on coding assignments (Culligan & Casey, 2018). This system evaluates and provides feedback for 'Java' programs. The major advantage of this system is providing feedback immediately when learners submit their programming assignments. However, it fails to provide enhanced error messages to support novice programmers. It follows the unit testing technique in order to grade students' work.

Einstein is another browser (docker) based learning system for programming modules (Azcona, 2017; Azcona et al., 2018). This is a standalone virtual learning environment and it is in use at Dublin City University (DCU) for a number of programming modules. This system lists out the programming exercises on its dashboard and allows the students to submit their solutions in 'Python' language. This system calculates the grades and provides the feedback immediately against predefined test cases. Neither of these two systems notify the students as to where the errors exactly are, how to improve them and what type of the error. However, they both provide immediate feedback and allow resubmission for improved grading.

A Gamified Web Based System (2TSW) is an automated assessment and feedback system to assess complex programming tasks (Polito et al., 2019). It follows a gamified structure to motivate and engage students in programming tasks. In addition to traditional grading, it provides reward batches for student's completions along with grade percentages. By motivating students to explore different test cases, providing actionable guidance on failed attempts, and supporting peer feedback, the system fosters a more interactive and iterative learning process.

In addition to these tools and articles, we have found a few other publications that have developed adaptive formative assessment and researched in other disciplines. One of them is "Firecracker", an online adaptive e-learning platform and assessment tool (Hasan et al., 2021). The authors developed a set of formative quizzes for medical (Cardio vascular course) students in Firecracker, which serve as self-regulated adaptive assessments and encourage critical thinking of students to get ready for challenging coursework and tests. Given that Firecracker was positively received by the majority of students, viewed as helpful, and associated with better academic achievement, it may be used in other courses to track students' progress.

Another study (Ross et al., 2018) examined how adaptive learning might enhance student learning outcomes, motivation and engagement among accounting students by introducing course contents through a set of quizzes. Here, the authors developed adaptive quizzes that enable formative assessment feedback for fundamental conceptual competency. Three levels of difficulty for each quiz were developed for each of the different topic areas. Students can advance to the next level once they reach 90% in the first level, which begins at the basic topics of the course. Even though there were slight rises in student scores, this study drew the conclusion that the adaptive quizzes were not linked to any appreciable gains.

The effectiveness of immediate corrective feedback in introductory physics courses was investigated in another study (Chen et al., 2018). This study used a

specific feedback mechanism called checkable answer feature (CAF) that provides students with immediate feedback on the correctness of their solutions to online assignment problems using green or red tick marks. It enables them to review their answers before submitting them in or proceeding on to the next evaluation round. It investigated the patterns of interaction between students and the CAF and, eventually, the relationship between these patterns with their academic performance. The learning process benefited from the provision of very basic online task-level feedback, however this study did not look into self-regulation, motivation, or productive struggle.

2.5.1 Use of GenAI in Formative Feedback

Recently, students stated using AI tools for a variety of programming activities, particularly concept exposition, code generation, and debugging (Keuning et al., 2024). In programming environments, students generally perceived GenAI tools as helpful, with many stating that they improve productivity and facilitate task completion. The potential of automated feedback has been greatly increased by recent developments in artificial intelligence, especially large language models (LLMs), which provide natural-language, context-aware explanations for errors (Bauer et al., 2025). LLMs like GPT-3.5 and GPT-4 can automatically evaluate programming submissions and generate comprehensive, human-like formative feedback, as demonstrated by recent studies (Azaiz et al., 2023, 2024). More sophisticated frameworks combine LLMs with traditional autograders to deliver feedback that is pedagogically richer and in line with the expectations of educators (Sahu et al., 2025). By linking LLM outputs to current feedback taxonomies, one study explores whether LLMs may produce various kinds of feedback (such as corrective, conceptual clues, and higher-order reflection) in order to elicit particular feedback categories for introductory programming tasks (Lohr, Keuning, & Kiesler, 2024). Furthermore, it has been demonstrated that refining generative models using feedback written by instructors or students enhances the relevancy, tone, and

perceived utility of AI-generated feedback (Pădurean et al., 2025). Using teacher insights and well-established educational feedback approaches, another research suggests a pedagogically grounded framework that directs LLM-generated feedback (Scholz et al., 2025). A benchmark evaluation of several LLMs for generating feedback on programming problem-solving tasks is presented in another piece of research (Silva & Costa, 2025). However, this study revealed that while 63% of feedback indications were accurate and comprehensive, 37% contained issues including incorrect line identification or erroneous explanations, underscoring the drawbacks and risks of depending just on commercial LLMs for feedback. According to another survey, "hallucinations"—situations in which the AI gives false information with high confidence—account for up to 53% of errors in some AI-driven CS assessments (Reihanian et al., 2025). These works emphasise the need for more reliability enhancements on AI, particularly for educational applications. According to research, although AI is effective, it has trouble reaching a modest level of agreement with human experts on generating feedback for complex and context-dependent code. The success of GenAI in programming education is determined by how well it supports or scaffolds the cognitive process, which makes it an outstanding Performance Assistant.

2.6 Summary of the systematic review

2.6.1 Purpose of Formative Feedback (SRQ-1)

In order to determine the aim and different kinds of formative feedback strategies for learning programming languages, the relevant research studies were examined in order to get the answers to the research questions. Overall, the majority of studies found that motivation, engagement, and scaffolding were the purposes of feedback, with metacognitive purposes being recognised in fewer studies (see Table 2.3). In general, the goal of all these studies is to inspire students who are learning programming at various levels.

Instructional strategies	#	**Studies**
Motivation	9	(Anfurrutia et al., 2018; Cipriano et al., 2022; Dhakshina Moorthy & Dhakshina Moorthy, 2020; Jansen et al., 2017; Lohr, Berges, et al., 2024; Lukose & Mammen, 2018; Marwan et al., 2022; Orozco-Garcia et al., 2019)
Engagement	6	(Benotti et al., 2018; Delgado-Pérez & Medina-Bulo, 2020; Gabbay & Cohen, 2022; Grawemeyer et al., 2022; Grover, 2021; Hao et al., 2019)
Scaffolding	5	(Barra et al., 2020; Bey et al., 2018; Grant, 2017; Lishinski & Yadav, 2021; Riese & Stenbom, 2020)
Metacognitive/ Self-efficacy	2	(Lishinski & Yadav, 2021; Sharmin et al., 2019)

Table 2.3: Purpose of formative feedback

2.6.2 Feedback Strategies (SRQ-2)

Across the range of computer courses offered, automated assessments are frequently employed to address the scalability of feedback, which is crucial for learning and assessment for certification (Luxton-Reilly et al., 2023). Customised error messages were provided for facilitating formative feedback that were most frequently used in addition to automated testing results and compiler-based standard error messages. They were customised by peer, manual feedback and rubric-based feedback. As shown in Table 2.4, Guided inquiry, feedback on the quality of the code, and chatbot interaction are among the assessment mechanisms. In general, most systems offer customised error messages to help students comprehend the errors they committed.

2.6.3 Novices' Support (SRQ-3)

The general goals of all these studies are to encourage students studying programming at various stages by giving them feedback on how they engage with the course materials, motivating their enthusiasm with rewards, and analysing and scaffolding their understanding. This analysis showed that some studies have focused on introductory programming (Bey et al., 2018; Dhakshina Moorthy & Dhakshina Moorthy, 2020; Lishinski & Yadav, 2021) and explicitly addressed

#	**Studies**	**Assessment approaches**	**Feedback types**	**Feedback Mechanisms**
6	(Barra et al., 2020; Cipriano et al., 2022; Dhakshina Moorthy & Dhakshina Moorthy, 2020; Gabbay & Cohen, 2022; Grant, 2017; Hao et al., 2019)	Automatic	Test case reports with or without grade	Automated testing
1	(Grawemeyer et al., 2022)	Automatic	Unit test results	Automated testing
2	(Anfurrutia et al., 2018; Delgado-Pérez & Medina-Bulo, 2020)	Automatic	Verification report	(Static) Verifier generated
2	(Lukose & Mammen, 2018; Sun et al., 2019)	Peer	Customised error messages	Peer (Lecturer or fellow) feedback
1	(Lishinski & Yadav, 2021)	Self	Customised error messages	Rubric-based
3	(Bey et al., 2018; Grover, 2021; Orozco-Garcia et al., 2019)	Automatic	Standard error messages	Compiler
1	(Sharmin et al., 2019)	Peer	Customised error messages	Guided inquiry
2	(Benotti et al., 2018; Jansen et al., 2017)	Semi-Automatic	Customised error messages	Feedback about code quality
1	(Riese & Stenbom, 2020)	Semi-Automatic	Customised error messages	Manual feedback
1	(Marwan et al., 2020, 2022)	Adaptive & Automatic	Customised pop-up messages	Feedback about code completion
1	(Lohr, Berges, et al., 2024)	Adaptive & Automatic	Customised content generation	Static & dynamic verifier based

Table 2.4: Nature of formative feedback

novices (Grover, 2021; Lohr, Berges, et al., 2024; Marwan et al., 2020). The assessment system that aids novices, however, makes use of TAs' feedback rather than automatic feedback (Riese & Stenbom, 2020). Nonetheless, the study that tracks learners' progress toward completion and employs automatic formative

feedback targets beginners in particular (Grover, 2021). Additionally, by providing feedback on the subsequent activity, the AIF system may enhance student learning (Marwan et al., 2020). Both of these systems were created to provide K–12 pupils with block programming, and the feedback focused on their development progress. Thus, there is a dearth of research that supports novices' learning of textual programming by focusing technical knowledge on them and using adaptive formative feedback.

2.7 Systematic Literature Review Findings

This systematic review provides an overall analysis of the formative assessment that underpins the programming module in various educational settings. The literature reviewed discovered evidence of several factors that are used in formative assessment, including scaffold, motivation, self-confidence, and engagement. The utilization of feedback techniques and student participation in formative assessment had an impact. This leads to the following findings.

2.7.1 Literature Review Finding-1: Customised feedback

Formative assessment needs to employ strategies for presenting error messages more effectively to assist novice programmers. In these systems, if the code or answer is correct, students can receive automatic formative feedback such as grades and textual feedback for code assignment based on automated unit and system tests (Barra et al., 2020; Grawemeyer et al., 2022). Students praised the quick and meaningful feedback and the ongoing drive to decrease the performance gap between actual and intended performance, and efficacy was rated successful (Grant, 2017; Hao et al., 2019). The results provide evidence that formative assessment is particularly effective in introductory programming courses, highlighting its advantages over summative assessment in enhancing student engagement and learning outcomes (Dhakshina Moorthy & Dhakshina Moorthy, 2020). Verifier

generated feedback enabled the students to recognize and fix the errors using verification techniques (Anfurrutia et al., 2018; Delgado-Pérez & Medina-Bulo, 2020). This systematic review revealed that most systems used customised feedback for formative assessment which adds more scaffolding to support learners' progression to the next level (Gabbay & Cohen, 2022; Grover, 2021).

In addition to automatic feedback, students' programming skills steadily increased because of formative assessment based on peer feedback (code review) (Lukose & Mammen, 2018; Sun et al., 2019). When an automated assessment system was incorporated with manual (peer) feedback, the result was more beneficial, but it was not always realistically accessible because a peer might not be available all the time (Riese & Stenbom, 2020). Researchers and practitioners are becoming more interested in self-assessment because of its potential to advance learning (Z. Yan et al., 2023). Another study (Sharmin et al., 2019) suggests that formative self-assessment may enhance students' performance in a course on basic programming. Students who received granular feedback during their learning were more motivated than those who only received evaluation using a rubric and open-ended questions (Lishinski & Yadav, 2021). Therefore, the research finding is that all these studies found the customised formative feedback in addition to standard error messages were helpful to motivate, scaffold, engage or self-assess the learners in learning programming (Benotti et al., 2018; Jansen et al., 2017; Lishinski & Yadav, 2021; Riese & Stenbom, 2020; Sharmin et al., 2019).

2.7.2 Literature Review Finding-2: Formative feedback for Novices learning and Introductory programming

In college courses, traditional assessment models might not be adequate for formative assessment, particularly when it comes to motivating students to review previously learned content and enhancing their learning outcomes (A. C. Yang et al., 2022). When a student provides an erroneous response to a question, the system can progressively lead the student through a discovery process that results in the proper

solution, breaking down complex concepts one step at a time (Marchisio et al., 2020). By showing the result of the program that was submitted and the anticipated output, the generated feedback instructs students about the test case's success or failure (Bey et al., 2018). However, only a few studies explicitly stated that they focused on novice learners and introductory programming (Bey et al., 2018; Lishinski & Yadav, 2021; Riese & Stenbom, 2020). One of the work's limitations was that it did not say whether it was useful for novice programmers who were just starting out or the introductory programming module (Barra et al., 2020). There is limited evidence that these formative assessments were helpful in increasing specifically novices' self-confidence in programming courses. This review found a few pieces of research that addressed adaptive formative assessment or feedback for inspiring novice learners (Lohr, Berges, et al., 2024; Marwan et al., 2020). Though these components are essential, they have not been used directly in assessment for motivation or programming learning.

2.7.3 Literature Review Finding-3: Adaptive formative feedback

The findings of this review also suggest that several variables may have an impact on the various formative assessment strategies. Adaptive techniques are used in formative assessment to achieve its goals (Vie et al., 2017). There is also less support for several criteria including adaptive strategy, purely because fewer researchers have looked into them (Luxton-Reilly et al., 2023). The limited exploration of adaptive strategies in existing research has resulted in comparatively less support for their integration into formative assessment practices in programming education. Adaptive mode is available in a formative assessment tool (CodeRunner) which allows students to write the programs to get immediate test case results (Lobb & Harlow, 2016). Students can resubmit their revised answers using this tool, however there will be a penalty. AIF offers immediate feedback tailored to each student's achievements regarding how well they performed on a particular open-ended

programming exercise (Marwan et al., 2022). Adaptive pop-up messages in AIF have shown that timely, well-designed feedback can enhance students' programming skills and learning, potentially increasing motivation at a crucial moment (Marwan et al., 2020). However, the feedback focuses more on its completeness than on offering advice on how to improve the solution.

ALS uses an automated feedback generator to enable personalised feedback in an adaptive learning assistant that offers learners personalised content to foster self-directed programming learning. ALS employs adaptivity to create customised content rather than increasing self-confidence. Consequently, rather than supplying resources that are universally applicable, adaptive assessment systems could customize instruction and assessment by considering each student's uniqueness. Recent advances in Item Response Theory/Computerized Adaptive Testing (IRT/CAT), Bayesian Knowledge Tracing (BKT), and Deep Knowledge Tracing (DKT) provide tools for such design (Sharpnack et al., 2024). Integrating embeddings of code edits, unit-test outcomes, and error traces allows the system to deliver granular scaffolds, such as debugging prompts or step-wise guidance (L. Yang et al., 2025). BKT offers interpretable mastery estimates for specific knowledge components (KCs). This transparent KC feedback helps novices understand progress and reduces frustration, a key factor in sustaining motivation (Shi et al., 2022). These timely supports keep learners engaged while gradually raising task difficulty. Research shows that learners' confidence is a powerful predictor of outcomes including skill development, though cognitive capacity and prior preparation are important factors in programming success (Shi et al., 2022). Thus, the improvement of programming self-confidence should be a top goal when developing curricula and assessments (L. Yang et al., 2025). Hence, this study investigates if the adaptive formative assessment increases participants' confidence in their ability to comprehend the underlying concepts of programming.

2.8 Conclusion

Based on this systematic review, we recommend integrating adaptive strategies into formative assessment to better support novice programmers, fostering both their self-confidence and the development of programming expertise. Hence, this study investigates if the adaptive formative assessment increases participants' confidence in their ability to comprehend the underlying concepts of programming. In this scenario, customised error messages might be extremely helpful in improving novices. Accordingly, our study will next develop a formative assessment system that employs adaptive strategies with enhanced error messages to evaluate the ability level of the novice students and increase their self-confidence.

Chapter 3

Development of Formative Assessment Framework

3.1 Introduction

This research utilises the knowledge gained from the systematic review discussed in chapter 2, which indicated that many programming assessment and feedback systems have certain shortcomings. The primary one is that they assess the students with the same questions, with no differentiation for student ability. (i.e. it asks the same questions for all students with different abilities). The second shortcoming is that these systems do not focus on increasing the novices' self-confidence in programming assignments apart from notifying the errors as compiler messages. Therefore, our primary aim is to design an adaptive assessment system that enables novices to comprehend programming fundamentals and increase their self-confidence in learning programming. This chapter outlines the progressive development of an adaptive formative assessment framework for introductory programming modules.

3.2 Formative assessment of Python programming

Python is an established programming language that is utilised in introductory programming courses due to its convenience and syntactical simplicity (Johnson

et al., 2020). It is increasingly used for teaching beginners in programming (Cutting & Stephen, 2021). Variables, operators, conditionals, loops, and functions were all covered in the introductory programming course (Dirzyte et al., 2023). Python is the language used for instruction, and novice programmers may find these topics more difficult (Bosse et al., 2019). This study only takes the introductory 'Python' programming language and its basic topics into account as shown in Table 3.1.

No	**Programming Topics**
1	Print
2	Variables & operations
3	Control structures (if/else)
4	Control structures (while loops)
5	Functions

Table 3.1: Summary of programming topics for intervention

As noted in Chapter 2, formative assessment is a useful methodology for enhancing learning outcomes and providing learning motivation. It is widely practised in educational institutions and proved that it is helpful to achieve the learning objectives (Grover, 2021). Inspired by this approach, we have designed a formative assessment framework for introductory programming modules and have investigated how they can be helpful in increasing self-confidence and motivating novice programmers. Formative assessment provides feedback immediately to the student's attempts irrespective of right or wrong answers. The students' program submissions are evaluated and immediate feedback is provided by an automated formative assessment system or manual assessor. Two characteristics were deemed crucial while intending to develop formative assessments of programming tasks. Firstly, for error messages to be properly understood, feedback needs to be quick and detailed. Second, students must be given the chance to understand their mistakes after making a number of attempts on different questions. Due to the nature of these elements, it is necessary to create quizzes large enough so that students may repeat assessments without encountering the same questions twice. This chapter presents recommendations on how to create these formative assessments in an efficient manner so that students can grasp the material and

gain motivation based on their level of computer programming proficiency. While considering the limitations of automatic assessment, this framework emphasises the importance of achieving comparable difficulty of questions and maximises the potential for randomization of exercise tasks. This helps to create meaningful feedback for students and supports their learning process.

3.3 Exploring Formative Assessments' Potential to Improve Programming Learning

Formative feedback is information provided to a learner with the intention of altering their behavior or way of thinking to improve learning (Shute, 2008). By providing students with feedback information that helps them during the learning process and results in improved learning outcomes, formative assessment is intended to support learning (Darrell Evans, 2013). Formative feedback gives teachers the chance to personalize their responses, motivate student interest, gather data on each student's progress, encourage reflective thought, and encourage self-directed learning (Näsström et al., 2021; Van der Kleij & Adie, 2018). By assessing a learner's current condition and choosing appropriate future steps, formative assessments, sometimes referred to as assessments for learning, are meant to support teaching and learning (Zhai et al., 2021). Therefore, formative assessment refers to routine, interactive evaluations of student development and comprehension that are used in educational settings to identify learning needs and adapt training (Ismail et al., 2022).

3.3.1 Enhancing Programming skills through Learning from Errors

When writing programs, both novice and experienced programmers make errors — just not to the same extent. Different kinds of errors demotivate novices and can lessen their engagement levels when learning to program (Bosch & D'Mello, 2015). Error detection and correction play a central role in learning to program. It is an

essential cognitive and pedagogical component (Misirli & Komis, 2023). Debugging techniques must be established to identify errors, which increases the likelihood that they will be corrected (Misirli & Komis, 2023).

Programming errors can appear in a variety of forms such as syntax and logical errors (Ahadi et al., 2018; Alzahrani & Vahid, 2021; Ben-Yaacov & Hershkovitz, 2022). A program error that breaks language conventions and stops execution is known as a syntax error (Alzahrani & Vahid, 2021). When a syntax error occurs in a compiled language, the compiler will usually produce an error message that points to the incorrect program lines. Most of the time, program compilers provide information on the kind and location of syntax errors. Compiler error messages can also be used to identify and fix syntax and semantic errors during program compilation or execution. However, for novice programmers, these error messages can occasionally be difficult to understand and make them confused (Becker et al., 2016). Therefore, some research suggested improving error messages to make them easier to interpret and understand (Charles & Gwilliam, 2023; Denny et al., 2021). They demonstrated that these enhancement messages were more useful than compiler messages for figuring out common syntax and semantic errors (Becker et al., 2016; Charles & Gwilliam, 2023).

In addition, logical errors irritate novice programmers more than syntax errors (Alzahrani & Vahid, 2021; Ettles et al., 2018). There are three possible explanations for these logical mistakes: Algorithm, misunderstanding, and false information (Ettles et al., 2018). Compilers are unaware of the ways in which students use logic. Another factor, called 'Programmer misconception', contributes to the misunderstanding of control flow in logic that led to the erroneous code (McCall & Kölling, 2014). Consequently, compilers may produce unintended results. It is also quite challenging to identify and correct logical errors. Various studies have identified common logical errors that people make frequently and offer suggestions on how to avoid them (Alzahrani & Vahid, 2021; Ettles et al., 2018). Thus, helping students understand their programming errors is another effective technique to

teach them programming, as mistakes are a powerful means of instruction for programming courses (Hoffman & Elmi, 2021; Zhou et al., 2021). Consequently, in order to facilitate students' understanding, our goal is to familiarise students with these types of errors in advance. It enables students to comprehend both common code errors and compiler error messages.

3.3.2 Enhancing Formative assessment with adaptive strategy

Assessments that are customised to each student individually based on their responses to previous test items are referred to as adaptive assessments (Papanastasiou, 2021). Students who complete computer-adaptive assessments have their ability level determined by the answers they provide; the most illuminating problems are then shown to the students to measure their abilities more accurately (Goto et al., 2023). When taking a formative adaptive assessment, every student will experience it differently from another (Vie et al., 2017). In a traditional assessment, every student works on the same set of tasks with varying degrees of difficulty. Students can work on the predetermined tasks in any sequence because they are predetermined. On the other hand, in a computer-adaptive assessment, every student works on a customised set of tasks since the questions are chosen by an algorithm that considers the answers that each student has previously provided. Therefore, it varies from traditional assessment in that each participant is asked a separate set of questions rather than all of the same ones (Vie et al., 2017). As a result, the issues that students work on vary, requiring varying amounts of time and complexity (Goto et al., 2023). Adaptive assessment comes in two main forms: variable length and fixed length (Phelan, 2019). In a fixed length adaptive assessment, each student responds to the same length questions with varying levels of complexity based on their ability. In an adaptive assessment with variable length, students respond to questions of varying lengths until they meet the system's required competency level. Personalised quizzes are generated by the

system with both attempted and unattempted items of suitable difficulty that need to be reviewed (A. C. Yang et al., 2022).

3.3.3 Encouraging Learning and Proficiency in programming through Adaptive Formative assessment

The process of learning programming involves starting from the beginning and using a completion method, such as changing or finishing a program code including writing a simple program from scratch. When it comes to cognitive skills, programming is more advanced than rote memorization; to complete programming activities, students must grasp particular concepts and understand how to apply them together (A. C. Yang et al., 2022). Self-regulated learning abilities, inventiveness, and self-efficacy of students can be used to adaptively generate assessments. These assessments are customised to each student in terms of question difficulty, assessment length, and question types (e.g., multiple choice, fill-in-the-blank, or short response) (A. C. Yang et al., 2022). An answer key and a number of choices make up the two components of a Multiple-Choice Question (MCQ) (Abreu et al., 2018). MCQ is a type of objective assessment wherein participants are requested to choose only the correct responses from a list of choices. MCQs are insufficient for evaluating a student's coding proficiency in programming modules since they do not encourage learners to write their own code, even when they are at an advanced level. However, MCQs have the ability to help students retain programming concepts and increase their engagement in learning (Abreu et al., 2018; Mohamed Shuhidan et al., 2010). The assessment length and question difficulty can also be modified. An adaptive approach in assessments allows students to learn from their mistakes and provides the option to retry with another attempt until they submit a right answer. This approach can facilitate their understanding. Therefore, this thesis proposes an adaptive assessment technique to assess the novice students in their skill level and encourage them to learn to achieve the required knowledge. Using adaptive assessment, which organises a collection of questions into three cognitive levels

according to complexity (easy, moderate, and difficult), programming adaptive testing evaluates students' knowledge in programming courses (Chatzopoulou & Economides, 2010). Consequently, rather than only supplying resources that are universally applicable, adaptive assessment systems could customize instruction and evaluation by considering each student's uniqueness. Therefore, the aim is to design a formative assessment framework to improve their programming proficiency. Hence, this research investigates if the adaptive formative assessment increases participants' confidence in their ability to comprehend the underlying concepts of programming.

3.4 Development of a Formative assessment framework

Formative assessment can help students feel less anxious about giving incorrect answers by providing feedback on how to improve their work. By using a constructive criticism technique that provides feedback in a standardised form, it can also be feasible to lessen some of the tension caused by getting unfavorable feedback (Burgess et al., 2013). Formative quizzes help students better understand the course material in addition to encouraging them to revise or practice (Ross et al., 2018). This has motivated researchers to create self-regulated MCQ quizzes that are an excellent method to increase student engagement and help them to remember the content (Feudel & Unger, 2022). We have created formative assessment quizzes to expose students to common program errors. We have a list of questions with various answers in these quizzes. These quizzes help them learn as they provide feedback for every answer. While feedback on the incorrect answer helps them locate the appropriate response, the feedback on the correct answer acknowledges their correct responses. Students can learn from their incorrect responses and determine the correct response. As a result, it is a system that helps them to learn from their mistakes. This study examines if the formative assessment

approach increases participants' confidence in their capacity to understand the fundamental ideas behind programming. It investigates if this approach assists students in comprehending the common code errors they make as well as compiler error messages.

3.4.1 Framework implementation

This research developed the framework using a set of MCQ quizzes. 'Google Forms' was utilised to implement the quizzes because it can be an effective tool for formative assessment and for promoting active learning (Djenno et al., 2022). Each quiz aims to educate students about common code errors that novice programmers usually make when learning a particular topic. An example question is shown in Table 3.2. Feedback will be given to students for each potential response, which will help them better comprehend the errors and help them to understand easily as shown in Table 3.2. The feedback consists of customised messages that are similar to enhanced error messages (Becker et al., 2016). Every incorrect reaction offers advice on how to respond next. Students can select the best alternative based on the feedback. These assessments are examined in four distinct models.

Question	**Responses**	**Feedback**
What will be the output of following code? **var = "computer"** **print(var[5::1])**	computer	**Incorrect!** This would be true if it is **print(var[::1])** (no starting index) or **print(var)**.
	compu	**Incorrect!** This corresponds to **print(var[::5])**, (everything up to but not including index 5).
	ter	**Correct! print(var[5 :: 1])** - it prints a range of items starting from index 5 with step 1.
	t	**Incorrect!** if the index is '5' **print(var[5])**, then it prints t.

Table 3.2: An example question with possible answers and accompanying feedback

3.4.2 Non-adaptive model

We carried out the experiments in a university programming course to evaluate the proposed formative assessment with an adaptive approach and a non-adaptive approach. In a non-adaptive model, students receive feedback for each response regardless of whether it is correct or incorrect, as seen in Figure 3.1. Depending on the student's correct or incorrect response to Q1, this system provides feedback or an explanation before moving on to Q2. This feedback allows students to learn from their mistakes and help them comprehend what the correct response is. However, they cannot reattempt the same question (Q1) to correct their answer. This approach continues until the quiz's end.

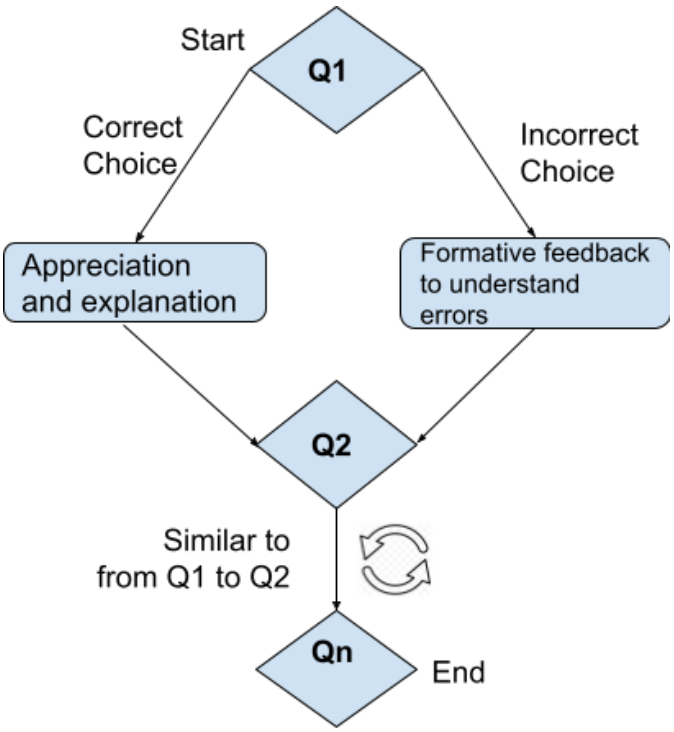


Figure 3.1: Non-adaptive model

3.4.3 Adaptive models

Enhanced personalization can be achieved through adaptive assessments, which arrange instructional resources (Vie et al., 2017). Research shows that the knowledge level of the learners increased by varying the order of the assessment

questions in an adaptive approach (Heitmann et al., 2018). Two criteria are used in the design of adaptive formative assessment: a next item criterion and a termination criterion (such as the number of questions to ask) (Vie et al., 2017). The two most powerful theoretical frameworks utilized in the subsequent item selection process are Item Response Theory (IRT) and Knowledge Space Theory (KST) (Ihichr et al., 2024). IRT can choose items based on examiner level and item features using either probabilistic or non-probabilistic methods. To choose items that are appropriate for the learner's level and the domain's underlying knowledge structure, KST usually employs a non-probabilistic method. In this research, the adaptability has been introduced using three different strategies to attain required knowledge using the KST approach. The adaptive sequencing mechanism is informed by KST which enabled branching rules based on prerequisite structures among programming concepts. The assessments full completion or the level of knowledge for a given topic gained could be the termination criteria. This study employs three adaptive models for question termination in order to ensure that the learner acquires a specific piece of knowledge in each question. The following section explains each model in detail.

Adaptive model-1

Mastery Learning is an educational approach that enables students to be more proficient in the material covered in a course through several attempts (Inventado, 2019; Toti et al., 2023). Application of Mastery Learning has been successful at all educational levels since it uses failed attempts to provide feedback (Toti & Chen, 2024). Hence, the first adaptive strategy involves asking questions on the same level until the right answer is provided as indicated in Figure 3.2. Here, different questions are asked on similar content, however the scenarios were different (Heitmann et al., 2018). Students may choose correctly based on the feedback; in which case they will be able to see exactly where their previous selections went wrong. **Limitation:** Students lose interest in studying because of this method's long completion time if

they still do not understand.

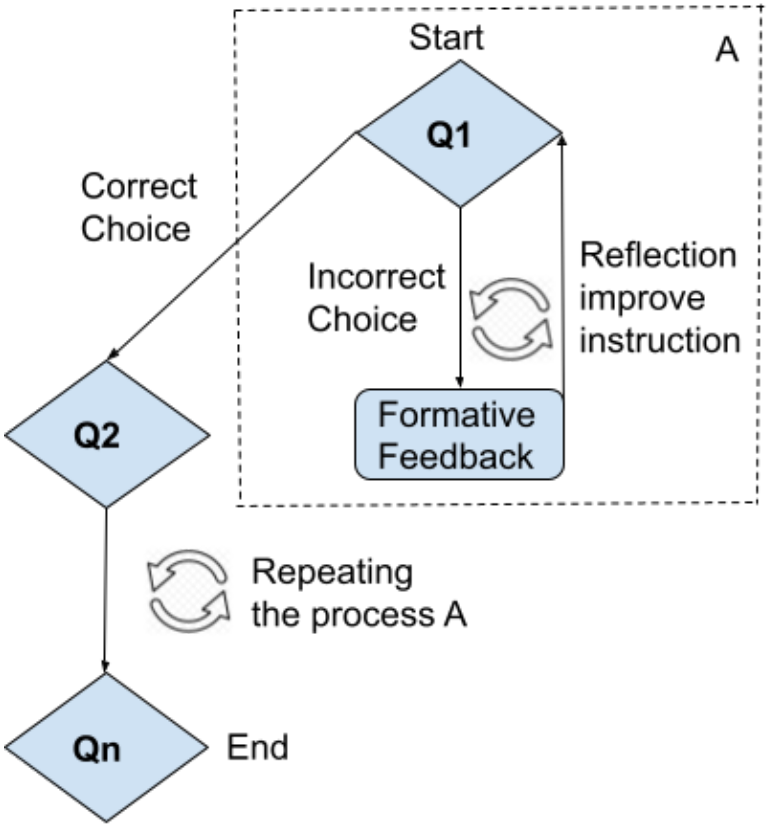


Figure 3.2: Adaptive model-1

Adaptive model-2

This model uses the block model of program comprehension to design the question traversals. According to (Schulte, 2008), the Block Model is a pedagogical model that demonstrates how students build up comprehension skills as they read a program. Programming problems are categorised using the Block model of programming. As seen in Table 3.3, it is presented as a four-layered structure. A single comprehension component is represented by each cell. This idea serves as the foundation for the quizzes' adaptability. By starting with easier tasks and progressively adjusting difficulty, systems maintain novices within an optimal challenge zone (A. C. Yang et al., 2022). Such dynamic selection encourages early success, builds confidence, and sustains persistence. In this model, the question starts from the first level (Macro structure). The following question at

the same level is posed if a student gives a suitable answer as shown in Figure 3.3. The question for the next level of the model will be posed if they do not. This study investigates whether this model increases students' confidence in learning programming.

Limitation: Despite providing thorough covering of all areas, this model requires a significant amount of time for less proficient students.

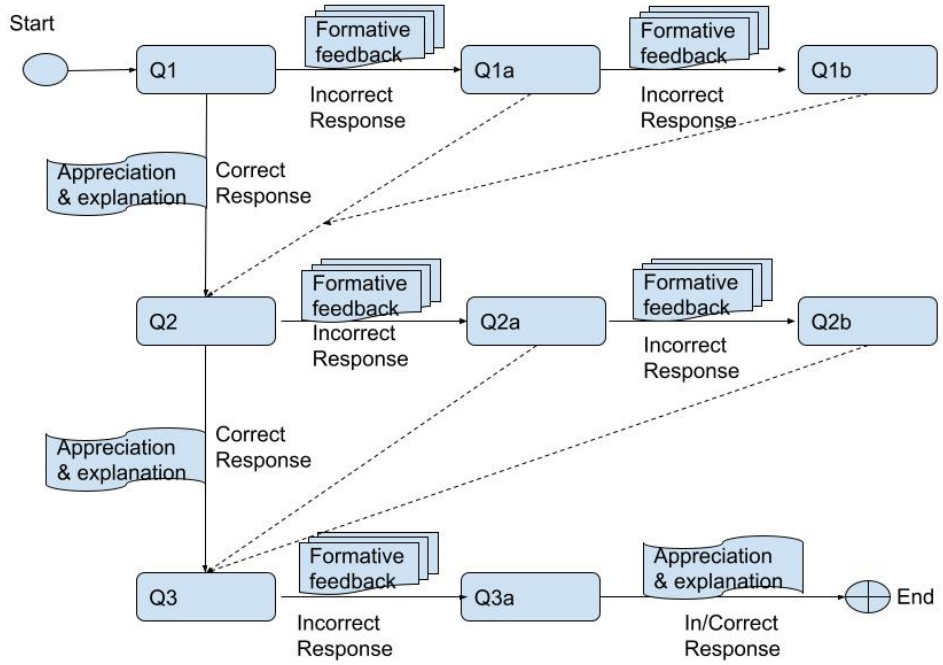


Figure 3.3: Adaptive model-2

Macro structure	Understanding the "algorithm" of the program
Relations	object sequence diagrams
Blocks	Operation of a block, a method, or a sequence of statements
Atoms	Operation of a statement

Table 3.3: Block model for programming

Adaptive model-3

Difficulty levels have been added to learning objects in this model. These things could be topics, questions, or a variety of errors. The goals relate to questions with varying degrees of difficulty. Difficulties in programming are classified using Bloom's taxonomy of programming (Thompson et al., 2008). Bloom's taxonomy offers a well-established cognitive framework that can inform automated question design for programming learning (Fuller et al., 2007). Research indicates that cognitive tasks aligned with Bloom's taxonomy exhibit higher discrimination indices, meaning they more effectively differentiate between learners of varying abilities (Hamamoto Filho et al., 2020). This supports the categorization of tasks into difficulty levels corresponding to Bloom's taxonomy, facilitating the estimation of item difficulty parameters in IRT models. The lower three levels of Bloom's taxonomy — Remember, Understand, and Apply — are particularly important when designing programming tasks for novices, as they provide the essential foundation for higher-order skills (Faradilla et al., 2019). Drawing on Bloom's taxonomy, programming difficulties can be categorised into three cognitive levels: Remember (Easy), Understand (Moderate), and Apply (Hard) (Sobral, 2021). This classification not only structures the progression of programming tasks but also serves as a foundation for adaptive formative assessment, where question difficulty can be dynamically adjusted to align with students' demonstrated mastery (Louhab et al., 2018). At the Remember level, learners acquire basic syntax and programming terminology, which provides the foundation for all subsequent learning. The Understand level develops comprehension of how programming constructs operate, enabling learners to trace, explain, and predict program behavior. At the Apply level, students practice writing small programs and solving procedural problems (Velázquez-Iturbide, 2021). In this model, easy questions assess the basic concepts, moderate questions assess comprehensive knowledge and difficult questions assess the application of the knowledge (Vie et al., 2017). The adaptive sequencing mechanism is informed by KST which enabled branching rules

based on prerequisite structures among programming concepts (Ihichr et al., 2024). If a student successfully responds to a moderate question on this assessment, the subsequent question is hard. If not, the easy questions will be asked as indicated in Figure 3.4. It goes on until the system forecasts the competency level of the students (Phelan, 2019). A sample classification is described as Table 3.4. The development process of the adaptive formative assessment framework is shown in Figure 3.5.

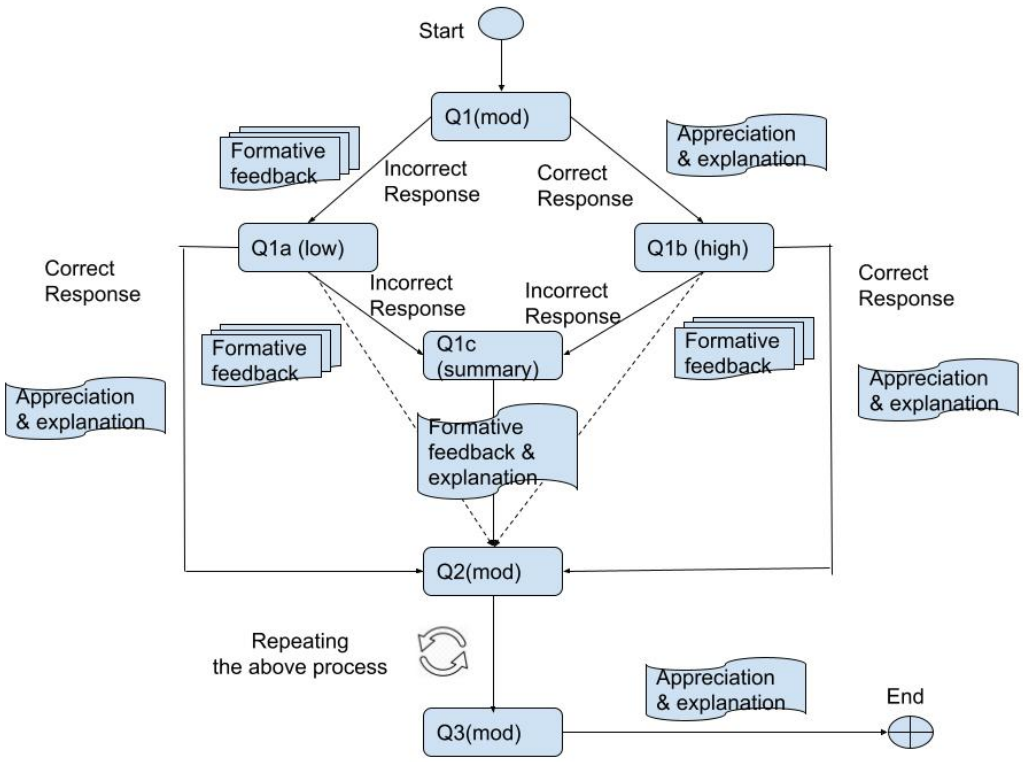


Figure 3.4: Adaptive model-3

Purpose	**Difficulty**			**Summary**
	low	**moderate**	**high**	
String notation and character traverse	var = 'Amazon' print(var[4])	var = 'computer' print(var[5 :: 1])	var = 'Ireland' print(var[4 :: -1])	var = 'James Bond' print(var) print(var[3]) print(var[5 :: 1]) print(var[5 :: -1])

Table 3.4: Summary of print statement question in adaptive model-3

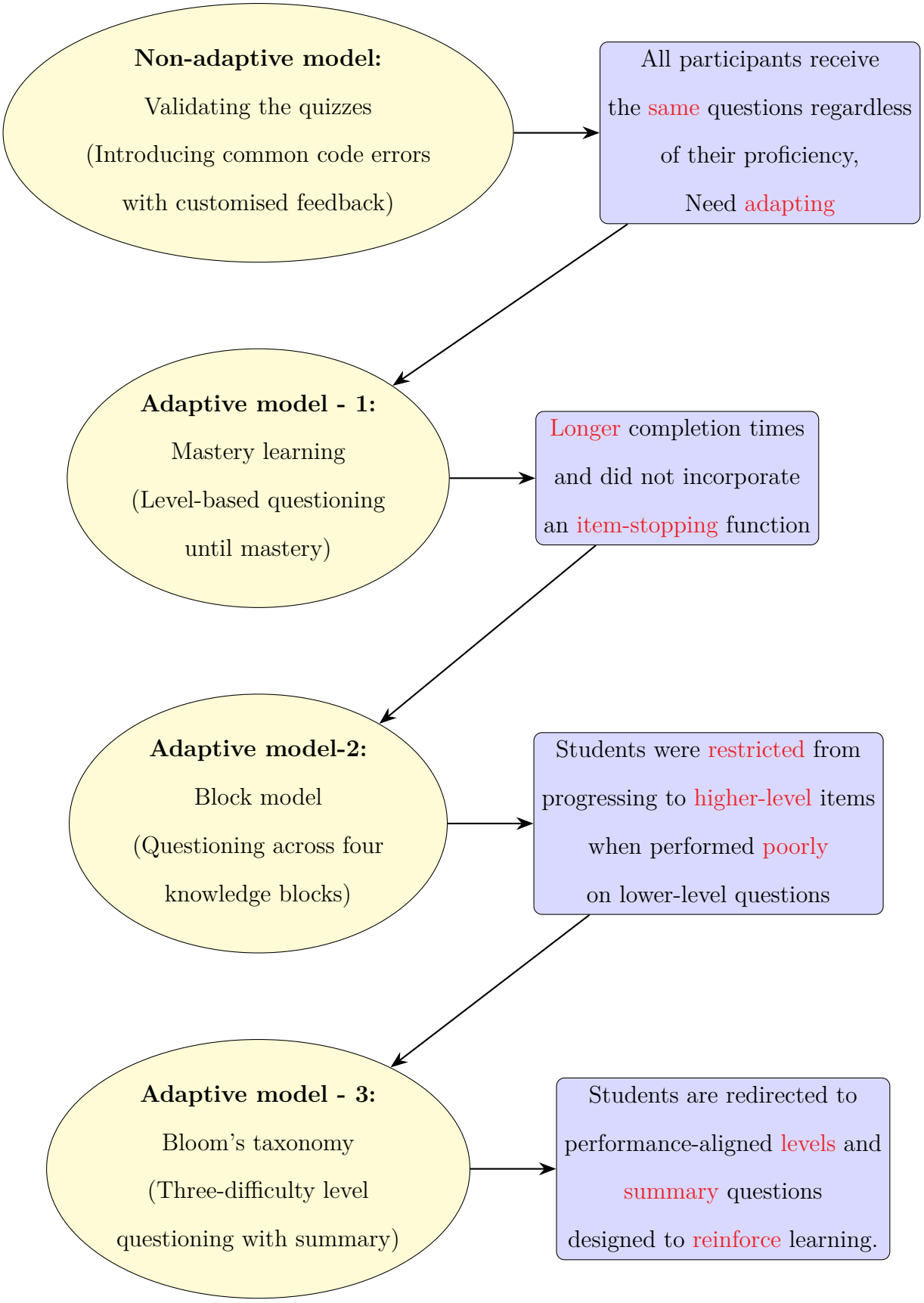


Figure 3.5: Development process of Adaptive model framework

3.4.4 Summary

We developed a collection of questions and answers in this framework, along with feedback for each option. In a non-adaptive approach, students receive feedback for each response regardless of whether it is correct or incorrect and onto the next question. The adaptive strategy involves asking different questions until the level of knowledge is attained. In three models, sample questions' classification for the 'if-else' concept is displayed in Table 3.5. The main question is, 'What will be the output of the following Python code?'. According to the answer, adaptive formative assessment presents appropriate sub-question.

Although adaptive formative assessment systems can now offer real-time personalised feedback, automated error detection, and dynamic question generation thanks to recent developments in Artificial Intelligence (AI), this study chose a non-AI approach because of the project's timeline and practical considerations (Bimpeh, 2024). Early AI models for teaching programming, particularly prior to 2022, were experimental and frequently insufficiently reliable or accessible for widespread classroom use (Chassignol et al., 2018; Zawacki-Richter et al., 2019). A transparent and replicable framework for isolating the effects of dynamic learning outcomes was provided by a non-AI adaptive design that was built on difficulty-driven sequencing, carefully chosen error-specific feedback, and controlled question branching (Katsaris & Vidakis, 2021; A. C. Yang et al., 2022). Therefore, the adaptive formative assessment framework in this study was developed with non-AI methods, taking into account the educational priorities and technical environment at the project's start in 2020. In Chapter 7, we will discuss possible directions for incorporating AI-based methods to enhance the adaptive formative assessment system.

Model	Main question	Response	Sub-question

Model-1 (Similar level)	Q1:
	—
	`if x > y:`
	`  print("Greater:x")`
	`else:`
	`  print("Greater:y")`
		Correct	`count = 0`
			`while count < 5:`
			`  print("Hello")`
			`  count += 2`

		Wrong	`if num > 0:`
			`  print("Positive")`
			`else:`
			`  print("Negative")`

Model-2 (Block)	Q1.Block: — `print("x is greater")`	Correct	Q1a.Relation: `print(score >= 90 )`

		Correct	Q1b.Macro:
			`if score >= 50:`
			`  print("Grade: Pass")`
			`else:`
			`  print("Grade: Fail")`

Model-3 (Easy, Moderate, Hard)	Q1.Moderate
	—
	`if weekend or holiday:`
	`  print("sleep")`
	`else:`
	`  print("Wake up")`
		Correct	Q1a: Hard
			`if a // b  > b or c <= a :`
			`  if (a-c) * (b+1) == 28:`
			`  print("Alpha")`
			`  else:`
			`  print("Beta")`
			`else:`
			`  if (a+c) // b == c-a+b:`
			`  print("Gamma")`
			`  else:`
			`  print("Delta")`

		Wrong	Q1b: Easy
			`if age >= 18 :`
			`  print("Can drive")`
			`else:`
			`  print("Cannot drive")`

Table 3.5: Sample questions in Adaptive models

3.5 Research questions

The following research questions are addressed in this thesis, as stated in Chapter-1:
RQ-1: Can adaptive formative assessment improve the self-confidence of novice programmers in:

a) accurately predicting the outcomes of fundamental programming concepts?

b) effectively identifying and correcting errors in Python programming?

RQ-2: To what extent can adaptive formative assessment encourage novice learners to improve their programming skills?
RQ-3: How effective is adaptive formative assessment in facilitating novices' understanding of distinct conceptual components in programming?

RQ-1: Can adaptive formative assessment improve the self-confidence of novice programmers in:

a) accurately predicting the outcomes of fundamental programming concepts?

b) effectively identifying and correcting errors in Python programming?

In the case of programming modules, MCQs are not enough to assess the student's coding knowledge and they do not engage students to write their own code when they are at an advanced level. However, it can be helpful for recalling a programming concept and motivating engagement. In these quizzes, we have a list of questions with different choices. Each choice provides different feedback when they attempt the question. As they offer feedback for each selection, these tests assist in fostering their learning. Students can learn from their incorrect choice and reconsider the options and try to select the correct response. Learning from errors is another effective approach of helping students learn in programming (Zhou et al., 2021). It helps students to understand the frequent errors they make while coding and the error messages from the compiler. In this question, the research will provide the feedback immediately and help the students to reflect and enable them to learn from the mistakes.

We first investigated this question with regard to building self-confidence in predicting correct answers in basic concepts of programming. The next question focuses on whether the adaptive formative assessment increases their self-confidence in their ability to identify the common errors and fixing the errors and enhancing their programming abilities. Therefore, the aim is to design an adaptive formative assessment framework to improve their self-confidence in programming learning.

RQ-2: How effective is adaptive formative assessment in facilitating novices' understanding of distinct conceptual components in programming? This research expands on the materials already available in the formative assessment quizzes, which mostly cover fundamental programming concepts. Independent programming components can cause more challenges for novice programmers (Luxton-Reilly et al., 2017). This research has developed formative self-test quizzes to expose learners to common errors in each programming concept. The contributions of this work includes an investigation into targeted error messages that novice programmers encounter in the introductory programming. Next, we investigate the differences in their degree of confidence across various concepts and trends between different student categories.

RQ-3: To what extent can adaptive formative assessment encourage novice learners to improve their programming skills? The adaptive approach allows students to learn from their mistakes and provides the option to retry with another attempt until they submit a right answer. This approach thereby maximises their understanding of independent components by splitting large questions into smaller ones. The research method is applying the adaptive assessment technique to assess the novice students at their skill level and encourage them to learn to achieve the required knowledge. Therefore, the aim is to design a formative assessment framework with adaptive strategy to investigate how effectively the adaptive formative assessment improves their performance in relation to various elements.

3.5.1 Addressing techniques

This study intends to investigate the scaffolding and support that formative assessment quizzes might provide for students learning to program. In order to examine the performance of the suggested adaptive formative assessment in real time, we established metrics to measure the efficacy of the suggested approach and the efficiency of the adaptive strategies.

Self-confidence: After using the instrument, students' responses can be used to gauge their level of self-confidence (Charles & Gwilliam, 2023; Hogan et al., 2023). The self-confidence levels of novices are measured following their completion of the adaptive formative assessments. After completing the preliminary experiments in three adaptive models, we used an ANOVA test to compare their responses to answerRQ-1a and RQ-1b (Moon et al., 2022; Tisza et al., 2023). We also used paired t-test of the pre and post confident responses as shown in Table 3.6.

Research Question	**Instrument/Data Source**	**Details**
RQ1a, RQ1b	Pre/post self-confidence surveys and predictive quizzes	Students will rate their confidence in predicting code output and correcting errors before and after AFA interventions.
RQ2	Concept-by-concept analysis	Student responses will be categorised by programming concepts (e.g., loops, conditionals) to assess learning patterns.
RQ3	Skill progression analysis from quiz	Track changes in performance and error frequency over time through platform analytics.

Table 3.6: Research questions summary

Effectiveness: The quality of novices' performance following the assessments is used as an indicator of the adaptive formative assessment's effectiveness (Vesin et al., 2022). In-depth analyses were performed on the self-rated confidence, term-end test results, completion time, and correlations. Comparing their performance across

groups allows for the measurement of effectiveness (Dirzyte et al., 2023; Hogan et al., 2023). We employed concept-by-concept heat map and line chart analysis to identify learning tendencies among students (RQ-2). We compared the grades of students who attempted and those who did not in order to assess the quality of the novices' grades (RQ-3). We used paired t-test of the term-end exam scores in order to find out the effectiveness of the system as shown in Table 3.6 (Charles & Gwilliam, 2023; Dirzyte et al., 2023).

3.6 Conclusion

As a conclusion, this chapter presented on how to use an adaptive strategy in the process of formative assessment, in order to especially motivate novice programmers and increase their knowledge and self-confidence. This chapter discussed the several forms of formative assessment and its significance in learning programming. Furthermore, this chapter outlined the research questions in detail that this study is examining. This study's next focus is to develop a research methodology to evaluate the proposed system and evaluate the outcomes.

Chapter 4

Research Methodology and Data Collection Strategy

4.1 Introduction

According to (Saunders et al., 2019) the process of designing a study is like layer-by-layer onion peeling. The researcher must make decisions for each layer of the "onion" research design, and each decision impacts the one that comes after. Together with the steps for gathering and analysing data, the methodology also covers the researchers' theoretical framework, procedural definitions, research hypotheses and population. This chapter describes the research methodology along with the research workflow that we employ. It details the research design, participants, instruments, data collection procedures, and data analysis strategies.

4.1.1 Research design

A quasi-experimental mixed-methods design was used, combining both quantitative and qualitative approaches (Taherdoost, 2022). This design enables an in-depth understanding of the effectiveness of adaptive formative assessments while capturing the subjective experiences and perceptions of participants regarding their self-confidence development. As an instrumental case study, this research developed

a set of quizzes for basic topics of introduction to programming. We conducted these assessment quizzes periodically during teaching sessions to build novices' confidence as well as to capture their barriers in programming. We conducted a survey about how it effectively helped them to learn programming at the end of the quizzes. The summary of the intervention is shown in Table 4.1.

Regular quizzes were offered during the study periods and this gave students the chance to take quizzes and consider what they had learned. For phase-1, a short optional and anonymous survey was employed to get an idea of how learners viewed and experienced the quizzes for introductory programming at the end of the semester. Due to their failure to finish the final quiz as the topic was not covered in their curriculum, several participants did not offer their opinions. Therefore, we were unable to gauge how motivated and confident they felt after each quiz. Consequently, in phase-2 and afterwards, we added survey questions at the conclusion of each assessment to get their feedback. This study offered an optional revision quiz before the exam to determine the effect of the adaptive formative assessment quizzes on exam scores in phase-4.

4.1.2 Data collection strategy: A mixed methods approach

Data collection, according to (Kabir, 2016), is the methodical process of gathering information from relevant sources in order to address research questions, test hypotheses, and achieve the project's objectives. This research is using a Mixed Methods Approach (MMA). Research that entails gathering, evaluating, and interpreting both quantitative and qualitative data in a single study or in a number of studies that look into the same underlying problems is known as MMA research (Mertens, 2019). Both qualitative and quantitative methods were used in this approach. Data was gathered using a survey consisting of both closed-ended and open-ended questions to obtain both qualitative and quantitative data (Mertens, 2019). Numerous studies have examined various methods for improving students' self-confidence (Ling et al., 2021; Strickroth, 2024). Self-rated confidence changes

are a technique they frequently employ to measure in programming learning (Faherty et al., 2021; Hogan et al., 2023; Ling et al., 2021). A survey, which is a research technique that gathers data from "a sample of people through their responses to a topic," was the method this study employed (Check & Schutt, 2012, p. 160). An online survey was used in this study to gather data that was both quantitative and qualitative. It is widely known that surveys are an effective tool for gathering data about the attitudes, beliefs, actions, and characteristics of respondents (Creswell, 2014; Jones et al., 2013; Ling et al., 2021).

The Computing Attitudes Survey (CAS) was used to gauge respondents' level of programming confidence (Bockmon et al., 2020). CAS is a scientific survey that measures changes in attitudes from novice to expert about the nature of knowledge and solving problems in computer science. The respondents were questioned about how they felt about formative assessment quizzes of each programming topic. A Likert scale questionnaire is the most popular and ideal type of scaled questionnaire (McMillan & Schumacher, 2014; Sullivan & Artino, 2013). The survey consists of both closed-ended and open-ended Likert scale questions to obtain both qualitative and quantitative data (Mertens, 2019). This approach works well with a combination of qualitative and quantitative techniques. The Likert scale had five scores: strongly disagree (1), disagree (2), neutral (3), agree (4), and strongly agree (5). Over the CS-specific tool created by Ramalingam and Wiedenbeck, 1998, this scale was selected. In this study, we use quantitative intervention to evaluate validity and reliability of formative assessment quizzes. A questionnaire-based survey was used to examine self-confidence on programming comprehension, common code errors understanding, and other programming learning prospects. It measures their perception about their comprehension of fundamental programming concepts, their level of self-confidence in learning programming, and their capacity to recognize and correct common errors in programming after attending the quizzes. It also includes qualitative elements to measure their performance in the quizzes and makes use of a range of data sources and data collection methods. To measure

the effectiveness of the intervention, a quantitative instrument was created to collect data regarding participants' opinions of formative assessments in learning programming. Assessments of effectiveness include confidence and interest in programming surveys conducted before and after, course grade comparisons, correct answer improvements and retention/continuation rates (Faherty et al., 2021; Hogan et al., 2023; Tisza et al., 2023). Programming self-esteem levels are compared across cohorts using descriptive and inferential statistics (such as ANOVA and t-tests) to determine effectiveness (Moon et al., 2022; Tisza et al., 2023). Every piece of qualitative data and quantitative data is anonymous. Focus group interviews and open-ended questions from the post-intervention survey were used to collect the qualitative data.

4.1.3 Survey questions

Students are asked a series of questions (S1-S9) at the conclusion of the quizzes in order to find answers to the research questions. Critical friends were used to review and refine each question's language and made any necessary revisions for good survey questions that were described as clear and unambiguous (Creswell, 2014; Fisher Jr, 2000). The survey questions were all asked using Google Forms just after the quiz completion. For each of the quantitative survey questions, the response frequencies are arranged and condensed into Likert charts so that the participant feedback may be understood.

4.1.4 The Population

DCU's Introductory programming modules provided the learner cohorts for this investigation. All introductory programming students with no regard to year of study were invited to take part in the course review, which was conducted via an anonymous Google Drive questionnaire. The list of modules are: Computer Programming I, Computer Programming II, Introduction to Programming, Programming Fundamentals I, Computer Programming 4 (Object

Oriented Programming) and Introduction to Programming (Python). A total of 730+ participants answered the survey across all cohorts.

4.1.5 Participants

In order to provide answers to the research questions, the study investigated the quizzes with several cohorts. The research was undertaken with students enrolled in the Introductory Programming Modules during the academic years (2022–2025), and nine different student cohorts as shown in Table 4.1. The study involved a diverse group of participants, including Computer Science CS students, Non-Computer Science (Non-CS) students, and Secondary school children. For ease of interpretation, we represent the cohorts using color coding: CS students, Non-CS students, both CS and Non-CS students and school children. The descriptive statistics tables are presented with these colors to distinguish between groups.

major students were both given the option to take this.

- Cohort C1: First, these quizzes were offered to the Introductory programming students of 2022–23 (sem-1). This cohort consists of both CS and Non-CS students of Introductory programming modules (CA116, CA167a and CA269). Total Number of Responses (N) received are 361.

- Cohort C2: Kahoot version of non-adaptive quizzes offered to the ICT summer camp students. It was held over three-weeks. The same course was repeated each week and provided to three distinct student cohorts. 121 Secondary school students participated in a minimum of one of the camps overall. 119 of those students provided their assent to taking part in the study by parent-signed consent and student assent.

- Cohort C3: The same set of quizzes with adaptive approach (Model-1) were offered to introductory programming students of 2022–23 (sem-2). In Pilot study-2, 40 Non-CS major students participated. 123 responses received in

total.

- Cohort C4: Next, the quizzes featuring Model-2 and 3 were distributed to students studying introductory programming in semester 1 of 2023–2024. In pilot study-3, 125 Non-CS major students participated in model-2. 56 Non-CS major students participated in model-3.

- Cohort C5-C7: Model-3 was selected for main study. It was conducted to the introductory programming students in semester-2 of 2023–2024. Both CS and Non-CS major students participated. Additionally, this model was tried with Secondary school students as well.

- Cohort C8-C9: Model-3 was selected for enhanced main study. It was conducted in semester-2 of 2024–2025. Both Non-CS major students of the introductory programming and CS students of Python programming participated. Additionally, this model was tried with Secondary school students as well.

The inclusion of multiple learner cohorts strengthens the generalisability of the proposed framework by evaluating its adaptive mechanisms under naturally varying educational conditions. Although cohorts differ in prior knowledge and learning behaviors, the framework consistently regulates feedback and task difficulty based on learner interactions without requiring cohort-specific configuration. The replication of effect patterns across cohorts—particularly for key survey dimensions—indicates that the framework generalises across learner populations rather than being tailored to a single cohort. The data comprises the programming quiz attempts of 1200+ students in total, which they submitted at the conclusion of all quiz sessions. The data presented here is both quantitative and qualitative types, covering a three year span (2022–2025) as shown in Table-4.1. The flow chart of the research is shown in Figure 4.1. Student surveys provide quantitative data. The student questionnaires and their reflective writing assignments provide the qualitative and quantitative data. Each piece of qualitative data is anonymous.

Phase	Study/Year	Model	Cohort	Purpose	Research Tool	Statistical Analysis
Phase-1	Pilot study-1 (2022–23 Sem-1)	Non-adaptive	C1 & C2	Quiz questions & Feedback validation	Self-rated Post-quiz Survey questions (S1 & S2)	Descriptive analysis
Phase-2	Pilot study-2 (2022–23 Sem-2)	Model-1	C3	Testing model-1	Self-rated Post-quiz Survey questions (S1-S5)	Descriptive analysis
	Pilot study-3 (2023–24 Sem-1)	Model-2 & Model-3	C4	Testing model-2 & model-3	Self-rated Post-quiz Survey questions (S1-S5)	ANOVA test
Phase-3	Main study-1 (2023–24 Sem-2)	Model-3	C5 , C6 & C7	Main experiment	Self-rated Post-quiz Survey questions (S1-S5)	Descriptive analysis, one sample T test & Qualitative data analysis
Phase-4	Main study-2 (2024–25 Sem-2)	Model-3	C8 & C9	Enhanced main experiment	Self-rated Post-quiz Survey questions (S6-S9), Post-quiz Confidence level in basic concepts (1-16), Pre and post confidence questions over four different factors & Exam score after quiz attempt	ANOVA test, Heatmap analysis, Pearson correlation matrix, paired sample T test & Qualitative data analysis

Table 4.1: Summary of interventions

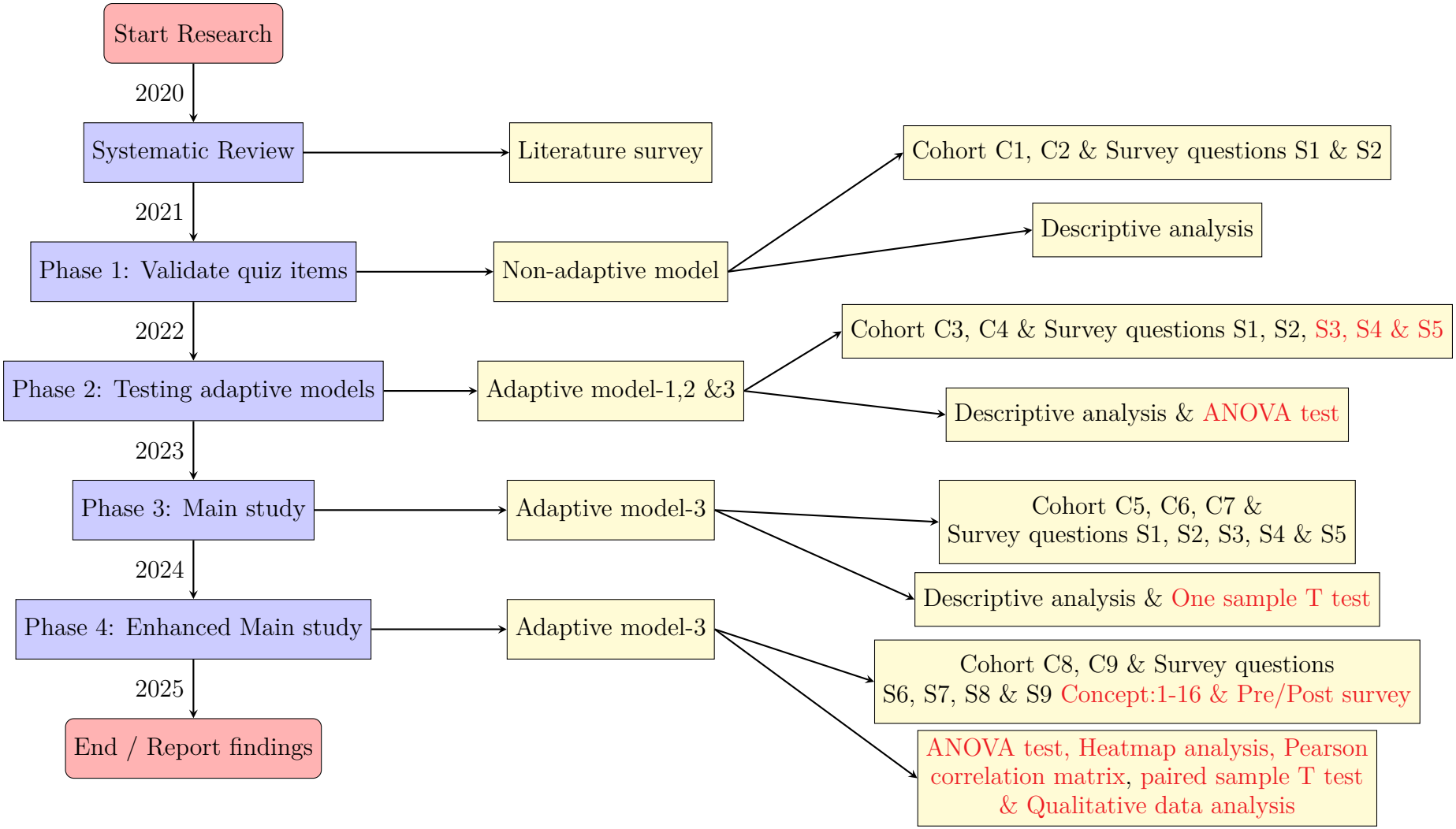


Figure 4.1: Flow chart of the research

4.2 Ethical Considerations

DCU endeavours to ensure that all research carried out by DCU researchers is ethically sound and adheres to the highest standards of research integrity. All research projects must be conducted in accordance with the law, and also according to acceptable ethical standards. This research project was carried out in compliance with DCU's ethical norms, and the research ethics committee granted approval for it after receiving a thorough research ethics submission. This research got ethical approval for 2021-25 from DCU's ethical committee. The reference number is DCUREC/2021/129. Plain language statements, informed consent, and the option to leave the study were all carefully considered and presented during the application procedure.

4.3 Summary

This chapter discussed the strategy used in this research work. In this study, the different adaptive approaches applied to formative quizzes to motivate novices in learning introductory programming. At the end of each quiz, data collected from them to analyse their responses on motivation and self-confidence. There are different survey questions used to collect the data. Detailed participant cohorts were presented. Chapter 5 will present the results of each pilot study.

Chapter 5

Preliminary experiments of Formative Assessment

5.1 Introduction

The preliminary experiment stage of this project is covered in this chapter, with particular emphasis on the formative assessment's prototyping process from non-adaptive to multiple adaptive models. Initially, a non-adaptive model was used in the experiment. In order to accomplish the research goal, it has been enhanced with adaptive models based on the findings. All of the experiments' execution and assessments are covered in detail. In-depth descriptions of the participants, the environment, instrument modifications, design choices, and study findings are included in the richly contextualised discussions. In this way the academic rigor of the study is protected. Recruitment, data analysis, conclusions, and recommendations for instructional design and content are among the issues covered in this chapter. The research design evolved over time. It also relates to assessments, since the purpose of the evaluation is to determine whether the formative quizzes are worthwhile, engaging, high-quality, and practical. The preliminary experiment of several formative evaluations between 2021 and 2023 is presented in this chapter.

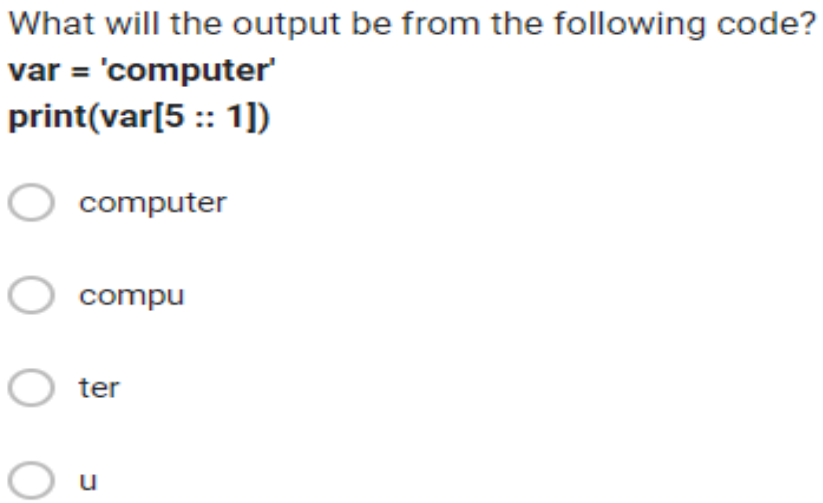


Figure 5.1: A sample question in Google Forms

5.1.1 Quiz implementation

As explained in chapter 3, a set of quizzes was developed on basic topics of introduction to programming. The quizzes are presented using 'Google Forms' as they can be an effective tool for formative assessment and for promoting active learning (Djenno et al., 2022). Each quiz aims to educate students about common code errors they made when studying the assigned topics. A sample question is shown in Figure 5.1. Feedback will be given to students for each potential response, which will help them better comprehend the errors and help them to understand easily as shown in Table 3.2. Feedback is in the form of customised messages that are similar to enhanced error messages (Becker et al., 2016). Every incorrect reaction offers advice on how to respond. Students can make the best alternative response based on the feedback. It may also be possible to reduce some of the stress brought on by receiving unfavourable feedback through employing a constructive criticism technique that offers a standardised style for constructive criticism (Marwan et al., 2022).

5.2 Phase-1: Validating the quizzes with Non-adaptive model

Our initial goal was to validate the quiz questions and answers by incorporating customized feedback from both students and staff, in order to confirm whether the formative assessment quizzes effectively support novices in learning the fundamentals of programming through the provision of instantaneous, enhanced feedback. Consequently, the non-adaptive strategy was used to provide the student groups with frequent quizzes 3.4.2. These quizzes were provided on a regular basis during teaching sessions to help novices gain confidence and identify any obstacles they may have with programming. Students receive feedback for each possible response, which will aid in their understanding of programming fundamentals. The right answer feedback recognises their responses, while the wrong answer feedback assists them in identifying the right one. It is similar to enhanced error messages (Becker & Glanville, 2016). The summary of Phase-1 intervention is described in Table 5.1.

Year/Sem	2022–23 Sem-1
Model	Non-adaptive
Cohort	C1 (Both/U) & C2 (Non-CS/S)
Purpose	Quiz questions & Feedback validation
Research Tool	Self-rated Post-quiz Survey questions (S1 & S2)
Statistical Analysis	Descriptive analysis
Outcome	Positive results on Quiz material

Table 5.1: Phase-1 intervention summary

5.2.1 Cohort-C1 (Both/U)

First, the quizzes were offered to the 'Introductory programming' students of 2021–22 on an optional basis. The topics of the weekly lectures indicated in Table 5.2 were covered in the quizzes that the students took. This cohort consists of both CS and Non-CS students of Introductory programming modules (CA116, CA167a and CA269).

Week 2-3	Week 4-6	Week 7-8
Variables and Operators	Control statements	Functions

Table 5.2: Summary of quiz topics for Cohort-C1

Survey questions:

To answer the research questions, we asked the following survey questions to cohort-C1 students. A short optional and anonymous survey was employed to get an idea of how learners viewed and experienced the quizzes for introductory programming. They completed the survey at the end of each quiz in this study period. The results are discussed in the following section.

S1: Do these quizzes help to understand basic concepts of Programming? The survey question set out to address is: "Do these quizzes help to understand basic concepts of Programming?". The concepts refer to 5.2. To ensure that no one's response was missed, we incorporated a survey question into each quiz. For cohort-C1, the answers range were 'Yes', 'May be' or 'No'. The survey results depicted in Table 5.3 that shows the cohort-C1 students' experience.

Quiz	Quiz-1	Quiz-2	Quiz-3	Overall
Yes	146	59	7	212
May be	43	42	5	90
No	13	27	4	44
Total participants	202	128	16	346

Table 5.3: Data of Cohort-C1 responses for S1

S2: Does formative assessment effectively aid in increasing the self-confidence of novices learning programming? To find the students' self-confidence, we asked this survey question from the cohort-C1 students at the end of each quiz: "Do these quizzes help to increase your self-confidence in learning programming concepts?". The answers range from 'Definitely No' to 'Definitely Yes'. The survey results of the student experience in 2022-23 depicted in Table 5.4.

Response	Quiz-1	Quiz-2	Quiz-3	Overall
Definitely No	16	12	1	29
No	29	15	1	45
Neutral	77	60	6	143
Yes	62	30	6	98
Definitely Yes	24	11	2	37
Total participants	207	128	16	351

Table 5.4: Data of Cohort-C1 responses for S2

Cohort-C1 feedback:

The two results show how these assessments help in learning the foundational concepts of programming. From cohort-C1, there were more yes responses than no responses. Over 60% mentioned the favorable comments to the question from Cohort-C1 too. These findings indicate that these quizzes help in gaining an understanding of the fundamental programming concepts. According to these results, these quizzes helped in increasing their self-confidence in understanding basic concepts of the programming.

5.2.2 Cohort-C2 (Non-CS/S)

These quizzes had been administered as one of the instructional strategies with Information and Communications Technology (ICT) summer camp 2022 to check whether this tool is suitable for Secondary school students. The Schools of Computing at DCU hold their annual ICT Summer Camp for second-level schools. Second-level students, mostly from first year to third year, are invited to attend a week-long course at the DCU Campus, where they learn many new and exciting computer science skills including programming. At the conclusion of each day's session, self-test quizzes were used in addition to presentations, live coding, and programming exercises. 'Kahoot' quizzes were used to offer these questions of above topics to the students attending the ICT summer camp in 2022. At the end of each session, the tutor provided comments on the correct and incorrect response choices. This section describes the informal learning structure, participants, programming teaching methodologies, and coding activities.

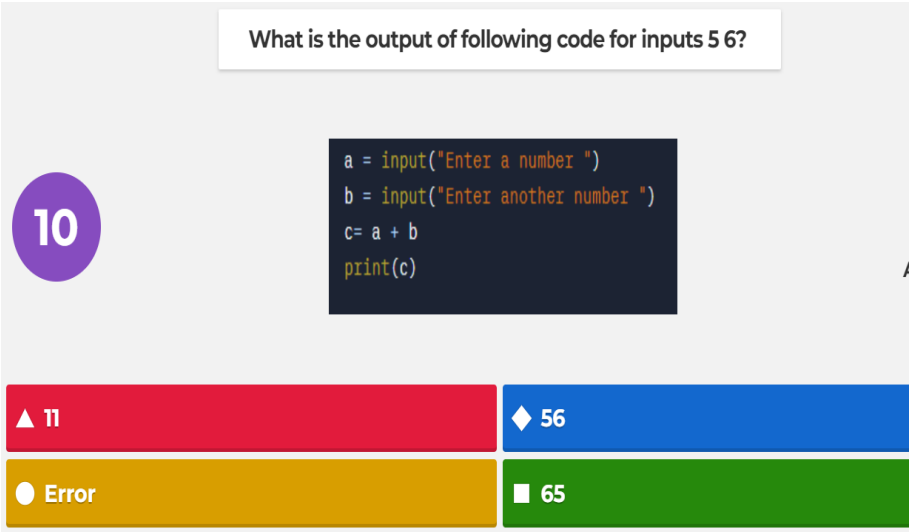


Figure 5.2: A sample 'Kahoot' question

Informal Learning Structure: The setting for our study was a 3-week (15-day) summer camp that included a 5-hour learning session with a programming assignment and an after-class quiz to review the day's coding skills. The learning session included a computer lecturer using presentations to introduce computer concepts, a worked coding example with live coding in Replit, a programming assignment by teaching assistants using CSCircles, highly scaffolded in-class coding activities with programming exercises with the help of student tutors, and a quiz competition at the end. Participants received the introductory code sequences and instructions for each daily live class coding activity, which contained both the necessary steps to finish the activity and suggestions for additional commands to try. The three-week summer camp was offered three times to accommodate student interest. 121 Secondary school students participated in a minimum of one of the camps overall. 119 of those students provided their assent to taking part in the study by parent-signed consent and student assent. The participants were of mixed level of skills and gender as shown in Table 5.5.

	Week-1 (n=41)	**Week-2 (n=40)**	**Week-3 (n=39)**
Gender	Females only	Mixed gender	Mixed gender
Skill level	Novices	Novices	Beginner

Table 5.5: Summary of ICT summer-camp participants

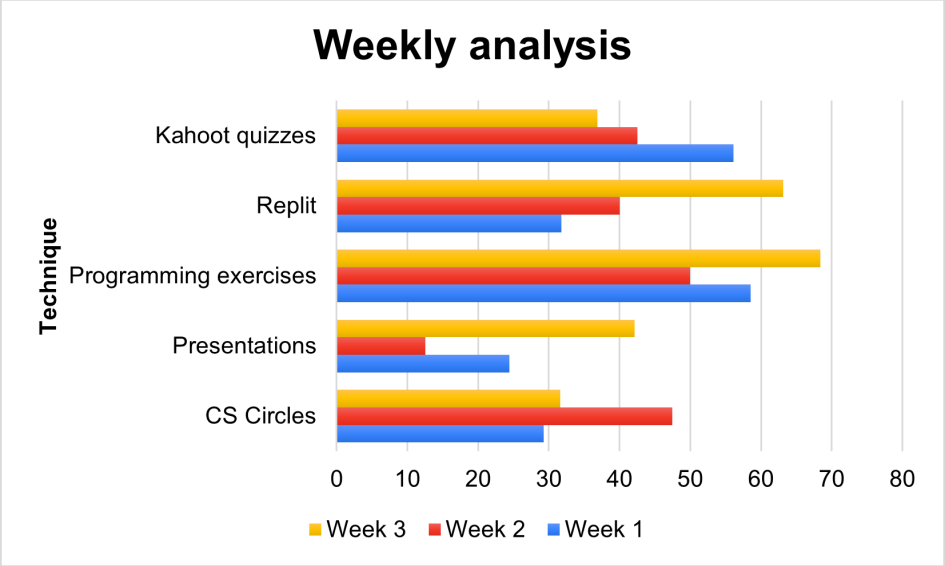


Figure 5.3: ICT camp students' feedback

Introductory quizzes: Kahoot was used for our introductory computing quizzes because of its simple accessibility through a web browser on any platform (King, 2017). After all of the coding assignments, we conducted a one-hour quiz. We implemented the same quizzes of Table 5.2. Every day, the participants were encouraged to take the quizzes to review the concepts for that day. The quiz interface consists of a timeline that displays the timing of each question about basic Python programming. A sample question of the 'Kahoot' quiz is shown in Figure 5.2. After the results were presented, the tutor gave commentary on the right and wrong responses to each question. All students actively participated and the top three performers in the quizzes were appreciated. This research focuses on the analysis of survey data collected at the conclusion of each week, which demonstrates the importance of the teaching technique. All intervention groups completed the survey at the end of the week.

Cohort-C2 Feedback: We asked the students, "Which resources helped you learn programming?". One option included Kahoot quizzes along with programming presentations using 'Replit', and other hands-on exercises. As a result, they chose the quizzes that aided their learning programming as shown in Figure 5.3. We provide our results for the n=41, n=40, and n=39 students in weeks 1, 2, and 3 of an

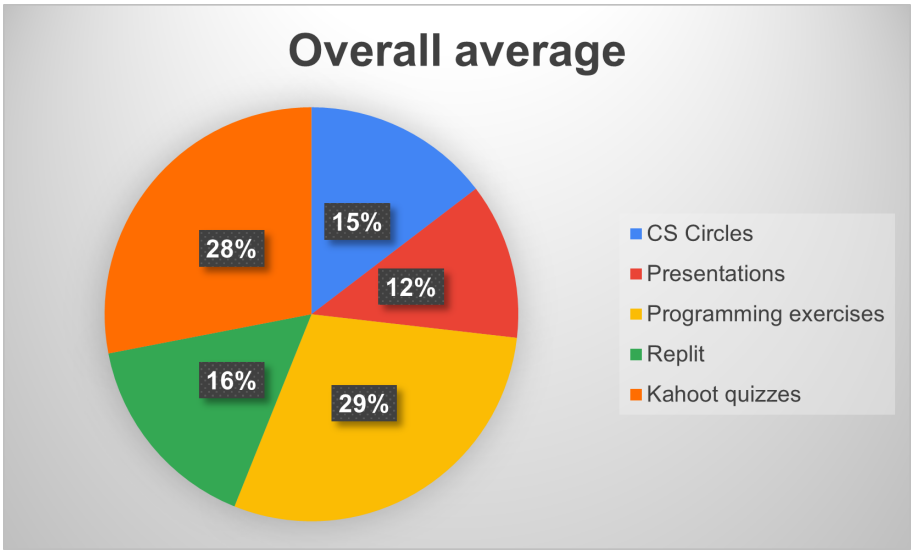


Figure 5.4: Overall recommendation

ICT summer camp in order to better understand which teaching approach promoted programming learning. Our findings, which show the percentages for each technique, are shown in Figure 5.3. In general, the Kahoot quiz preference rate ranged from 39% to 57%. This indicates that the formative assessment method (Kahoot) can be used to teach programming as shown in Figure 5.4. Kahoot quizzes (28%) are followed by the scaffolded programming exercises (29%) in their overall preferences as shown in Figure 5.4. When we divided the data by gender, we discovered that males and females preferred suitable learning strategies. In addition to asking participants about their gender, our pre-camp survey also inquired about their prior experience with programming and interest in computers. Less than 20% of participants in each of the three representative weeks had some prior programming experience (beginner), while roughly 50% had an interest in computers. We discovered some significant variations in the preferences of the different learning strategies. According to these results, novice female students preferred the formative assessment more than the male students with extensive experience. The contention is, in light of these findings, that novices can benefit from formative assessment.

5.2.3 Faculty feedback

Prior to starting the study, we solicited feedback from three academics with backgrounds in computer science and education regarding the quiz's questions and answers along with customised feedback. In addition, we gathered information from faculty members who conducted the phase-1 experiment as well. It provides qualitative data that faculty feel about quizzes aid in improving programming skills. Here F1 & F2 stand for faculty-1 and faculty-2. F1 delivered an introductory programming module with Cohort-C1 students. F2 taught computer programming to second level students. They valued these quizzes for various reasons. List of questions and faculty's responses listed below:

1. *Are the quizzes engaging your students?*

   F1- Yes, I think so. With reminders, students tend to take the quiz every week.

   F2- Python tasks are engaging and completely appropriate for Leaving Cert Computer Science students.

2. *How easy is it to administer them to the students?*

   F1- It is fairly easy. I put the link on the module website. It is just that I need to remind them from time to time.

   F2- Very easy for me to administer to the students – share the link.

3. *Did questionnaires make it hard for students to interpret the question being asked?*

   F1- I haven't got any questions from students regarding the interpretation of the questions being asked, so I think it is not hard for students to interpret them.

   F2- Sometimes the formatting of the questions within the questionnaire made it hard for students to interpret the question being asked.

4. *How did weaker students find it particularly?*

   F1- I don't know particularly how weaker students find the quizzes as they

are all taking them anonymously and no students approached me for further explanation or clarification.

F2- Weaker students found it particularly difficult at times and I had to advise them how to interpret the question.

5. *Was it helpful from your point of view?*

   F1- Yes, I think so. The quizzes are good exercise for students to improve their theoretical knowledge, and this knowledge was tested with a quiz in the mid-term. So, the quizzes are well-aligned with the curriculum of the module.

   F2- That's basically it! Overall, a great exercise in terms of revision, and for me to see what level of understanding students have.

5.2.4 Phase-1: Conclusion

The surveys' responses demonstrate that the students firmly believe the quizzes assisted in grasping fundamental programming principles. Also, students' confidence in learning programming is reportedly much increased by the formative quizzes. ICT camp students supported formative assessment as one of the computer programming learning activities. Based on the faculty surveys' responses, it demonstrates that the faculty firmly believe the quizzes assisted in improving students' programming skills by engaging and reviewing. According to them, the quizzes were a great resource for reviewing students' knowledge in mid-term or prior to the exams. Weaker students, however, require interpretation in order to comprehend the remarks. According to the results, these quizzes largely aid them to improve their programming skills through customised feedback. As a result, this study enhanced the quizzes even more and had them verified with next cohorts.

5.3 Phase-2: Adaptive strategies

The KST is a collection of concepts and frameworks designed to evaluate and depict one's current level of knowledge (Heck & Noventa, 2020). It is crucial to

evaluate people's knowledge in order to determine their ability to complete tasks or to offer assistance on how to become an expert in a particular field (H. Yan, 2020). KST gives the learner a progress evaluation to verify her learning once she has completed a specific number of practice items or after she has worked in the learning mode for a specific period of time since her previous assessment (Cosyn et al., 2021). In KST, a learner's knowledge state is modeled as a subset of items that they can successfully solve, constrained by prerequisite relationships among items (Falmagne et al., 2013). KST has been used in fields like adaptive testing and assessment to support competency-based learning that is self-regulated and personalised (Rong et al., 2023; Steiner et al., 2009). Formative feedback can be given by some adaptive formative assessments, although the majority of them make the assumption that a student's knowledge does not grow during the test (H. Yan, 2020). Personalised adaptive assessment, however, has the potential to improve learning, student engagement, and academic achievement by increasing their self-confidence (Stefanutti & de Chiusole, 2017). This study investigated the adaptive formative assessment using three different strategies to see which adaptive model best increases their self-confidence. The summary of the phase-2 intervention is shown in Table 5.6.

Year/Semester	2022–23 Sem-2	2023–24 Sem-1 (Week 1-5)	2023–24 Sem-1 (Week 6-10)
Models	Model-1	Model-2	Model-3
Cohort	C3 (Non-CS/U)	C4 (Non-CS/U)	C4 (Non-CS/U)
Purpose	Testing adaptive models		
Research Tool	Self-rated Post-quiz Survey questions (S1 - S5)		
Statistical Analysis	ANOVA test		
Outcome	Model-3 is recommended		

Table 5.6: Phase-2 intervention summary

5.3.1 Adaptive model-1

As outlined in 3.4.3, students are asked the similar level questions repeatedly until they select the correct response in this model. Students will receive feedback

for each possible response, which will aid in their understanding of programming fundamentals. Consequently, it enables students to comprehend the independent components and master the knowledge. As used in the previous phase, the right answer feedback recognises their responses, while the wrong answer feedback assists them in identifying the right one. It is comparable to enhanced error messages. These quizzes were provided on a regular basis during teaching sessions to help novices gain confidence, understand the errors and correct them with programming.

Cohort-C3 (Non-CS/U): This model tested the quizzes with Cohort-C3 students. These quizzes were offered to the Introductory programming students of 'Programming Fundamentals' (CA177) module of 2023-24 (Sem-1). The topics of the weekly lectures indicated in Table 5.7 were covered in the quizzes that the students took.

Week - 2	**Week - 5**	**Week - 8**
Print statements	Variables, Operators and Control statements	Functions

Table 5.7: Summary of quiz topics for Cohort-C3

5.3.2 Adaptive model-2

The quizzes were then put to the test using two adaptive models: Adaptive model-2 & Adaptive model-3. Adaptive model-2 methodology uses a four-layer Block model to classify questions according to their degree of difficulty as shown in Table 3.3 (Shute, 2008). In the prior model, the question was repeated until a student provided a right response. To avoid an incessant cycle of same questions, these have been grouped by block model. Basic questions at the atomic level of programming in the block model are asked first. If the student answers correctly, the operation block will be the next question, followed by relations. If not, questions at the same level will be asked as shown in Figure 3.3. Instead of using a brute force approach, the students will master independent components. However, it results in even longer question and answer sessions. Setting up advanced (application) questions is also

challenging when the fundamentals are covered. Therefore, adaptive model-3 was developed.

5.3.3 Adaptive model-3

In adaptive model-3, difficulties in programming are classified as Bloom's taxonomy of programming as outlined in 3.4.3 (Thompson et al., 2008). In this model, we classified a list of questions in three cognitive levels based on the programming complexity (easy, moderate, and difficult) (Louhab et al., 2018). If a student successfully responds to a moderate question on this assessment, the subsequent question is hard. If not, the easy questions will be asked as indicated in Figure 3.4. It goes on until the system forecasts the competency level of the students (Simon-Campbell & Phelan, 2018). A sample classification is described in table 3.4.

Cohort-C4 (Non-CS/U): The quizzes of these two models tested with cohort-4. These quizzes were offered to the Introductory programming students of 'Computer Programming-I' module (CA116). Two quizzes covering the topics listed in Table 5.8 were available in this study. Quiz-1 used adaptive model-2 in weeks 1-5 and quiz-2 with adaptive model-3 in weeks 6-10 of 2023–24 Semester-1.

Quiz	**Quiz-1**	**Quiz-2**
Topics covered	Print statements and Variables	Operators and Control statements
Approach	Adaptive model-2	Adaptive model-3

Table 5.8: Summary of quizzes for Cohort-C4

5.4 Survey questions

We asked the cohort-C3 and cohort-C4 students the following survey question in order to collect their experiences with these models. They completed the survey at the end of each quiz. The results are discussed in the following section.

S1: Do these quizzes help to understand basic concepts of Programming?

The question was set out to address whether these quizzes help to understand basic concepts of Python programming. The answers accepted were 'Yes', 'May be' or 'No'. After a survey of the intervention group, the analysis was done.

Quiz	Quiz-1	Quiz-2	Quiz-3	Overall
Yes	43	32	17	92
May be	8	10	4	22
No	1	4	2	7
Total participants	52	46	23	121

Table 5.9: Data of Cohort-C3 responses for S1

The survey results of the student experience in 2022-23 (sem-1) depicted in Table 5.9. The survey results of Cohort-C4 for the same question are depicted in Table 5.10 to show the results. It proves that both of these models helped them to understand the basic concepts. More than 76% said these quizzes helped them to understand the basic concepts. According to these results, these quizzes help them to understand the basic concepts of programming.

Quiz	Quiz-1	Quiz-2
Approach	Adaptive model-2	Adaptive model-3
Yes	68	80
May be	36	15
No	21	13
Total participants	125	108

Table 5.10: Data of Cohort-C4 responses for S1

S2: Does formative assessment effectively aid in increasing the self-confidence of novices learning programming?

Next, we asked the survey question from the cohort-C2 students to check whether these quizzes help to increase your self-confidence in learning programming. The answers range from 'Definitely No' to 'Definitely Yes'. The survey results of the cohort-C3 students' experience in 2022-23 depicted in Table 5.11.

Response	Quiz-1	Quiz-2	Quiz-3	Overall
Definitely No	1	0	1	2
No	0	3	1	4
Neutral	14	15	7	36
Yes	21	15	10	46
Definitely Yes	15	13	4	32
Total participants	51	46	23	120

Table 5.11: Data of Cohort-C3 responses for S2

Next, the same survey question has been asked of the cohort-C4 students. The survey results of the student experience in 2022-23 depicted in Table 5.12. For this question, students answered 38.33% as 'Yes' and 26.67% as 'Definitely yes' in model-1. It demonstrates that these quizzes increased their self-confidence to understand the basic concepts of the programming when using adaptive model-3 rather than model-2.

Quiz	Quiz-1	Quiz-2
Adaptive	Adaptive model-2	Adaptive model-3
Definitely No	8	4
No	11	10
Neutral	54	26
Yes	30	36
Definitely Yes	21	31
Total participants	124	107

Table 5.12: Data of Cohort-C4 responses for S2

S3: Do these quizzes help students to understand the common errors in programming?

Few additional questions were added to the survey to analyse the instruments effectiveness. The first one is, Do these quizzes help to understand the common errors in python programming?. The answers range from 'Definitely No' to 'Definitely Yes'. The responses of the cohort-C3 students' experience in 2022-23 (sem-1) depicted in Table 5.13. For this question, students answered 40.83% as 'Yes' and 33.33% as 'Definitely yes'. The responses of cohort-C4 are shown in Table 5.14. For this question, students answered more responses for 'Yes' and 'Definitely yes' in

Response	Quiz-1	Quiz-2	Quiz-3	Overall
Definitely No	0	1	1	2
No	2	1	3	6
Neutral	8	10	5	23
Yes	24	18	7	49
Definitely Yes	17	16	7	40
Total participants	51	46	23	120

Table 5.13: Data of Cohort-C3 responses for S3

model-3 than model-2. Therefore, these quizzes helped them better to understand and correct Python programming common errors when using adaptive model-3 than model-2.

Quiz	Quiz-1	Quiz-2
Adaptive	Adaptive model-2	Adaptive model-3
Definitely No	9	3
No	8	5
Neutral	43	20
Yes	38	43
Definitely Yes	26	36
Total participants	124	107

Table 5.14: Data of Cohort-C4 responses for S3

S4: Do these quizzes increase the self-confidence to correct the errors in programming?

We then asked the question: Does this quiz increase your self-confidence in correcting python programming errors?. The answers range from 'Definitely No' to 'Definitely yes'. The survey results depicted in Table 5.15, show the feedback from Cohort-C3. For this question, students answered 33.33% as 'Yes' and 14.17% as 'Definitely yes'. Therefore, in the students' opinion, these quizzes improved their comprehension of typical Python programming errors.

Response	Quiz-1	Quiz-2	Quiz-3	Overall
Definitely No	3	0	0	3
No	13	6	2	21
Neutral	14	15	10	39
Yes	14	18	8	40
Definitely Yes	7	7	3	17
Total participants	51	46	23	120

Table 5.15: Data of Cohort-C3 responses for S4

S5: Feedback provided by this quiz has helped to correct the errors in my program.

In addition, we inquired as to whether the feedback was useful for understanding and determining the right response on the subsequent try. It included the first 'Likert' question, "Does the Feedback provided by this quiz help you to correct the errors in programming?", at the end of the quizzes. The responses ranged from 'Strongly disagree' to 'Strongly agree'.

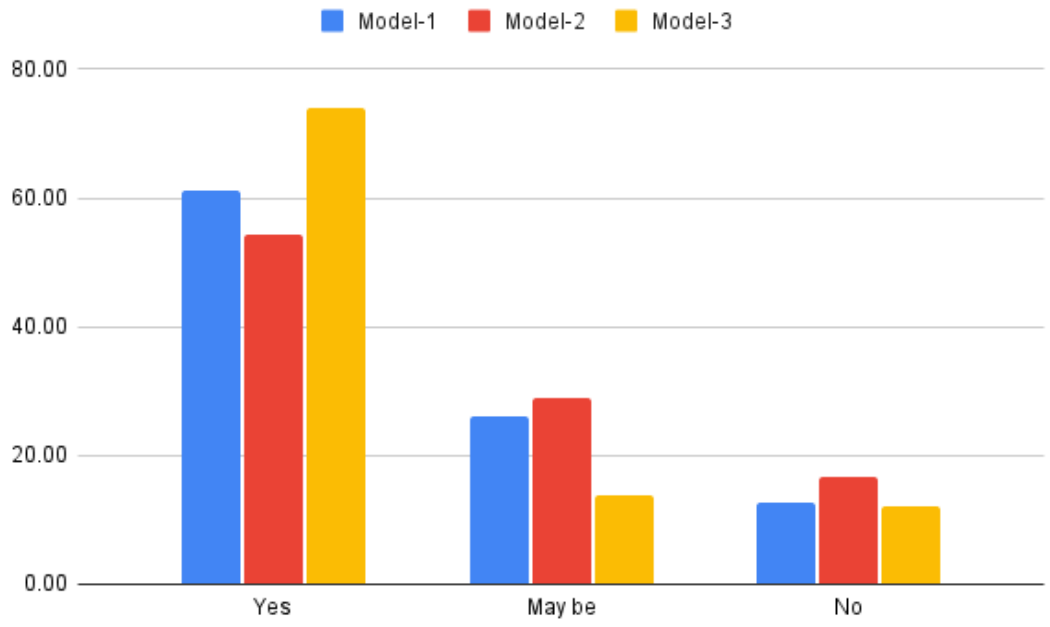


Figure 5.5: All cohort's feedback on learning the Python basics

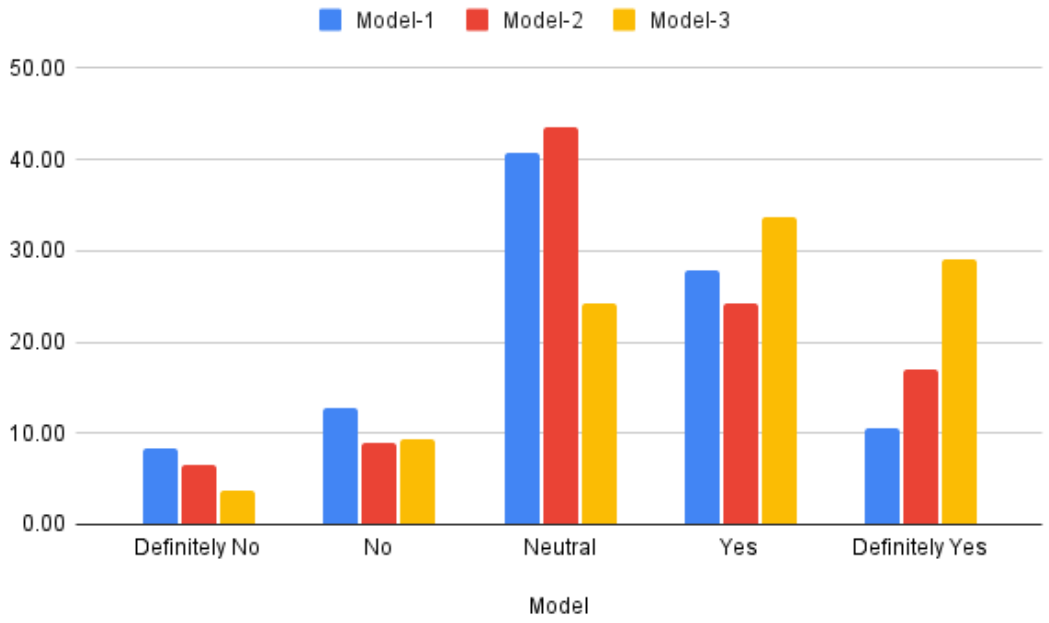


Figure 5.6: All cohort's feedback on self-confidence

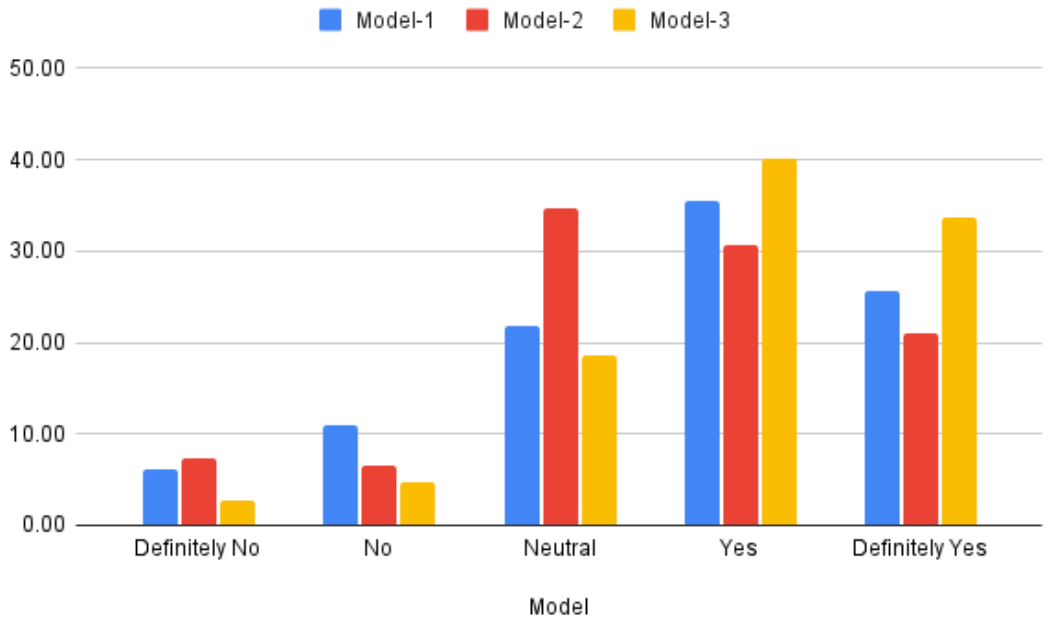


Figure 5.7: All cohort's feedback on understanding errors

5.5 Survey results

All model responses are compared in the Figure 5.5, 5.6 and 5.7. Based on the surveys' responses, it demonstrates that the students firmly believe the quizzes

assisted in grasping fundamental programming principles when using adaptive model-3. Also, students' confidence in learning programming is reportedly much increased by the formative quizzes. In addition, these quizzes improved their comprehension of typical Python programming errors and helped to understand in correcting them. However, to demonstrate that the adaptive model-3 outperforms the others, we conducted statistical analysis. The next section compares all these results using statistical analysis.

5.5.1 Statistical analysis

This section presents the statistical analysis results collected from all preliminary studies. There are three main considerations gathered from the participants: Understanding basic concepts, increasing self-confidence and understanding and correcting common errors in Python programming. At the end of phase-2, we examined every response from these investigations to determine which adaptive model best fits novices' knowledge. For the data analysis SPSS Statistics version 25 software was used. Using one way ANOVA, the impact of adaptive models on their perception was examined (Tisza et al., 2023). This method allows numerous group means to be compared simultaneously while accounting for the inflated Type I error rates that result from repeated pairwise testing (Field, 2017). Table 5.16 shows the result of the analysis. We also plot the outcome charts in Figure 5.6, 5.7 & 5.5 that indicate the influence on understanding basic concepts, increasing self-confidence and comprehension of errors and supporting learning in order to analyse the impact of the adaptive technique. To examine differences among adaptive formative assessment models, a one-way ANOVA was conducted for each survey question. The null hypothesis assumed no significant differences in mean survey scores across Model-1, Model-2, and Model-3, whereas the alternative hypothesis assumed that at least one model would differ significantly.

Survey question	Model	N	Mean	Std. Dev.	95% CI	
					Lower	Upper
S1	Model-1	121	1.70	0.572	1.60	1.81
	Model-2	125	1.38	0.758	1.24	1.51
	Model-3	109	1.62	0.691	1.49	1.76
	Total	355	1.56	0.692	1.49	1.64
S2	Model-1	120	1.65	0.603	1.54	1.76
	Model-2	124	1.56	0.701	1.44	1.69
	Model-3	107	1.69	0.650	1.57	1.82
	Total	351	1.63	0.654	1.56	1.70
S3	Model-1	120	3.99	0.939	3.82	4.16
	Model-2	124	3.52	1.115	3.32	3.71
	Model-3	108	3.98	0.986	3.79	4.17
	Total	352	3.82	1.040	3.71	3.93
S4	Model-1	120	3.39	1.015	3.21	3.58
	Model-2	124	3.01	1.246	2.79	3.23
	Model-3	108	3.20	1.109	2.99	3.42
	Total	352	3.20	1.137	3.08	3.32
S5	Model-1	120	3.85	0.913	3.68	4.02
	Model-2	124	3.36	1.069	3.17	3.55
	Model-3	108	3.75	1.086	3.54	3.96
	Total	352	3.65	1.044	3.54	3.76

Table 5.16: Descriptive Statistical analysis of phase-2 experiment

RQ-1: Can adaptive formative assessment improve self-confidence of novice programmers in

a) accurately predicting the outcomes of fundamental programming concepts?

b) effectively identifying and correcting errors in Python programming?

Survey question	95% CI		η^2	Sig (P)	F
	Lower	Upper			
S1	0.042	0.009	0.087	<0.001	7.735
S2	0.042	0.009	0.087	<0.001	7.671
S3	0.047	0.011	0.093	<0.001	8.583
S4	0.020	0.000	0.054	0.031	3.520
S5	0.007	0.000	0.029	0.317	1.151

Table 5.17: ANOVA effect between the adaptive models

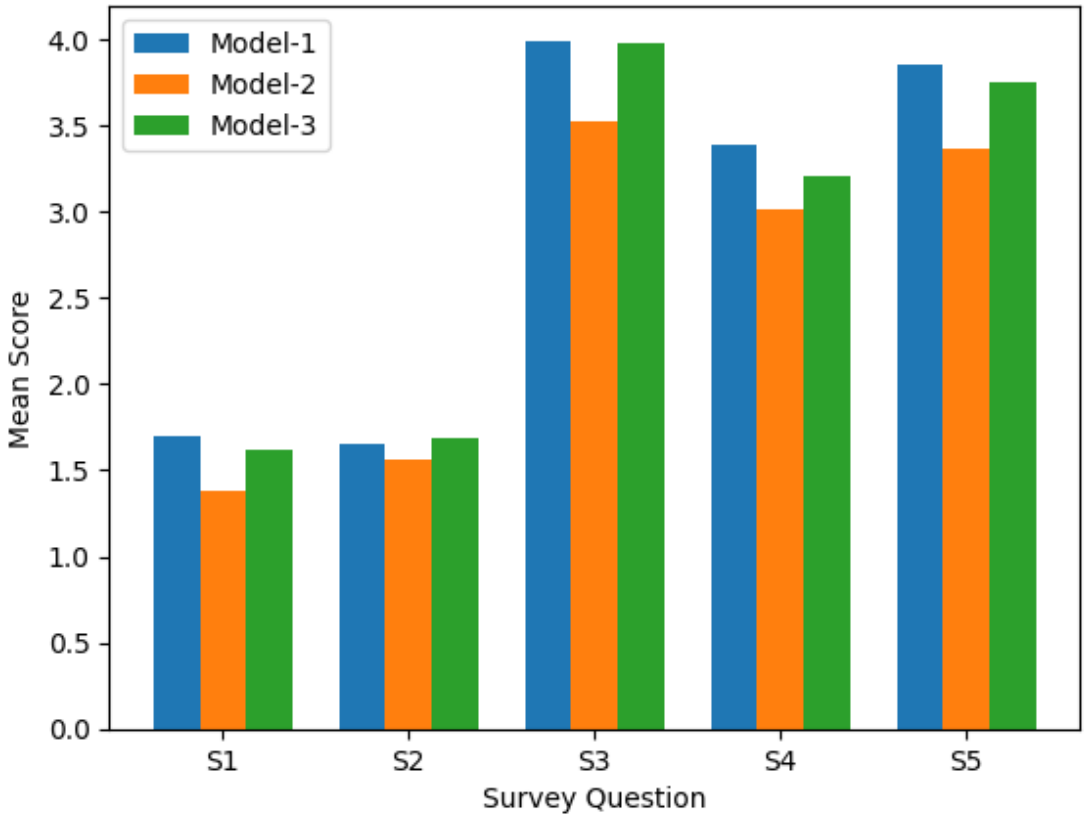


Figure 5.8: Mean difference between adaptive models

Based on the surveys S1 and S2 responses, it demonstrates that the students firmly believe the quizzes significantly assisted in grasping fundamental programming principles as P values less than 0.05. Figure 5.8 depicts a comparison of levels of self-confidence. Students' confidence in learning programming is reportedly much increased by the adaptive model quizzes. The adaptive model-1 and 3 were far more helpful than model-2. It also demonstrates that adaptive assessments helped them understand errors better than non-adaptive ones based on surveys S3 and S4 results. Additionally, students firmly felt that adaptive assessments increased their confidence relative as mean value is over 3.5 for S3 and over 3 for S4. The analysis shows that the mean values for all indicators are positive such as more than 1.5 for S1 and S2; and more than 3 for S3, S4 and S5. The results provided so far indicate that there is a statistically significant difference between these models as shown in Table 5.17. Except for S5, all of the models' variations are significant as the P value is less than 0.05. Analysis of variance revealed significant differences between

models for survey questions S1–S4, with medium effect sizes observed for S1–S3 (η^2 = 0.087–0.093) and a smaller effect for S4 ($\eta^2 = 0.054$). These findings indicate that the model condition accounted for a meaningful proportion of variance in learner responses, particularly for S3, which exhibited the strongest effect (F = 8.583, p < .001). In contrast, no significant differences were found for S5 (F = 1.151, p = .317), suggesting comparable perceptions across models for this dimension. Eta squared variation is at least 2%. Here, P values are less than 0.05 which means they have significant variance in their perceptions over three models.

RQ-2: How effective is adaptive formative assessment in facilitating novices' understanding of distinct conceptual components in programming? In adaptive model-2 quiz, if students did not provide a correct response, the same level of questions were asked. By correctly completing the next or subsequent tries, students will demonstrate their understanding of independent components. We examine the percentage of accurate responses in each attempt for adaptive model-2 in order to respond to the RQ-2. The results represented in Table 5.18 and the corresponding chart displayed in Figure 5.9. The result indicates that students respond to level-2 questions less correctly than level-1 ones. It demonstrates that most of the students (>70%) answered correctly. This result confirms that adaptive model-3 assisted in learning individual components more than adaptive model-2.

Question	**Attempt-1**	**Attempt-2**
Q1	84.8	89.47
Q2	87.2	61.47
Q3	91.12	65.48
Q4	96	64.17
Average	89.78	70.15

Table 5.18: Cohort-C4 correct answers for adaptive model-2 quiz

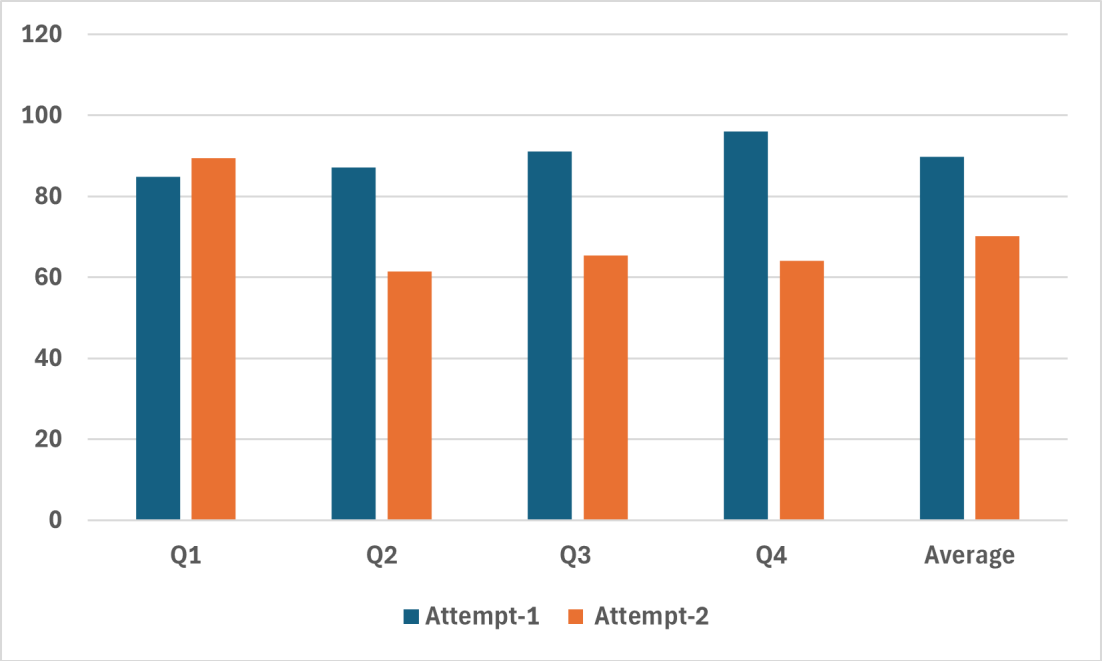


Figure 5.9: Comparison of correct answers between attempts of Model-2

Question	Moderate	High	Low	Summary
Q1	71.90	76.69	86.67	100.00
Q2	65.39		94.59	
Q3	79.39	95.28	81.82	100.00
Q4	97.23	55.70	75.00	100.00
Q5	92.51	87.95		
Q6	96.25	68.86	50.00	
Average	83.78	76.90	77.62	100.00

Table 5.19: Cohort-C4 correct answers for adaptive model-3 (A blank indicates no questions were asked)

RQ-3: To what extent can adaptive formative assessment encourage novice learners to improve their programming skills? Based on the analysis results, all these models helped to increase students' confidence in learning programming by the adaptive formative quizzes. In addition, these quizzes improved their comprehension of typical Python programming errors and helped them to understand how to correct them. Significant differences (P<0.05) exist between these models with S1, S2, S3, and S4. However, the difference (F=1.151) is minimal for S5. In order to determine the impact of models 1 and 3, we used a paired t test. Cohen's d is the most commonly reported effect size when two

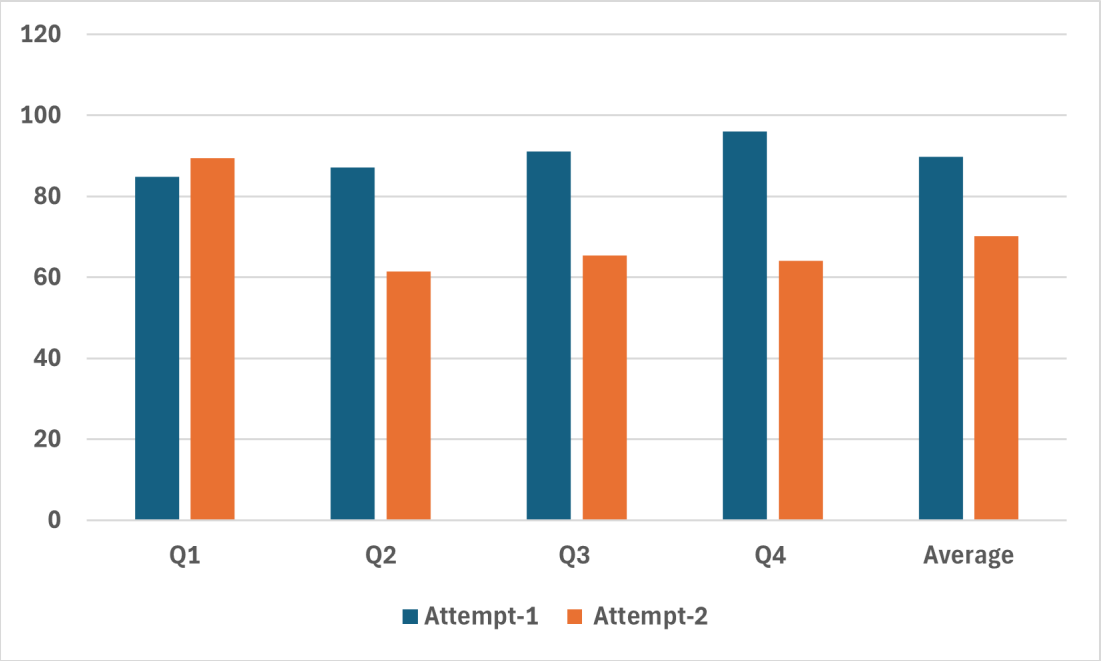


Figure 5.10: Comparison of correct answers between difficulty levels of Model-3

means are compared (Tisza et al., 2023). Cohen's d is 0.125 and 0.100 for S1 and S2 respectively which is the effect size is too small between these models. In addition, critical friends who are conference reviewers when submitting the article have cautioned that adaptive model-1 quizzes operate as a brute force technique that could undermine their confidence in learning programming. As explained earlier, adaptive model-3 uses three different difficulty levels: low, moderate and high. The subsequent question is low if a student answers incorrectly on a moderate question; otherwise, it is high. If they stumble again, they will go to a summary of the specific subject that enables learning about it. Therefore, model-3 maximizes their rate of comprehension by using three levels with summary questions shown in Table 5.19 and the corresponding chart displayed in Figure 5.10. It makes it possible to construct the question to assess present knowledge and generate the subsequent test item until students successfully acquire the necessary knowledge. Therefore, we decided that adaptive model-3 is better for the following reasons with KST:

- It offers three different question types of flexibility so that students of varying skill levels can be engaged.

- According to their responses, it varies greatly in size for students of varying ability. For example, students with excellent grasping abilities finish earlier with lesser attempts than others. Consequently, it engages efficiently.
- Due to its suitability for both basic and advanced topics, it has good curriculum alignment.
- Based on research question outcomes, it is a supporting strategy for students.

5.6 Conclusion

In conclusion, we argue that formative assessment quizzes motivate students to evaluate and learn from their mistakes, which in turn encourages them to learn computer programming, in particular adaptive model-3. As they can effectively aid in the learning of programming by increasing students' confidence and understanding the frequent errors. This research demonstrates that adaptive quizzes, using model-3, help engage and motivate novice programming students, thus improving their programming comprehension. Our next phase involves further iterations of adaptive model-3 quizzes to include closer alignment with the curriculum teaching in order to answer the research questions using deeper analysis.

Chapter 6

Adaptive Formative Assessment Framework

6.1 Introduction

The preliminary experiments have led us to create formative assessment quizzes to introduce common programming errors using adaptive model-3. These quizzes are an effective way to increase student confidence and familiarise more frequent errors in Python programming. These quizzes consist of a list of questions with multiple choice responses. As they offer feedback for each selection, these quizzes assist in fostering learning. Students can learn from their incorrect responses and determine the correct response. As a result, it is a system that progresses and aids in their ability to learn from mistakes. Building on the preliminary results, this phase of the research examines if the adaptive formative assessment increases participants' confidence in their capacity to understand the fundamental ideas behind programming. In-depth descriptions of the participants, the environment, instrument modifications, design choices, and study findings are included in this chapter.

6.2 Phase-3: Main study with Adaptive Model-3

This research's main study examines the adaptive model-3 that is explained in chapter 3. Difficulty levels have been added to learning objects in the model. These learning objects could be topics, questions and a variety of errors. The goals relate to questions with varying degrees of difficulty. Difficulties in programming are classified as Bloom's taxonomy of programming (Thompson et al., 2008). As explained in section 3.4.3, the quiz questions are classified into three cognitive levels based on the complexity (like easy, moderate, and difficult) in this model (Louhab et al., 2018). A sample classification is described in Table 3.4. If a student successfully responds to a moderate question on this assessment, the subsequent question is hard. If not, the easy questions will be asked as indicated in Figure 3.4. It goes on until the system forecasts the competency level of the students (Simon-Campbell & Phelan, 2018).

6.2.1 Questions Development

As mentioned in chapter 3, 'Google Forms' is utilised to implement the quizzes in this study. Each quiz aims to familiarise modular parts of programming and educate students about common code errors they made when studying the assigned topics. Feedback is given to students for each potential response, which will help them better comprehend the errors and help them to understand the programming fundamentals easily. Feedback are customised messages that are similar to enhanced error messages (Becker et al., 2016). Every incorrect reaction offers advice on how to respond. Students can select the best alternative based on the feedback. Details of the questions are shown in Table 6.1. Quiz topics included syntax errors, logical errors, and common misconceptions among novice students (Ahadi et al., 2018; Ettles et al., 2018; McCall & Kölling, 2014). Every quiz has at least five questions of moderate difficulty. Depending on the responses, the question moves automatically from a moderate level to a high level or low level as shown in Figure 3.4. To determine proficiency level, it proceeds to a summary question that explains the appropriate

concept if they are unable to respond to all of the questions. With the help of the lecturer, we conducted these quizzes periodically during teaching sessions to build the novice students' confidence as well as to capture their barriers in programming.

Quiz no	**Topic**	**Modular parts**	**Objectives**
1	Print and operators	Print statement	Familiarising syntax errors in print statement
		Arithmetic operators	Familiarising syntax errors in arithmetic expressions
		Assignment operator	Familiarising syntax errors in assignment
2	Variables, input and operations	Input function	Familiarising variables and type conversion when read a value from keyboard
		Operators and precedence	Familiarising operators and precedence in expressions
		Higher arithmetic operator	Familiarising higher order arithmetic operators (%, //, **)
3	If/Else statements	The if/elif/else structure with comparative operators	Familiarising if / else process and comparative operators
		Operators and precedence	Familiarising comparative expression using AND, OR, NOT
		Errors in If/else	Familiarising syntax error and indentation error in assignment
4	While loop	while loop process	Introducing while loop elements, process before/after increment/decrement
		Introducing multiple while loops and infinitive executions	Familiarising multiple loops and syntax errors and infinitive executions
		Mixed loops	Familiarising multiple loops (while and if/else) and Syntax/logical errors
5	Strings and Functions	String representation	Familiarising different string array representations
		Functions	Familiarising values passing to parameters and syntax errors
		Global and Local variables in function	Familiarising logical errors in assignment of global/local variables

Table 6.1: Summary of questions of each quiz

6.2.2 The Population

In this primary study, first-year undergraduate students from the CS1 module for the 2023–24 academic year participated. In phase-3, our main study tested the quizzes with three student groups (C5, C6 and C7). The data includes 267 students' programming quiz attempts that they submitted at the end of each quiz session. These participants were from both CS and Non-CS major courses. We used three cohorts for our study in Phase-3 to address RQ-1, 2 and 3.

Cohort C5 (Non-CS/U): First, these quizzes are offered to the 'Introductory programming' (CA146) students of CA146 of 2023–24 (sem-2). Throughout the study period, regular quizzes were available to cohort C5. The topics of the weekly lectures indicated in Table 6.2 were covered in the quizzes.

Quiz - 1	**Quiz - 2**	**Quiz - 3**	**Quiz - 4**	**Quiz - 5**
Print and Operators	Variables, input and Operations	If/else/elif statements	While loops	Strings and Functions

Table 6.2: Summary of quiz topics for Cohort-C5, C8 & C9

Cohort C6 (CS/U): These quizzes are offered to the 'Introductory programming' students of CA177 of 2023–24 (sem-2) on an optional basis. The topics were covered using two quizzes with cohort C6 is shown in Table 6.3.

Quiz - 1	**Quiz - 2**	**Quiz - 3**
Operators and If/else/elif statements	While loops	Strings and Functions

Table 6.3: Summary of quiz topics for Cohort-C6

Cohort C7 (Non-CS/S): Furthermore, Secondary school (K12) students studying computer science in leaving certificate are given access to the quizzes. The topics offered are shown in Table 6.4.

Quiz - 1	Quiz - 2	Quiz - 3
Print statements	Input, Variables and Operators	If/else/elif statements

Table 6.4: Summary of quiz topics for Cohort-C7

6.3 Phase-3 Survey Results

In order to collect more thorough data for analysis, this study gave students quizzes at regular intervals. Each quiz contained survey questions in addition to other quiz questions. Following an intervention survey, all analysis was completed. To answer the research questions, we asked the following survey question. They completed the survey at the end of each quiz in Phase-3.

S1: Do formative assessment quizzes help to understand basic concepts of Programming?

The first question set out to address is: **"Do formative assessment quizzes help to understand basic concepts of Programming?"**. The possible answers were 'Yes', 'May be' or 'No'. All three cohorts answered their responses to this question. The statistical results are shown in Table 6.5. It shows the Mean (M) and Standard Deviation (SD) values for each cohort. These results indicate that there is little variation between the cohorts (Kotronoulas et al., 2023). A more effective method would be to view the distribution of likert scale responses using counts or percentages that occur in the different response categories (McMillan & Schumacher, 2014). This can be shown using bar charts (Petrillo et al., 2011). The overall survey results depicted in Figure 6.1 paint a holistic picture of the student experience. More than 66% said these quizzes helped them to understand the basic concepts. C5 students answered 66.15% as 'Yes' and C6 & C7 students answered 68.94% and 80% as 'Yes' respectively. According to these results, these quizzes largely help them to understand the basic concepts of the programming. Based on these findings, the majority of cohorts responded in favor of 'Yes'. Therefore, these quizzes helped them to understand the basic concepts of programming.

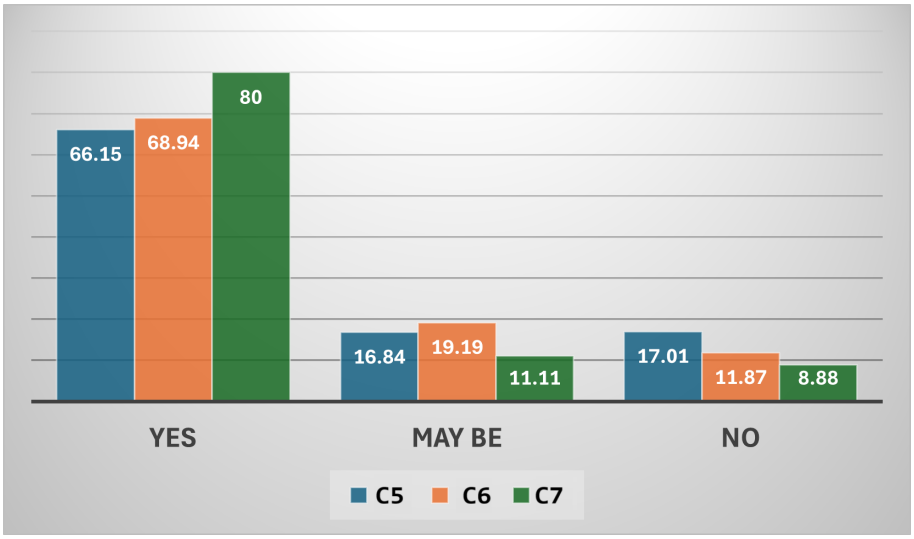


Figure 6.1: Graph of phase-3 cohorts' responses for S1

Cohort	No: 1	May be: 2	Yes: 3	M	SD
C5 (Non-CS/U)	40	45	182	2.53	0.74
C6 (CS/U)	3	5	25	2.67	0.64
C7 (CS/S)	4	4	32	2.7	0.64

Table 6.5: Statistical results of all cohorts' S1 responses

S2: Is the feedback you receive for each question helpful in finding the correct answers?

To answer RQ-1a, we asked the following survey question: **"Is the feedback you receive for each question helpful in finding the correct answers?"**. The answers range from 'Definitely No' to 'Definitely yes'. The survey results depicted in Figure 6.2, paint a holistic picture of the student experience in 2022-23. All three cohorts answered this question. The survey results depicted in Figure 6.2. For this question, C5 & C6 students answered more favourably to 'Definitely Yes' (23.56% & 32.07%) and 'Yes' (30.39% & 34.86%) as shown in Figure 6.2. However, C7 students answered less (11.02% & 29.09%) than other cohorts to 'Yes' parts. The statistical results are shown in Table 6.6. For those majoring in CS (C6), the variation is especially small. Additionally, some neutral responses are received here. These responses may arise from their inability to assess their own learning or from differences in expertise. According to statistical results, these quizzes increased their self confidence in learning programming.

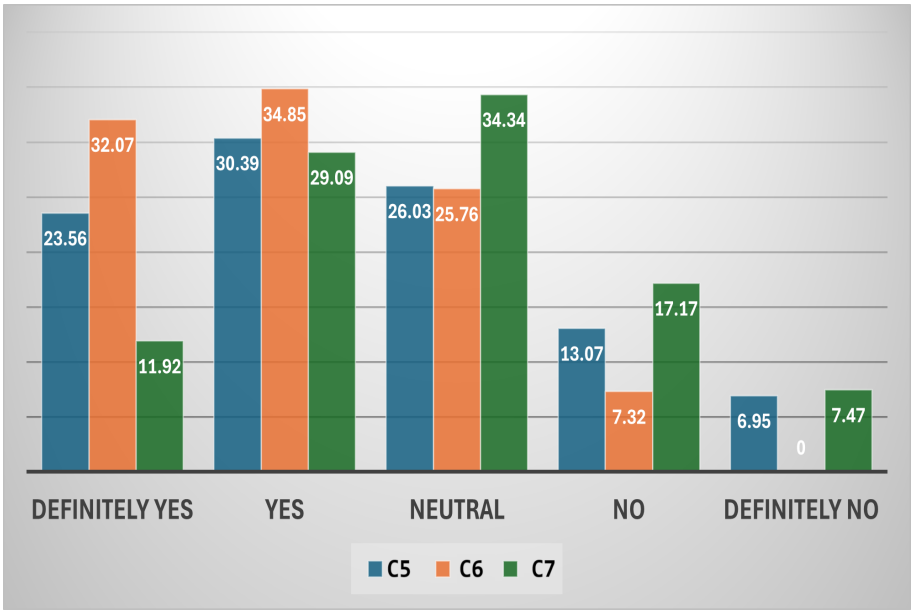


Figure 6.2: Graph of phase-3 cohorts' responses for S2

Cohort	Strongly Disagree:1	Disagree: 2	Neutral: 3	Agree: 4	Strongly Agree:5	M	SD
C5 (Non-CS/U)	17	31	72	86	59	3.52	1.15
C6 (CS/U)	0	2	8	12	11	3.97	0.90
C7 (CS/S)	3	7	14	12	5	3.22	1.09

Table 6.6: Statistical results of all cohorts' S2 responses

S3: Does formative assessment act as an additional method of learning programming?

Finding out what students think and feel about formative assessment as a computer programming learning activity is critical. An open-ended Likert question has been posed to elicit their thoughts regarding the quiz's efficacy as an educational tool that gets qualitative information. According to their feedback, we divided the data into three main categories as shown in Table 6.7. According to quality data analysis of this feedback, they delighted in gaining knowledge by taking quizzes in adaptive models and they also valued these quizzes for various reasons as stated in the comments. There were some slightly negative comments, however the majority were positive. Students value the quizzes to be useful for learning, reviewing, and presenting new concepts of programming that they may not have even learned yet. Negative comments inspired us to improve the quizzes. By establishing difficulty

levels, we shortened the length of the quizzes and added more detailed feedback. In order to keep their confidence, we first gave them feedback with 'Well tried' comments. However, students believed that it was unclear if the response was correct or not. Consequently, we indicated correct or incorrect, accordingly.

Category	**All cohorts' qualitative feedback**
Learning tool	*...Very useful tool for learning. Cleared up a lot of my misunderstandings (C5) ...made me realise what I didn't know...very helpful exercises (C4)...I think they are much better than the way the lectures are being taught...Maybe do the quizzes in the lectures to fully understand what is being taught...(C4)*
Helpful for revision	*.....Nice as a checkup on my current ability (C6) ..It helped to recall..was good to refresh my brain (C2).. Good one for improvement.. this quiz is helpful to recollect my knowledge on Python.. (C6).. Actually this is the perfect way of practising in a competitive way..plz keep these tests regularly (C6)...*
Introduces new concepts	*.....introduced me to new elements of python.....I think they are much better than the way the lectures are being taught. Maybe do the quizzes in the lectures to fully understand what is being taught (C5).. good & interesting question (C6)*
Other comments (including negative)	*.....Expand on the explanation's a little bit more. (C4).. Too long (C4)... increase the difficulty level (C5).. The images in the last question are uncaptioned, meaning I as a blind student could not access them (C5).. Sometimes the feedback can be confusing. Saying "well tried" after giving the correct answer makes me feel like the answer that I gave is wrong (C5).."Good effort" is too vague of a response, unclear if answer is correct or not. Stick to clear responses i.e. "correct" / "incorrect" (C6).. After the option is selected no back option and recorrect the answers (C5).*

Table 6.7: Students' feedback for S3

S4: Do formative assessment quizzes help students to understand the common errors and to correct these errors in Python programming?

To address RQ-1b, we have included another survey question as, **Do these quizzes help to understand the common errors and correct them in python programming?**. The answers range from 'Definitely No' to 'Definitely Yes'.All

three cohorts answered their responses to this question. The overall survey results depicted in Figure 6.3. For this question, all cohort students answered more favourable to 'Definitely Yes' and 'Yes' as shown in Figure 6.2. C5 answered 26.38% and 33.22% as 'Definitely Yes' and 'Yes' respectively. Similarly, C6 answered 30.3% & 43.94% and C7 answered 19.06% & 25.12% respectively for 'Yes' parts. When comparing, C7 students answered less than other cohorts to 'Yes' parts. The statistical results are shown in Table 6.8. Here, the mean values are between 3 and 4.03, indicating that the cohorts agree that the quizzes helped in understanding and correcting the errors. According to these results, these quizzes improved their comprehension of typical Python programming errors.

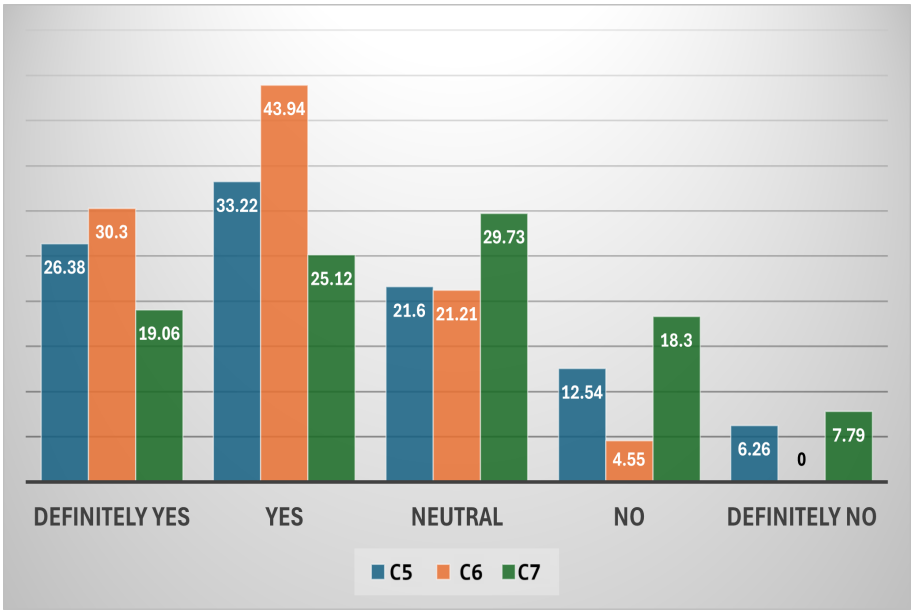


Figure 6.3: Graph of phase-3 cohorts' responses for S4

Cohort	Strongly Disagree:1	Disagree: 2	Neutral: 3	Agree: 4	Strongly Agree:5	M	SD
C5 (Non-CS/U)	16	29	58	94	68	3.64	1.15
C6 (CS/U)	0	1	7	15	10	4.03	0.80
C5 (CS/S)	3	7	12	10	8	3.32	1.19

Table 6.8: Statistical results of phase-3 cohorts' S4 responses

S5: Does formative assessment effectively aid in improving programming skills?

To address RQ-2, we set a question: **"Is the feedback you receive for each question helpful in finding the correct answers?"**. The answers accepted were 'Yes', 'May be' or 'No'. After a survey of the intervention group, the analysis was done. All three cohorts answered their responses to this question. The overall survey results depicted in Figure 6.4 paint a holistic picture of the student experience. More than 66% said these quizzes helped them to understand the basic concepts. C5 students answered 66.52% as 'Yes' and C6 & C7 students answered 73.48% & 54.55% as 'Yes' respectively. Additionally, statistical findings clearly demonstrate that the quizzes improved their programming skills. The statistical results are shown in Table 6.9. These results indicate that there is little variation between the cohorts. Furthermore, these quizzes helped them to understand the basic concepts of programming.

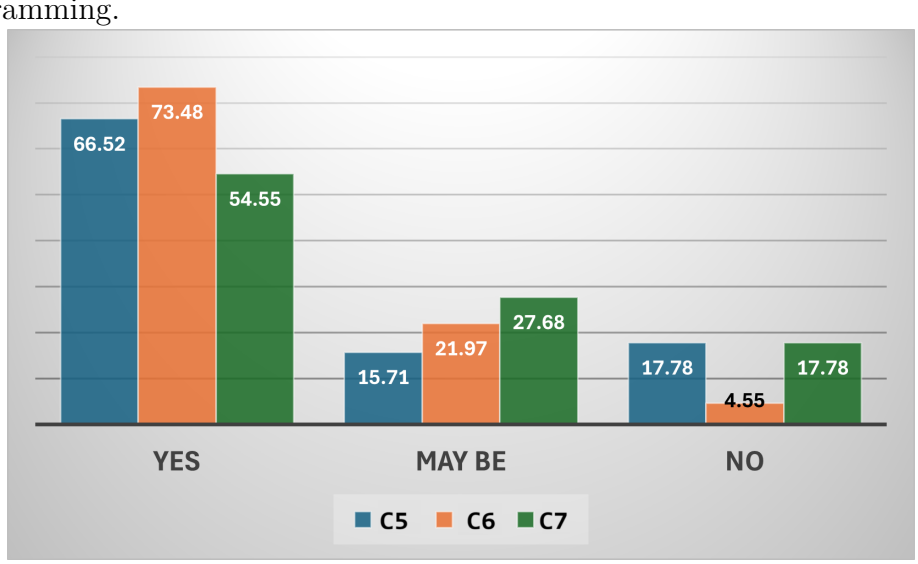


Figure 6.4: Graph of phase-3 cohorts' responses for S5

Cohort	No: 1	May be: 2	Yes: 3	M	SD
C5 (Non-CS/U)	43	39	182	2.53	0.76
C6 (CS/U)	1	7	25	2.73	0.51
C7 (CS/S)	8	11	22	2.34	0.78

Table 6.9: Statistical results of Phase-3 cohorts' S5 responses

6.4 Phase-4: Enhanced main study

Using the same instrument, we enhanced our study in this phase by including more comprehensive survey questions to elicit additional user insights. We added more survey questions to the same quizzes that addressed the topics in Table 6.1.

6.4.1 The Population

In this phase, undergraduate students from the CS1 module and advanced module of academic year 2024-25 participated. To find further insights, we administered the same set of quizzes to two additional cohorts (C8 & C9) in Phase-4. The data includes 252 students' programming quiz attempts that they submitted at the end of each quiz session.

Cohort C8 (Non-CS/U): The same quizzes are offered to the 'Introductory programming' (Non-CS) students of CSC1008 of 2024–25 (sem-2). Throughout the study period, regular quizzes were available to cohort C8 with the same topics as indicated in Table 6.2.

Cohort C9 (CS/U): The same quizzes are offered to the 'Advanced programming' (CS) students of 2024–25 (sem-2). These quizzes of the same topics in Table 6.2 offered as a bridging element at the beginning of the course at the end of each teaching session.

6.5 Phase-4 Survey Results

In addition to quiz questions, each quiz included survey questions. All analysis was finished after the phase-4 experiment. A total of five quizzes were administered, with 77 students attempting the most. A self-rated confidence survey was completed by 62 students (80%) both before and after the intervention. At the conclusion of each Phase-4 quiz, they filled out the survey. We posed a number of survey questions in

order to address the research questions. Four different factors were measured by a survey consisting of 21 items with use of the post-survey attitude of self-confidence.

- Self-confidence in programming understanding (16 items in total)
- Self-confidence of understanding common code errors (3 items in each quiz)
- Quiz difficulty level and Gender

S6: Do formative assessment quizzes help to understand basic concepts of Programming? At the end of each quiz, the first three to four survey questions set out to determine whether the novices can comprehend the modular components of programming as shown in Table 6.10. Using a self-rated Likert scale, the responses were accepted that had five possible scores: strongly disagree (1), disagree(2), neutral(3), agree(4), and strongly agree(5). The C8 and C9 students completed five quizzes and provided a self-rated confidence response of 252 (82.5%) out of 292 attempts. Their distribution of confidence across sixteen topics shown in Figure 6.5. These findings indicate that they feel more confident after they have completed the adaptive formative assessment quizzes covering the fundamental subjects with mode values of 3 or higher. However, the self-rated confidence declines over the advanced topics of programming such as functions as shown in Figure 6.5. The fact that C9 students' mode value is higher than C8 students' indicates that it benefited CS students more.

In order to extract gender-based trends, we used Python to generate an appropriate heatmap from the student feedback. Figure 6.6 illustrates how all students, regardless of gender, feel more confidence in the subjects. Consequently, Concepts 1, 2, 3, 8, and 9 were rated particularly highly ($\geq$4.3) by non-binary students, whereas Concepts 1 and 8 were rated positively by male students. However, both the male and female groups were comparatively less confident in Concepts 10, 14, and 15. In general, mid-to-high ratings were more constant for male and female students (usually ranging from 3 to 4.5), whereas non-binary responses showed higher fluctuation between topics.

No	**Question** (I know..)	C8 Mode	C9 Mode
1	how to use print statements in Python programming	5	5
2	how to use the Python programming variables.	4	5
3	what the operators +, -, *, /, >,<, = mean in Python programming.	3	4
4	how to use input functions and arithmetic operations in Python programming.	3	4
5	how to use variables and their type conversions in Python programming.	3	4
6	the orders of operator precedence in Python programming.	3	5
7	what the comparative operators (>,<, >=, <=, == & !==) mean in Python programming.	5	5
8	how to use if/else statements in Python programming.	5	5
9	the syntax and indentation errors while using if/else statements in Python programming.	3	4
10	what the logical operators (AND, OR, NOT) mean in Python programming.	5	4
11	how to use a while loop in Python programming.	5	5
12	what process before or after increment or decrement happens in the while loop of Python programming.	5	5
13	how to use functions in Python programming?	3	5
14	how arguments work functions of Python programming.	3	5
15	how to pass arguments in function calls of Python programming.	3	4
16	what are default arguments in functions of Python programming.	3	4

Table 6.10: Novices Likert scale responses on Python basic concepts

S7: Do formative assessment quizzes help students to understand the common errors and to correct these errors in Python programming? Three questions about their confidence on comprehension of errors and how to correct them were asked using the following three statements in this phase: Feedback provided by the quiz has helped to understand the errors, increase the confidence in learning programming, and better understand common errors and correct them in Python programming. Following each quiz, a total of 246 responses from cohort-C8 were obtained. The responses ranged from 'Strongly disagree'(1) to 'Strongly agree'(5). Table 6.11 displays the results of one sample t test. According to the One-Sample T test, the 95% confidence interval falls between 3.30 and 3.56 for understanding common code errors. Also, it falls between 3.28 & 3.55 and 3.29 &

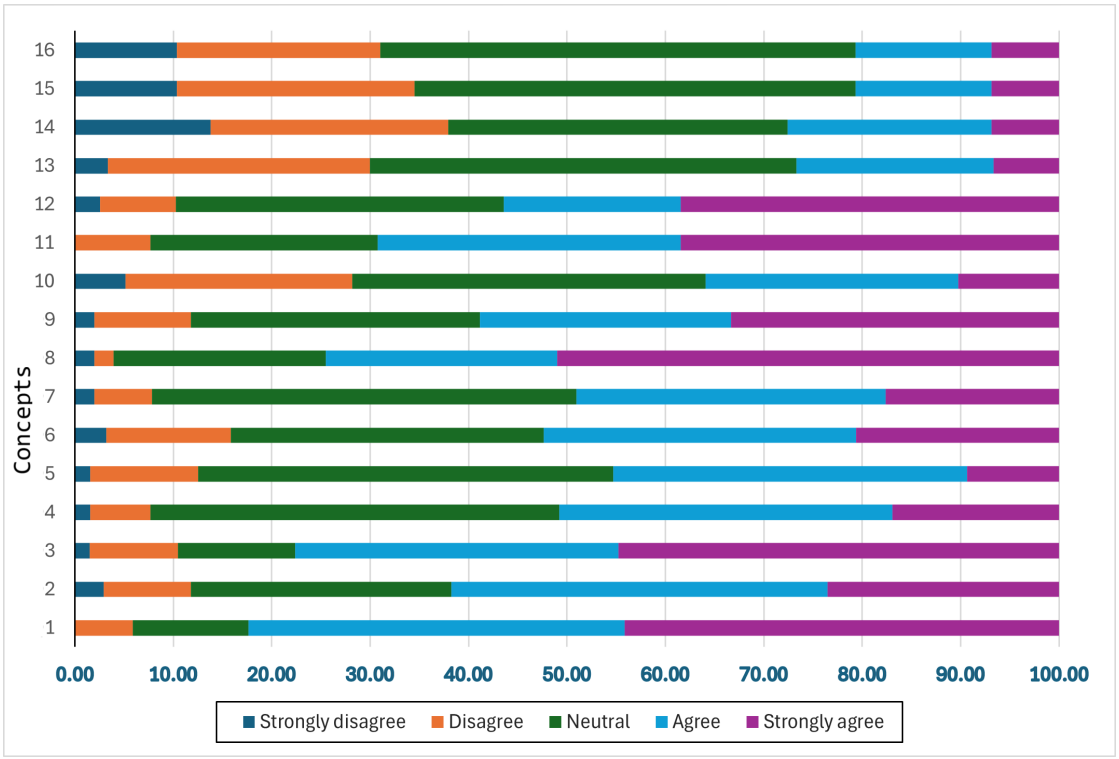


Figure 6.5: All students' feedback (in %) on understanding errors

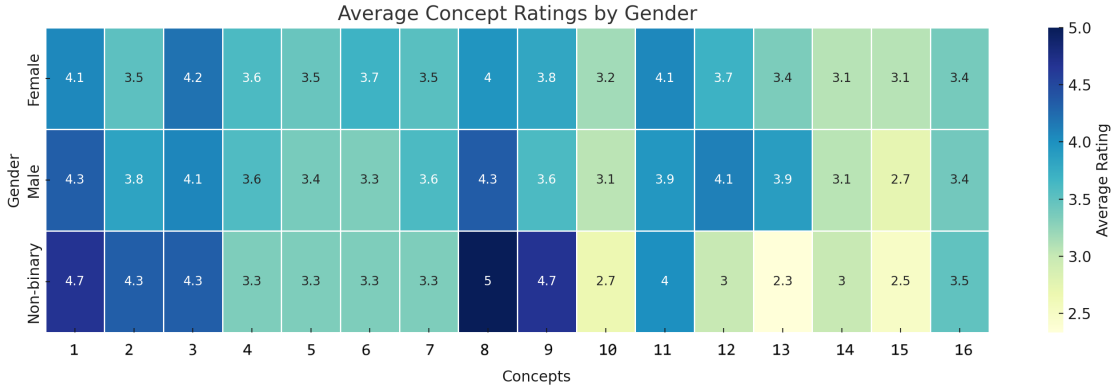


Figure 6.6: Average confidence ratings for each of the 16 concepts, segmented by gender

3.56 to correct the common errors and increase the confidence in recognizing and fixing common code errors. Additionally, the same set of quizzes were given to students in cohort-C9 as a bridging activity because they are advanced students. A maximum of 82 answers were obtained. We received responses with M>2.94 for the 3-scale and M>4.3 for the 5-scale questions. These findings indicate that the adaptive formative assessment quizzes improved their understanding of common code errors as mean value is between 3 and 4. These outcomes show that the quizzes

aid in their understanding of the common code errors of Python programming.

Statement	**C8** (Non-CS/U)		**C9** (CS/U)	
	Mean	SD	Mean	SD
Feedback provided by this quiz helped to understand the common errors	3.43	1.046	2.95	.222
Feedback provided by this quiz helped to correct the common errors	3.41	1.057	4.33	.890
This quiz increased the confidence in recognizing and fixing common code errors	3.43	1.096	4.43	.847

Table 6.11: Descriptive analysis of C8 and C9 responses on Python common errors

S8: How do you rate this formative assessment quiz? Following each quiz, the next question about the quiz's difficulty was asked. A total of 250 responses were obtained. The responses ranged from 'Very easy'(1) to 'Very hard'(5). The responses from students to the quiz's difficulty are shown in Table 6.12. All quiz difficulty levels range from 3 to 4, which most students find challenging. According to the descriptive analysis, students in Cohort-C8 perceived the quizzes as less difficult compared to students in Cohort-C9. According to gender, female students (M=3.27) have more difficulty than male (M=3.13) or other (M=3.00) students in Cohort C8.

Quiz	**C8** (Non-CS/U)		**C9**(CS/U)	
	Mean	SD	Mean	SD
1	3.07	1.297	4.33	0.985
2	3.16	0.902	3.88	1.42
3	3.24	0.885	4.00	1.21
4	3.32	0.842	3.93	1.43
5	3.27	0.828	3.77	1.09
Total	**3.19**	**1.002**	**3.96**	**1.26**

Table 6.12: Descriptive statistics for students' rate on quiz difficulty

Using Pearson correlation in SPSS, we examined the correlation between the difficulty rate of the quizzes and their confidence on comprehension of errors in both cohorts. The results are shown in Table 6.13. The r value is approximately 3 between Rate and three feedback, indicating that students who find quizzes more challenging are more likely to have greater confidence rates. Likewise, a

moderate association exists between the themes and confidence. If a student is more comfortable recognizing errors, they will also be more comfortable fixing them in both cohorts.

	Rate	**Feedback1**	**Feedback2**	**Feedback3**
Rate	1			
Feedback1	0.318**	1		
Feedback2	0.322**	0.773**	1	
Feedback3	0.281**	0.762**	0.798**	1
Rate	1			
Feedback1	0.146**	1		
Feedback2	0.035**	0.347**	1	
Feedback3	0.063**	0.384**	0.942**	1

Table 6.13: Pearson correlation matrix between Quiz Rate and Confidence Feedback of C8 and C9. **Correlation is significant at the 0.01 level (2-tailed).

S9: Any other comments or suggestions. At the conclusion of the survey, we asked all cohort students if they had any other comments or recommendations. As discussed in Table 6.7, they provide qualitative data regarding the quizzes. As listed below, we grouped these comments into a few categories (Raj et al., 2018).

1. *Positive Feedback on Learning Value*

   Very useful tool for learning, Nice as a checkup on my current ability, Good quiz, made sense and was informative, A truly perfect to test fundamental knowledge of Python, very helpful for better self-valuation for a beginners, Great test, Nice questions.

   As a conclusion, these comments provide clear pedagogical value. With refinements, the AFA quizzes can be a strong asset for self-paced learning.

2. *Desire for More Help and Interactivity*

   It would be nice if you would help us in class, More interactive, Need more example for practice and if possible, provide task.

   These comments show that some learners seek interactive support or guidance, especially for harder or new material. Instructors or peer support could enhance the experience.

3. *Clarity and Accuracy of Feedback*

   Incorrect! You might be correct. - this doesn't make sense, Make the writing after answering clearer as to if it is wrong or right.

   These comments show that feedback must be precise, contextual, and error-free. *Nearly there* or *good try* were what we used. Students are confused by these, whether they are right or wrong. Therefore, we had to get rid of ambiguous or conflicting phrases and we adjusted the latter version as correct or incorrect accordingly.

4. *Ambiguity in Questions and Answer Expectations*

   It had 'If, if, else, else' while we had used 'if, elif, elif, else', Didn't know that float was short for 'floating point'.

   Learners expressed confusion over what the quiz was asking, especially in logic/code structure. Therefore, questions should be clearly worded, align with prior learning, and explain new terms when introduced.

Apart from them, students also informed us of a few technical issues, such unclear feedback. However, these quizzes assisted them in reviewing their prior knowledge and preparing for the course's advanced level according to their responses. According to these findings, these quizzes were beneficial for review, introduced new concepts, and received positive feedback on their learning value in a pedagogical approach.

6.6 Addressing Research Questions

My thesis focuses on proving the hypothesis which is *"Using an adaptive approach to formative assessment can increase novices' self-confidence in terms of comprehending fundamental concepts and errors in introductory programming"*. Three research questions have been proposed to concrete this hypothesis. In phase-3, the adaptive formative assessments were tested with university-level introductory programming students (Cohort-C5 and C6). In order to determine whether this paradigm is appropriate for use in school curricula, we next tested the outcomes with

Secondary school students (C7). These quizzes were administered to two additional university-level cohorts (Non-CS (C8) and CS (C9)) in phase-4, along with extra survey questions.

6.6.1 Research Question-1

The first research question is, *"Can adaptive formative assessment improve the self-confidence of novice programmers a) accurately predicting the outcomes of fundamental programming concepts b) effectively identifying and correcting errors in Python programming?"*. This question focuses on whether the adaptive formative assessment increases their confidence in their ability to comprehend the basic concepts of programming. This question aims to find the appropriate adaptive formative assessment system to be the baseline in the comparison of the hypothesis. The baseline must include an instructional model in addition to an intuitive user interface in order to function as an adaptive assessment system. In order to answer the first research question, which was discussed in Chapter 4, we developed a formative assessment system and upgraded it with three adaptive models. We developed automated and adaptive quizzes that covered the basics of introductory programming to create efficient formative assessments. The system used an adaptive function to retrieve and return a ranked list of appropriate questions by matching responses to a formative inquiry with customised feedback that was indexed using a database based on difficulty. These formative assessment quizzes motivate students to evaluate and learn from their mistakes, which in turn encourages them to learn computer programming.

This study utilised a formative assessment framework—an adaptive tool aimed at supporting novice programmers in developing confidence. Students from five distinct cohorts participated in the experiments: CS, Non-CS, and Secondary school. As discussed in the results in chapters 5 and 6, all adaptive models increase novices' self-confidence. According to their responses to survey questions S1-S4, all cohorts showed positive feedback from these models. According to the ANOVA test results

Survey question	Cohort	N	Mean	Std. Dev.	95% CI	
					Lower	Upper
S1	C5	270	2.53	0.740	2.44	2.62
	C6	33	2.67	0.645	2.44	2.90
	C7	53	2.70	0.696	2.51	2.89
	C8	251	2.33	0.736	2.24	2.42
	C9	79	2.94	0.293	2.87	3.00
	Total	686	2.52	0.718	2.47	2.58
S2	C5	267	2.53	0.757	2.44	2.62
	C6	33	2.73	0.517	2.54	2.91
	C7	54	2.11	0.925	1.86	2.36
	C8	251	2.33	0.736	2.24	2.42
	C9	78	2.95	0.222	2.90	3.00
	Total	683	2.48	0.745	2.43	2.54
S3	C5	268	3.53	1.146	3.40	3.67
	C6	33	3.97	0.918	3.64	4.30
	C7	54	3.28	1.036	3.00	3.56
	C8	250	3.37	1.116	3.23	3.51
	C9	82	4.33	0.890	4.13	4.52
	Total	687	3.57	1.130	3.48	3.65
S4	C5	268	3.63	1.152	3.49	3.77
	C6	33	4.03	0.810	3.74	4.32
	C7	53	3.36	1.162	3.04	3.68
	C8	251	3.47	1.063	3.34	3.61
	C9	82	4.43	0.847	4.24	4.61
	Total	687	3.67	1.115	3.58	3.75

Table 6.14: Descriptive Statistics for S1–S4 by Cohorts (C5-C9)
[C5 (Non-CS/U), C6 (CS/U), C7 (CS/S), C8 (Non-CS/U), C9 (CS/U)]

presented in Table 6.14, the adaptive model-3 considerably contributes to increasing self-confidence. The one-way ANOVA, presented in Table 6.15, shows statistically significant differences between the five cohorts for the survey questions (S1–S4). Based on the surveys S1 and S2 responses, it demonstrates that the students firmly believe the quizzes significantly assisted in grasping fundamental programming principles as P values less than 0.05. The experiments showed that CS (C6 and C9) students have higher significant confidence increases after their quiz attempts. Based on survey data from S3 and S4, it is evident that adaptive assessments significantly increase their confidence in understanding errors better. According to the ANOVA effect size as shown in Table 6.15, it is more effective when provided as a revising

material to students with prior programming knowledge than the novices. Reflection on the feedback makes it easier to comprehend the basic concepts with self-efficacy. Higher eta-squared value (η^2=0.125) indicates the grouping variable had a notable effect on the variance in scores. The students claim that learning through formative assessment quizzes has improved their comprehension and self-confidence in learning programming. Although the effectiveness of adaptive formative assessment varies across cohorts—perhaps as a result of student demographics, instructor engagement, or delivery methods—it may have a favorable impact on students' confidence. Small but consistent gains in confidence suggest it is a viable teaching and learning tool for computer programming, particularly when tailored.

Survey question	**95% CI**		η^2	**Sig (P)**	**F**
	Lower	**Upper**			
S1	0.071	0.035	0.106	<0.001	13.050
S2	0.087	0.047	0.125	<0.001	16.089
S3	0.077	0.040	0.113	<0.001	14.275
S4	0.078	0.040	0.114	<0.001	14.275

Table 6.15: ANOVA effect for Survey Questions S1–S4 Across Cohorts

Chapter 3 described how I address this question. In formative assessment, Bloom's difficulty level-based adaptive model is employed to capture relationships between concepts with confidence. Novice students' opinions on their level of confidence were recorded in primary study phase-3 following their completion of the quizzes on each topic. In order to overcome this constraint, we expanded the primary study phase-4 and used extra survey questions to integrate the entire responses for comparison. We compared the self-confidence levels before and after practicing all of the assessment quizzes. Additionally, we gathered novice (C8) students' confidence in four subjects before and following all the quizzes: their ability to learn computer programming, create new programs, comprehend how programs operate, and recognize programming errors. The question was, 'How confident are you in your ability to do the following, using a number between 0 and 10?'.

The mean difference between their confidence levels before and after the quiz

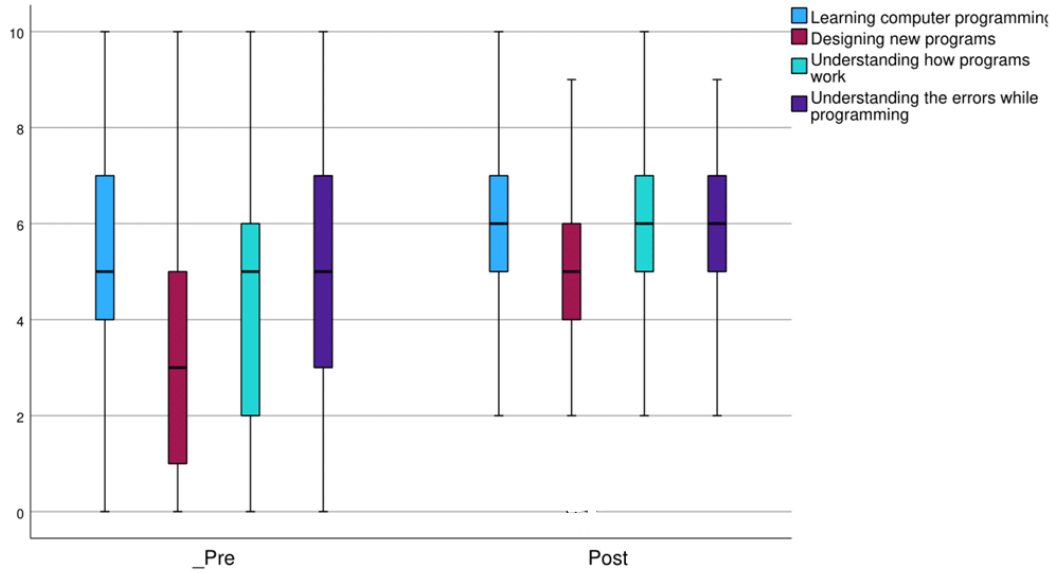


Figure 6.7: Mean difference on self-confidence

attempts is depicted in the figure 6.7. It demonstrates that their confidence levels have significantly increased. Before the quizzes, the confidence scores ranged from 0 to 10, and after the quizzes, they rose to between 2 and 10. Additionally, the mean value has grown. To determine the impact of adaptive formative assessment in these subjects, a paired-sample *t-test* was used. Table 6.16 indicates that their confidence levels before and after the quiz exercises differed statistically significantly. The post-quiz mean increased by 4.832 (95% CI: 0.34–0.88) for learning computer programming. Significant improvements were observed for designing new programs ($t = 5.879$, $\eta^2 = 0.125$), understanding program functionality ($t = 5.921$, $\eta^2 = 0.128$), and understanding programming errors ($t = 6.092$, $\eta^2 = 0.132$), all $p < 0.001$, two-tailed. These findings indicate that the adaptive formative assessment not only produced statistically significant gains but also meaningful effect sizes across these learning contexts.

6.6.2 Research Question-2

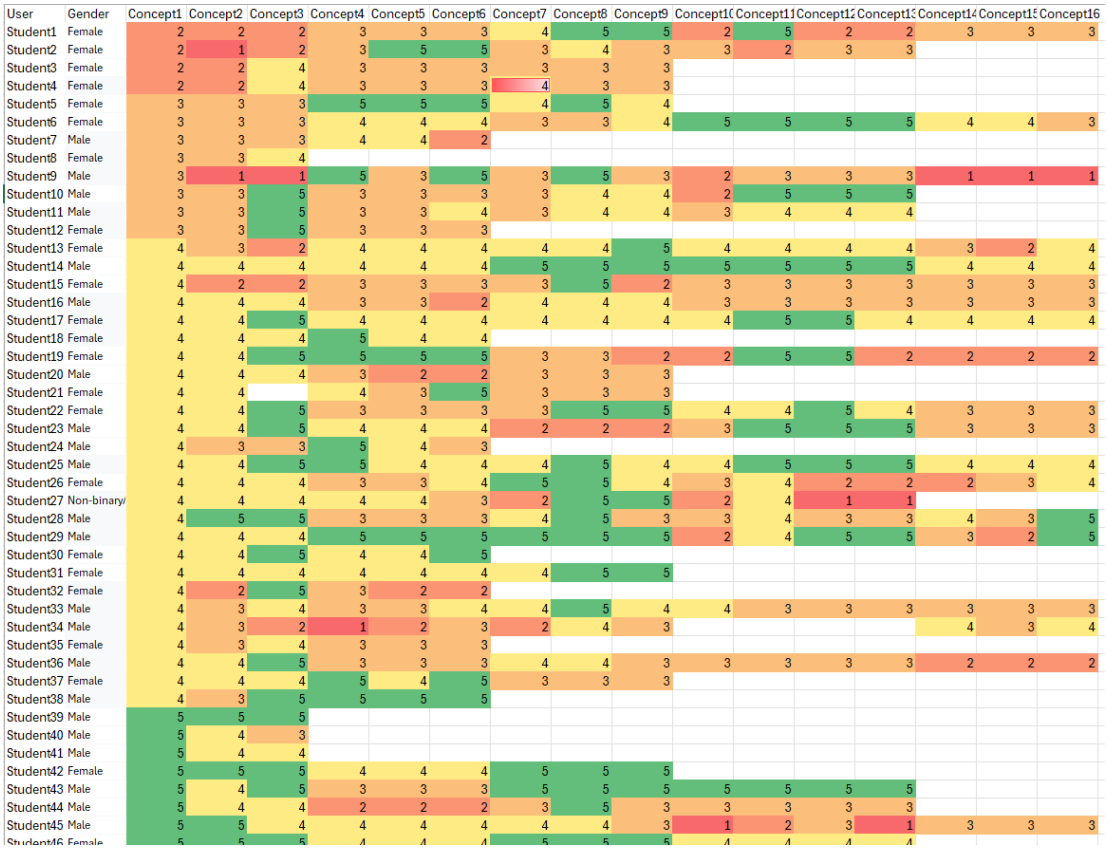
Upon conducting a literature review in order to answer the first question, I found that very few studies have used adaptive systems in formative assessment. Although

Confidence in	**Mean**		**SD**		**Diff**		**t**
	Pre	Post	Pre	Post	Mean	SD	
learning computer programming	4.63	6.23	2.212	1.593	1.597	2.602	4.832
designing new programs	2.74	4.77	1.872	2.012	2.032	2.722	5.879
understanding how programs operate	3.98	6.05	2.258	1.644	2.065	2.745	5.921
understanding programming errors	4.26	6.19	2.225	1.618	1.935	2.502	6.092

Table 6.16: Comparison of the novices' confidence before and after quiz attempts

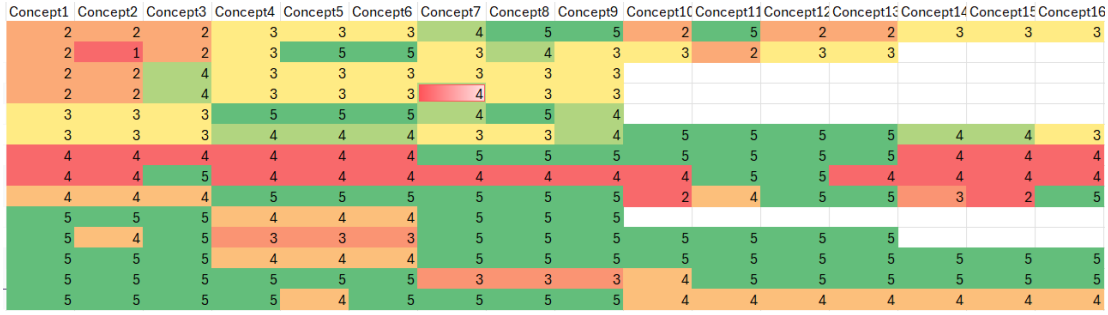
prior studies have employed adaptive frameworks, they have placed limited emphasis on strengthening novices' self-confidence in introductory programming concepts. This leads to the second research question which is, *How effective is adaptive formative assessment in facilitating novices' understanding of distinct conceptual components in programming?*. Following the final quiz and five weekly quizzes, C8 students' confidence patterns were combined to produce a heat map for the whole semester quizzes in order to address this question. As illustrated in Figures 6.8 and 6.9, two heat maps were produced: one for each student's confidence pattern and another for some evolving patterns derived from their feedback. It demonstrates that the confidence levels of low-confident students have considerably grown, while those of high-confident students are constant. As mentioned in section 6.5, certain concepts have more confidence rates than others, while advanced conceptions have lower rates. Additionally, we have compiled the percentage of correct answers for different questions of varying complexity between Non-CS (C5 & C8) and CS (C6 & C9) cohorts as shown in Table 6.17. The corresponding graph illustrated in Figure 6.10 compares how two cohorts performed across five quizzes, broken down by question difficulty (Easy, Moderate, Hard). Analysing these scores across CS and Non-CS cohorts helps evaluate whether this design is equitable, adaptive, and informative. Non-CS students (represented in dotted lines) are more balanced and consistent, especially in Moderate and Easy questions across most quizzes. Quiz 4 (Loop) emerged as a significant turning point, with the results indicating that Non-CS students outperformed their CS counterparts across all measured categories.

Up until Quiz-5, Non-CS students perform better than CS students on moderate questions. CS students (represented in continuous lines) consistently perform well on hard questions, particularly in Quiz-5. Quiz-5 might be easier for CS students, given the cluster of 100s.



User	Gender	Concept1	Concept2	Concept3	Concept4	Concept5	Concept6	Concept7	Concept8	Concept9	Concept10	Concept11	Concept12	Concept13	Concept14	Concept15	Concept16
Student1	Female	2	2	2	3	3	3	4	5	5	2	5	2	2	3	3	3
Student2	Female	2	1	2	3	5	5	3	4	3	3	2	3	3			
Student3	Female	2	2	4	3	3	3	3	3	3							
Student4	Female	2	2	4	3	3	3	4	3	3							
Student5	Female	3	3	3	5	5	5	4	5	4							
Student6	Female	3	3	3	4	4	4	3	3	4	5	5	5	5	4	4	3
Student7	Male	3	3	3	4	4	2										
Student8	Female	3	3	4													
Student9	Male	3	1	1	5	3	5	3	5	3	2	3	3	3	1	1	1
Student10	Male	3	3	5	3	3	3	3	4	4	2	5	5	5			
Student11	Male	3	3	5	3	3	4	3	4	4	3	4	4	4			
Student12	Female	3	3	5	3	3	3										
Student13	Female	4	3	2	4	4	4	4	4	5	4	4	4	4	3	2	4
Student14	Male	4	4	4	4	4	4	5	5	5	5	5	5	5	4	4	4
Student15	Female	4	2	2	3	3	3	3	5	2	3	3	3	3	3	3	3
Student16	Male	4	4	4	3	3	2	4	4	4	3	3	3	3	3	3	3
Student17	Female	4	4	5	4	4	4	4	4	4	4	5	5	4	4	4	4
Student18	Female	4	4	4	5	4	4										
Student19	Female	4	4	5	5	5	5	3	3	2	2	5	5	2	2	2	2
Student20	Male	4	4	4	3	2	2	3	3	3							
Student21	Female	4	4		4	3	5	3	3	3							
Student22	Female	4	4	5	3	3	3	3	5	5	4	4	5	4	3	3	3
Student23	Male	4	4	5	4	4	4	2	2	2	3	5	5	5	3	3	3
Student24	Male	4	3	3	5	4	3										
Student25	Male	4	4	5	5	4	4	4	5	4	4	5	5	5	4	4	4
Student26	Female	4	4	4	3	3	4	5	5	4	3	4	2	2	2	3	4
Student27	Non-binary	4	4	4	4	4	3	2	5	5	2	4	1	1			
Student28	Male	4	5	5	3	3	3	4	5	3	3	4	3	3	4	3	5
Student29	Male	4	4	4	5	5	5	5	5	5	2	4	5	5	3	2	5
Student30	Female	4	4	5	4	4	5										
Student31	Female	4	4	4	4	4	4	4	5	5							
Student32	Female	4	2	5	3	2	2										
Student33	Male	4	3	4	3	3	4	4	5	4	4	3	3	3	3	3	3
Student34	Male	4	3	2	1	2	3	2	4	3					4	3	4
Student35	Female	4	3	4	3	3	3										
Student36	Male	4	4	5	3	3	3	4	4	3	3	3	3	3	2	2	2
Student37	Female	4	4	4	5	4	5	3	3	3							
Student38	Male	4	3	5	5	5	5										
Student39	Male	5	5	5													
Student40	Male	5	4	3													
Student41	Male	5	4	4													
Student42	Female	5	5	5	4	4	4	5	5	5							
Student43	Male	5	4	5	3	3	3	5	5	5	5	5	5	5			
Student44	Male	5	4	4	2	2	2	3	5	3	3	3	3	3			
Student45	Male	5	5	4	4	4	4	4	4	3	1	2	3	1	3	3	3
Student46	Female	5	5	5	4	4	4	5	5	5	4	4	4	4			

Figure 6.8: All students' self-confidence responses on all concepts



Concept1	Concept2	Concept3	Concept4	Concept5	Concept6	Concept7	Concept8	Concept9	Concept10	Concept11	Concept12	Concept13	Concept14	Concept15	Concept16
2	2	2	3	3	3	4	5	5	2	5	2	2	3	3	3
2	1	2	3	5	5	3	4	3	3	2	3	3			
2	2	4	3	3	3	3	3	3							
2	2	4	3	3	3	4	3	3							
3	3	3	5	5	5	4	5	4							
3	3	3	4	4	4	3	3	4	5	5	5	5	4	4	3
4	4	4	4	4	4	5	5	5	5	5	5	5	4	4	4
4	4	5	4	4	4	4	4	4	4	5	5	4	4	4	4
4	4	4	5	5	5	5	5	5	2	4	5	5	3	2	5
5	5	5	4	4	4	5	5	5							
5	4	5	3	3	3	5	5	5	5	5	5	5			
5	5	5	4	4	4	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	3	3	3	4	5	5	5	5	5	5
5	5	5	5	4	5	5	5	5	4	4	4	4	4	4	4

Figure 6.9: Self-confidence evolving pattern over concepts

Additionally, a box plot illustrating the distribution and variation of scores for Non-CS and CS's moderate, easy, and hard questions has been created in Figure

Difficulty	Moderate		Easy		Hard	
Quiz	CS	Non-CS	CS	Non-CS	CS	Non-CS
1	88.89	77.28	58.33	50.00	87.94	84.82
2	90.64	86.00	72.22	78.59	89.68	86.79
3	70.42	88.51	83.33	59.56	71.37	66.99
4	60.50	90.62	50.00	79.17	82.42	89.20
5	90.48	78.68	100.00	69.20	97.14	86.63

Table 6.17: Correct responses percentage by Cohort (CS vs Non-CS) for Each Quiz and Difficulty Level

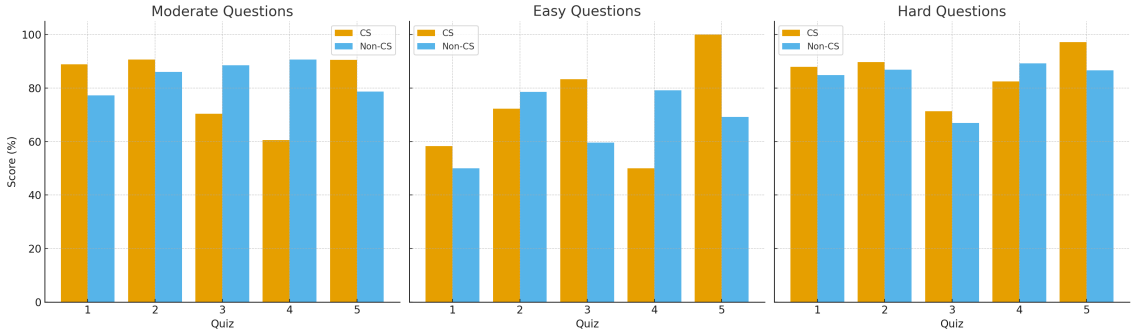


Figure 6.10: Correct responses percentage by Difficulty levels across quizzes

6.11. It shows that both cohorts' correct responses on moderate questions are comparatively high and stable, suggesting steady performance. Easy questions, on the other hand, show more variation, especially within Non-CS, which could be due to overconfidence, inattention, or insufficient fundamental knowledge. It is interesting to note that CS outperforms Non-CS and exhibits less variability when asked difficult questions, suggesting that CS may be better prepared or supported more successfully when dealing with hard questions. The significance of varied scaffolding in filling up foundational gaps and pushing advanced learners is highlighted by these patterns.

6.6.3 Research Question-3

The next research question is, *To what extent can adaptive formative assessment encourage novice learners to improve their programming skills?*. To address this research question, we then chose to examine if there were significant variations in scores between the two groups of students who attended and those who did not,

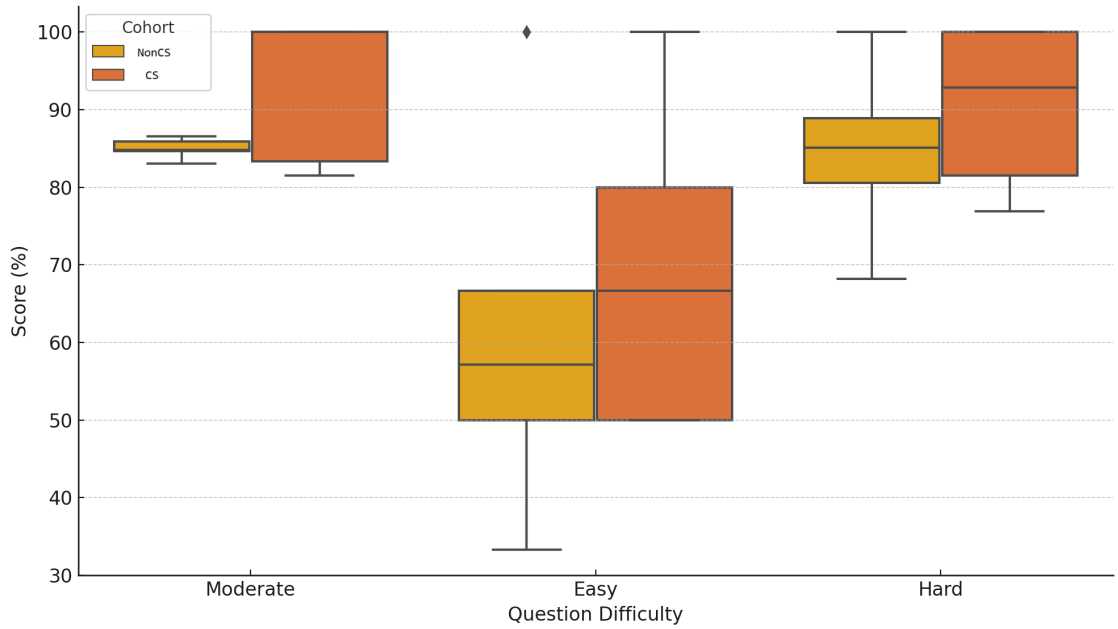


Figure 6.11: Correct responses distribution by Difficulty and Cohorts

Aspect	**Participant**	**N**	**Mean**	**SD**	**t**
Score	Yes	84	6.33	1.500	0.865
	No	37	6.05	1.914	
Time Taken	Yes	84	15.4014	6.79762	1.09
	No	37	16.6420	5.26143	

Table 6.18: Mean difference between quiz participants and non-participants

as well as how gender affected these differences. Therefore, this study offered an optional revision quiz to Cohort-C8 before the exam in order to determine the effect of the adaptive formative assessment quizzes on exam scores. Prior to the exam, 84 out of 121 students took the revision quiz. We examined the differences in exam scores between students who took the quiz and those who did not. Figure 6.12 displays exam score differences by gender and quiz participation. To determine the difference in scores between participants and non-participants, an independent t test was used. Table 6.18 shows the mean difference of the exam score and completion time between participants and non-participants. These findings show that the adaptive quiz significantly influences both scores and completion times between quiz participants and non-participants. This also reveals that female students benefited more than male students, as Figure 6.12 illustrates.

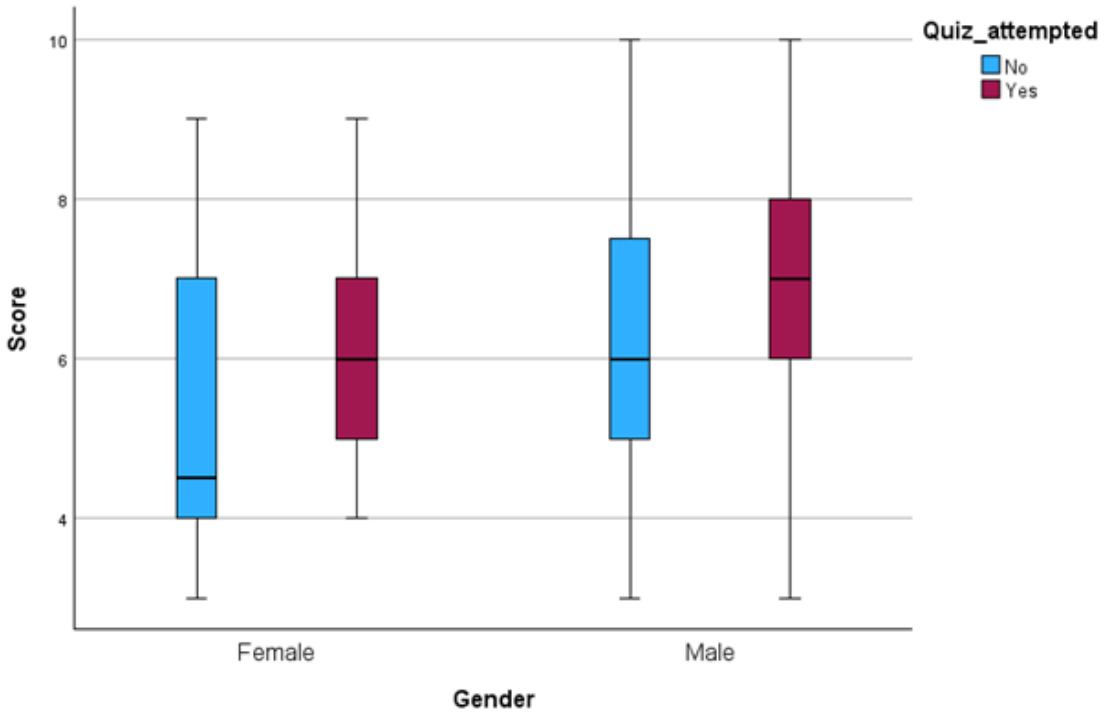


Figure 6.12: Difference in exam scores based on quiz participation and gender

6.7 Conclusion

This chapter presented a discussion of the study using adaptive formative assessment. This study was driven to explore how an adaptive formative assessment might increase novices' confidence in their ability to learn programming. We employed two different kinds of explicit intervention metrics to address the research questions, such as students' self-rated confidence and their actual exam scores following the quizzes. By exploring novices' self-rated confidence and evaluating whether adaptive formative assessment would help them, this study found that there was a statistically significant difference between the pre and post self-confident values. According to the findings, the majority of students entered with high confidence, while just a small percentage entered with low confidence. This implies that the majority of students think that learning from mistakes through adaptive formative assessment aids in their comprehension of fundamental concepts of programming. Additionally, it shows how adaptive assessments improved their comprehension of errors. Overall, the self-rated confidence, exam score, and completion time have a significant positive correlation. As a conclusion, adaptive

formative assessment can be employed to teach novices introductory programming concepts through increasing their self-confidence and helping them learn from their mistakes. As they can effectively aid in the learning of programming, one approach we proposed in this contribution is to develop formative quizzes with adaptive techniques and difficulty levels.

Chapter 7

Conclusion and Future work

Presented in this thesis are, Chapter 1 - Introduction to several assessment techniques, including the background of formative assessments in programming education and the aim and research objective for this thesis; Chapter 2 - A systematic literature review of formative assessment for computer programming; Chapter 3 - Development of formative assessment framework; Chapter 4 - Research methodology; Chapter 5 - Preliminary experiments of formative assessment for programming; and Chapter 6 - Main experiment using adaptive formative assessment framework and its results. In Chapter 5 and 6, detailed findings from the student survey are presented.

7.1 Conclusion and Key findings

This study investigated how adaptive formative assessment can enhance novice programmers' self-confidence in learning programming by using common programming errors as learning resources. Adaptive formative quizzes, which adjust difficulty based on prior responses, were found to enhance understanding of programming concepts, foster confidence, and promote learning from mistakes. Two interventions were employed—tracking students' self-rated confidence and measuring exam scores after adaptive quizzes. The findings showed that self-rated confidence, exam scores, and task completion time were positively correlated. There is a

statistically significant improvement in self-confidence from pre- to post-assessment. Findings indicated that adaptive assessment supports comprehension of fundamental programming concepts, particularly by helping students understand and correct common errors. The adaptive quiz design, which adjusts difficulty based on prior responses, allowed proficient learners to progress quickly while providing targeted support to others. This approach expands learning opportunities, fosters confidence, and offers a viable teaching tool for introductory programming. Starting with the three research questions that guided this study, this section discusses the findings from Chapter 5 and 6, and summarises the main conclusions from the findings.

- **Key finding-1: Enhancing Student Confidence Through Adaptive Formative Assessment**

  In Chapter 1 our hypothesis was stated as 'Using an adaptive approach to formative assessment can increase novices' self-confidence in terms of comprehending fundamental concepts and errors in introductory programming'. According to the findings shown in sections 5.5, 6.3, and 6.5, adaptive formative assessment as supplementary resources significantly increased the self-confidence of novice students learning programming. As a conclusion, we argue that these supportive resources increase their chances of success when learning programming. These formative assessment quizzes motivate students to evaluate and learn from their mistakes, which in turn encourages them to learn computer programming. As a result, learning opportunities have expanded through increasing students' confidence, and understanding the frequent errors. It can be a viable teaching and learning tool for computer programming. Reflection on the feedback makes it easier to comprehend the basic concepts with self efficacy. The students claim that learning through formative assessment quizzes has improved their comprehension and self-confidence in learning programming.

- **Key finding-2: Confidence Gains Across Cohorts**

  Students from three distinct cohorts participated in the experiments: CS, Non-CS, and Secondary school. All cohorts showed positive results from the models. Findings show that all these models increase novices' self-confidence. In particular, adaptive model-3 considerably contributes to increasing self-confidence as discussed in 5.5. The findings show that CS students have higher confidence increases than other cohorts according to the results for S1 and S2 in Table 6.14. It can be as a result of their familiarity with computing-related subjects and the faculty's skilled delivery of instruction. These experiments not only showed an increase in self-confidence but also encouraged us to see how well these tools functioned as a teaching tool. Giving feedback has a substantial impact on formative assessment during learning, and it helps students understand the material, especially for revision and bridging to advanced levels.

- **Key finding-3: Learning from Errors Through Adaptive Feedback**

  The enhanced error messages were used as feedback in the quizzes at various points. The aim was to familiarize the role and source of errors at various programming levels to the novice students in a step-by-step manner. The findings revealed that adding most common errors to adaptive formative assessment greatly benefits the learning process of programming. It demonstrated that this model considerably contributes in improving their comprehension of frequent errors in programming. According to Table 6.14's statistical findings, CS students are better than other cohorts at understanding and correcting errors, as indicated by the higher mean values of C6 and C9 for S3 and S4. This indicates that knowledge of CS facilitates a better understanding of the errors.

- **Key finding-4: Tailored Feedback for Mastery of Programming Concepts**

  Through adaptive formative assessment, we investigated Bloom's taxonomy difficulty levels for navigating quizzes in the conducted experiments. We observed that the formative assessment approach enhances students' understanding of independent parts of programming by providing them with additional explanations as feedback when they provide an incorrect answer. The findings shown in Figure 6.10 and 6.11 imply that understanding of independent components can be efficiently improved by adaptive formative assessment based on Bloom's taxonomy difficulty levels. We also discovered that secondary level students require different explanations than CS and Non-CS students as the outcome of qualitative analysis in section 6.3 and 6.5. Additionally, each question needs to be summarised so that students who have failed at all levels can understand it.

- **Key finding-5: Confidence and Challenge**

  Table 6.13 illustrates how question difficulty rates compare to students' self-confidence. The findings demonstrate that there is a positive relationship between students' self-confidence and difficulty rates. Thus, when students handle harder questions, they are more confident. CS students, in particular, feel more confidence than Non-CS when they tackle harder questions. Students that are more confident are more ready to attempt challenging tasks like problem solving, per the qualitative analysis in section 6.3.

- **Key finding-6: Participation Reflects Quiz Effectiveness**

  More than 80% of quiz participants responded to the survey in the main study, demonstrating strong student interest and willingness to engage with feedback after the quiz. They appear eager to practice these quizzes in order to gain knowledge even if they are provided as an optional resource with voluntary survey participation. It demonstrates the effectiveness of the quizzes

in reaching the research objectives.

- **Key finding-7: Supporting Learning and Revision**

  Our research indicates that non-adaptive teaching methods, such as employing Vevox and Kahoot, are effective in keeping school children engaged. The most effective way to understand a mandatory topic is through Adaptive model-1. Adaptive model-2 is a good way to help students become more comfortable with progressive concepts, such as introducing syntax, writing brief programs, and eventually using numerous programming concepts to solve complex issues. As discussed in section 5.5, Adaptive model-3 is closely aligned with course curricula. Also, our findings show that this model significantly increased students' self-confidence in different aspects. The adaptive formative assessment quizzes can be supported as a learning, revision and bridging instrument. As discussed in section 6.6, the adaptive formative assessment quizzes can be used as a bridge, learning, and revising tool for both CS and Non-CS students. It increases exam scores and decreases test completion times when used as revision material.

Additionally, the system features an easy-to-use interface that should make assessment easier for inexperienced users. We include them into Google Forms, enabling a standalone tool where a user can provide resources in any format, at any time, and without the need for an evaluation system on the LMS. As a conclusion, we argue that these supportive resources increase their chances of success when learning programming. As a result, learning opportunities have expanded through increasing students' confidence, and understanding the common errors. It can be a viable teaching and learning tool for computer programming.

7.2 Contributions

This dissertation's primary contribution is the framework for adaptive formative assessment. After developing and analysing three different adaptive approaches, we

suggested adaptive model-3 for learning programming. By providing different sets of questions using difficulty levels for students with varying skill levels, this approach focused on filling the research gap on motivating novices to learn. This framework significantly increased novice students' self-confidence in learning programming as discussed in previous chapters. Our framework offers some additional features, which are listed below.

7.2.1 Student-centred

A student-centred assessment system is an assessment approach that focuses on the learner's needs, goals, progress, and active involvement—rather than being driven solely by the instructor's requirements or a fixed testing schedule (Glavind et al., 2023). The results show that the students engaged fully in the assessments regardless of time, location, or skill level. The following characteristics of a student-centered approach are met by our framework (Rich et al., 2014):

- Coverage of course content: Ensuring that students have read the entire course is essential for effective teaching. Quiz strategies make sure the students have read the entire content covered in it by answering every question.

- Foundation needs met: Given that it is in line with the lecture schedule, the content of the quizzes is relevant to the requirements and interests of the students. The week's topics could be assessed in-class, and students are made aware of this. Providing the quizzes on the lecture materials could be helpful in reviewing the topic for each week's lecture, since many learners tend to skim the chapter rather than carefully reading it.

- Anytime, anywhere: Students can take them online or on campus whenever it is convenient for them. As students finish quiz questions, this framework provides them with instant text feedback, regardless of whether the answer is correct or incorrect.

- Authentic: This framework serves to authenticate the knowledge and experiences of learners. Students can form better study habits by receiving assistance and having their learning styles accommodated.

- Student ownership: Throughout the entire learning process, teachers provide their students the freedom to be independent. Students ought to understand what and how they are learning. Adaptive formative assessment is one way to accomplish this.

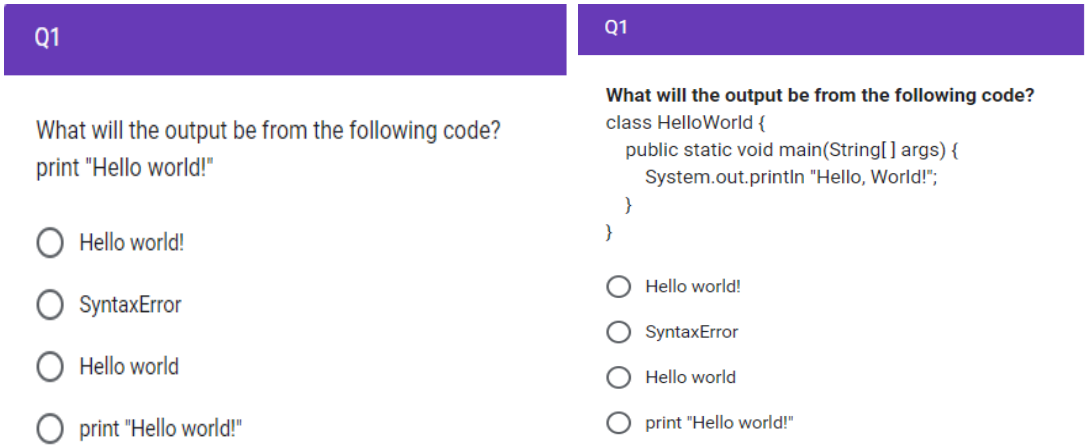
- Modular design of the Framework: The framework facilitates the straightforward development and adaptation of quizzes across multiple programming languages, enhancing its versatility for diverse instructional contexts.

7.2.2 Programming language independent

The objective of all introductory programming courses is to introduce the fundamental concepts of programming using various programming languages. In our framework, we designed a series of quizzes covering fundamental programming concepts in Python. However, our framework is suitable for all other programming languages covering similar concepts. For example, Figure 7.1a displays in a Python question. By mapping the matching syntax, it can be converted to Java language as shown in Figure 7.1b.

7.2.3 Suitable to other subjects

Other computer-related areas can also benefit from the framework. Our system works well with other computer courses at introductory level such as Introductory databases. However, the questions must be categorised according to their varying degrees of difficulty using a model such as Bloom's taxonomy to use Adaptive model-3. Figure 7.2 displays an example question in the database course. With an additional Cohort at DCU, we administered a quiz on fundamental database



(a) Python (b) Java

Figure 7.1: A sample question (Python & Java)

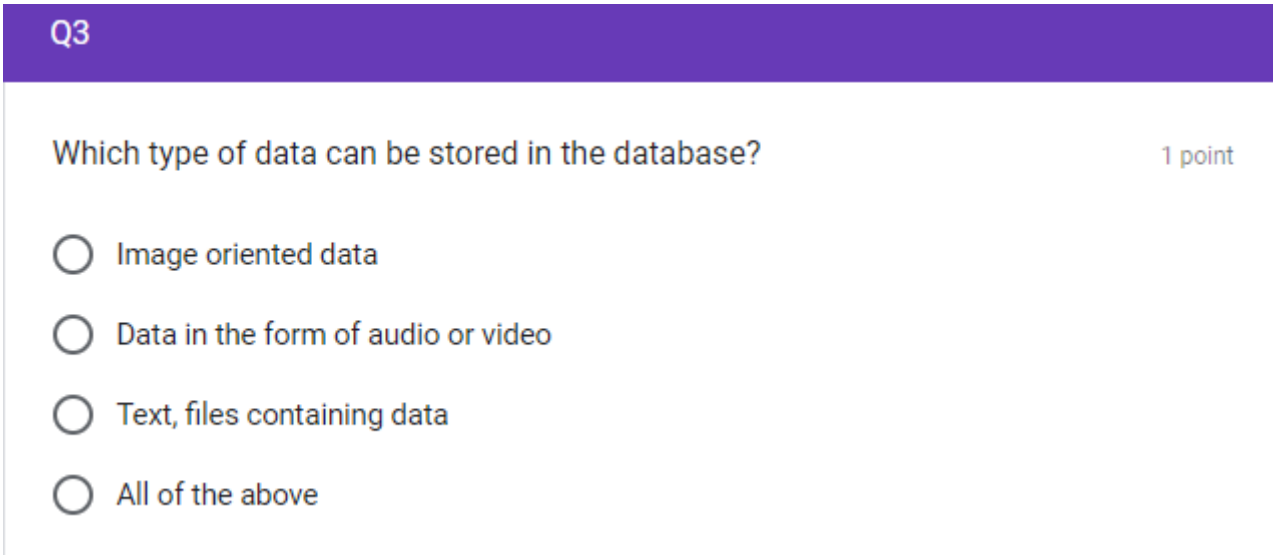


Figure 7.2: A sample question in databases

concepts. There were seven students that took part and we received positive feedback from them. They requested to use this approach for concept revision.

7.2.4 Flexibility and adaptability

Since the proposed framework aligns with the core fundamentals of introductory programming, it is inherently adaptable across multiple educational contexts. Prior research has shown that early programming concepts and common error patterns are largely consistent across secondary and tertiary education, making adaptive instructional frameworks transferable across levels (Ihantola et al., 2015). In particular, it is well suited for introductory programming courses as well as

secondary-level computer science curricula, such as Leaving Certificate computer science, where learners exhibit diverse prior knowledge and learning needs.

A key strength of the framework lies in its adaptive feedback layer, which can be hybridised with AI-based components. Rather than relying exclusively on fully automated generation, the framework supports a hybrid approach in which generative AI models augment structured, pedagogically grounded feedback rules. This design aligns with recent work advocating for human–AI partnership models in programming education, where AI-generated feedback adapts to learner behavior while preserving instructional intent and reliability (Lohr, Keuning, & Kiesler, 2024; Scholz et al., 2025). Through this hybridisation, feedback can be dynamically tailored based on learner interaction data, error patterns, and progression, supporting not only cognitive outcomes but also motivational and self-efficacy-related dimensions of learning (Hattie & Timperley, 2007). For example, AI-generated feedback can adapt tone, specificity, and scaffolding level depending on learner confidence and performance history, while rule-based or instructor-authored feedback ensures pedagogical consistency and reliability. This capability is particularly relevant for addressing challenges associated with novice programmers, such as overconfidence or disengagement, which have been widely documented in the literature (Loksa et al., 2016).

The framework's adaptability also extends to delivery modalities. It can be deployed in traditional classroom settings, synchronous online environments, or asynchronous, on-demand learning scenarios without modification to its underlying logic. This flexibility is consistent with evidence that adaptive feedback mechanisms can be effective regardless of delivery mode when grounded in learner interaction data rather than instructional context (Aleven et al., 2016).

From a systems perspective, the framework supports interoperability through link-based access, enabling seamless integration with learning management systems (LMS), institutional platforms, or standalone applications. Such loosely coupled architectures are recommended for scalable educational AI systems, as they facilitate

incremental adoption and reduce dependency on specific platforms or interfaces (Holmes et al., 2019). This design allows AI-enhanced feedback services to be embedded into existing educational infrastructures without requiring invasive changes. As a result, educators can incrementally adopt AI capabilities—such as automated feedback generation or adaptive difficulty adjustment—while maintaining control over instructional goals and assessment standards.

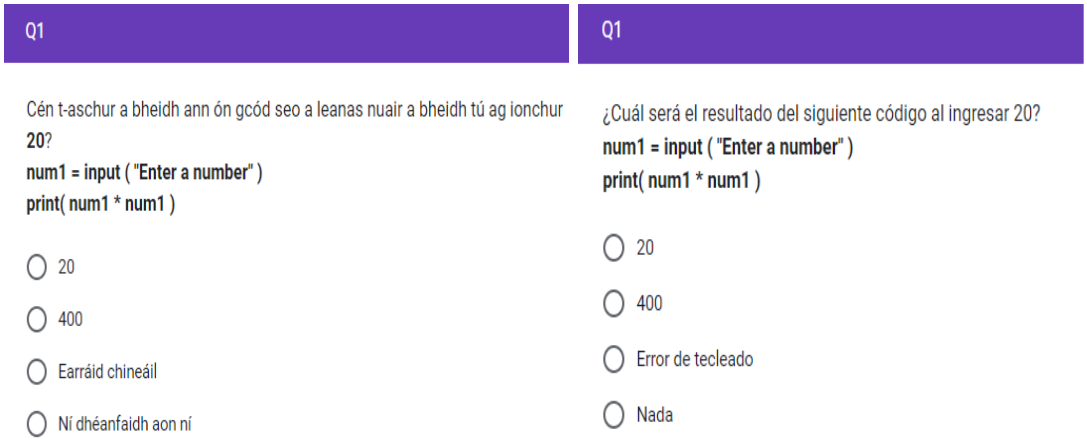
7.2.5 Scalability

Scalability refers to a system or process's ability to manage increasing workloads or its potential for expansion to meet the need (Balderas et al., 2018). Our system was developed using Google Forms, which can help users grow because we can provide it as a stand-alone tool by giving them the link. Additionally, it features a reusable content bank that enables the creation of questions from a pool of labeled questions with corresponding answers and feedback.

Overall, the hybridisation of structured pedagogical design with AI-driven feedback mechanisms enhances both the adaptability and scalability of the framework. By supporting multiple educational levels, delivery modalities, and technological ecosystems, the framework addresses current calls for responsible, scalable, and learner-centered AI integration in programming education (Holmes et al., 2019; Zawacki-Richter et al., 2019).

7.2.6 Interface language independent

Our framework can be used in several languages, similar to program language independence. With translation support (Google translate), it is straightforward to pose assessments in various languages. We developed Irish and Spanish language interfaces to the quizzes. Figures 7.3a and 7.3b display sample questions in both Irish and Spanish. After taking the quizzes, users of the corresponding language expressed appreciation.

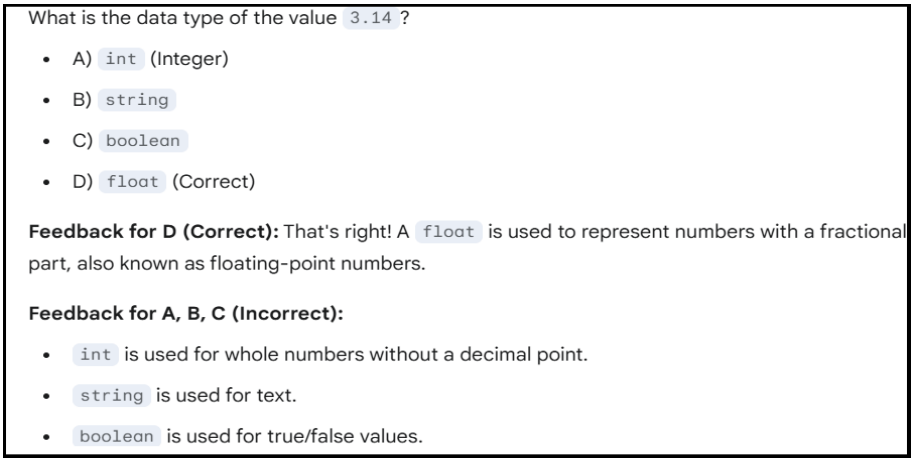


(a) Irish (b) Spanish

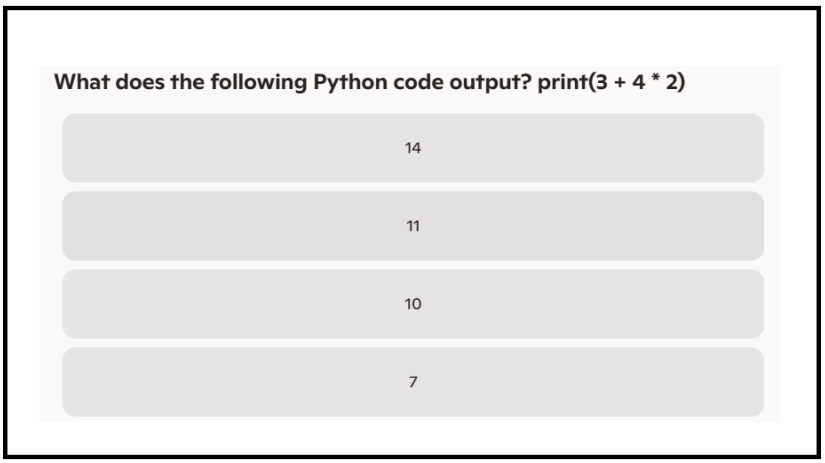
Figure 7.3: A sample question (Irish and Spanish)

7.2.7 Quality of Assessment & Feedback: Better Than AI

Recently, AI-based assessment techniques are being employed increasingly in higher education because they offer benefits like greater flexibility, swift feedback, and personalised learning experiences (Luo et al., 2025; Moorhouse et al., 2023). Examining the quality of formative assessments is pertinent given the increasing role of AI in programming education and its effects on the educators creating the assessments (Azaiz et al., 2024; Matthews, 2025). The features of formative assessment tools and AI tool capability must be compared in order to identify potential impacts on offering recommendations for content or organization (Yu & Xie, 2025). Some large language models (LLMs) that have been quite popular among students for answering questions from many educational fields are ChatGPT, Gemini, Copilot and Meta AI (Das Roy et al., 2025). DCU has approved the use of a few enterprise AI tools, such as Copilot and Gemini, for educational purposes. Therefore, we made the decision to compare our formative assessment questions and responses with these tool-generated. With these AI tools, we prompted them to generate formative assessment questions and feedback. AI generated questions are shown in Figure 7.4a and 7.4b. We evaluated the features of these AI tools with regard to formative assessment quizzes (Das Roy et al., 2025; Luo et al., 2025). As shown in Table 7.1, our framework generates adaptive formative quizzes according to



(a) Gemini



(b) Copilot

Figure 7.4: AI generated questions

the level of difficulty. This comparison shows that, with the exception of dynamic question generation, our framework performs better in the majority of contexts. However, it could be improved by using AI tools. Further formal investigation is required to better assess the advantages and drawbacks of these AI tools, this will be discussed in Future Work.

7.3 Limitations

Every research project has its limitations, and these limits arise from a series of challenges. Here is a summary of the relevant challenges in this research context:

- Here, a key factor in adaptive formative assessment is programming complexity. However, the measure of programming complexity was tied to

Task	Our framework	Gemini	Copilot
Generating formative questions	Pre-defined Questions for different topics with complexity.	Well-presented questions with different topics and complexity.	Questions without relevant information (Additional involvement is required to achieve the desired result.)
Presentation of the response	Well-presented responses with highlighted titles.	Responses were structured with important points highlighted.	Most responses were quite brief and in the form of buttons.
Accuracy	Responses were accurate.	Although most responses were accurate, some lacked complete information.	Accuracy was observed but some answers lacked important details.
Length	Elaborative answers, with excessive information.	Responses were moderate in length.	Very short, point-wise answers.
Adaptivity	Difficulty-based.	Generates an adaptable quiz that utilizes additional inputs.	produces several forms of adaptability, such as mastery, topic-driven, or difficulty-based.
Lucidity	Lucid.	Lucid.	Short sentences that included only key points.
Relevance to the question asked	Relevant.	Relevant.	Relevant. (Not most of the time)

Table 7.1: Comparison with AI tools

the adaptive system's classification of questions as easy, moderate, or hard. While this provides a practical framework, it does not fully capture the nuanced cognitive demands of different programming concepts (e.g., syntax vs. algorithmic reasoning, or loops vs. recursion). As a result, some tasks classified at the same difficulty level may not have been experienced equally across students.

- This study relied on self-reported confidence ratings. Although these measures provide valuable insights into student perceptions, they may not always align

with demonstrated competence. Students may overestimate or underestimate their understanding, which introduces the risk of bias in interpreting adaptive assessment outcomes.

- To address concerns regarding leading questions and inflated self-efficacy, survey items were carefully phrased to focus on learners' interactions with the feedback rather than perceived programming ability. Additionally, self-reported perceptions were triangulated with objective performance data, enabling the identification of discrepancies consistent with the Dunning–Kruger effect. Effect size analysis and the presence of non-significant survey dimensions further suggest that results are not solely driven by overconfidence. Nevertheless, self-report bias remains a limitation, which future work will address through validated self-efficacy instruments and longitudinal evaluation.

- Cohort differences (e.g., CS vs. non-CS students) and prior exposure to programming were not fully controlled. This may explain observed variability in confidence, particularly on advanced topics, where background knowledge could strongly influence both performance and self-assessment.

- In order to examine this assessment approach's complete learning impact, we were unable to incorporate it into the lesson plans of other faculty members. As supplementary materials, we provided these assessments, and we examined the results. Working more closely with the lecturer/teacher to incorporate the quizzes at appropriate times could enhance use of this approach.

- As the questions in adaptive formative assessment are designed using a well-designed taxonomy, setting them up requires extra planning.

- It cannot be combined with/used in conjunction with programming editors because it employs stand-alone quizzes. Consequently, evaluations of programming code submissions are not supported.

- Feedback for multiple-choice questions has to be designed with care and consideration in terms of identifying whether the answers are entirely accurate or which parts are incorrect and this can be time-consuming.

- Although our framework maintains its effectiveness and efficiency as the number of students, courses, or assessment tasks increases — without a proportional rise in resources — it is important that questions are designed to address the needs of all students. Since the focus was on novice learners, advanced students may find the tasks comparatively easier.

- Quiz-based assessment in programming education is inherently sensitive to question design, as relatively small variations in wording, structure, or required cognitive processes can influence learner performance independently of actual conceptual understanding (Ihantola et al., 2015). To mitigate this sensitivity, quiz questions were designed to align closely with clearly defined learning objectives, with each question targeting a single programming concept. This approach reduces ambiguity and avoids conflating multiple skills, such as syntax, logic, and program tracing, within a single item (Biggs, 2003).

7.4 Future Work: Possible directions for Adaptive formative assessment

Adaptive formative assessment is one of the approaches to motivate novices for effective programming learning that aims to increase their understanding and learning by providing feedback on programming submissions. Enhanced personalization can be achieved through adaptive assessments, which arrange instructional resources (Vie et al., 2017). This section outlines future research directions and development plans. The following areas are identified as key priorities for further exploration and development.

7.4.1 Formative Assessment using AI

Feedback in the form of enhanced error messages in programming was effective when used in adaptive formative assessment. However, creating enhanced error messages from compiler messages is challenging and it requires more work to generate the enhanced error messages. Recent advancements in AI have made it possible to generate such messages (Koutcheme & Hellas, 2024; Koutcheme et al., 2025). Hence creating enhanced error messages using extensive AI language models is a further possibility when using adaptive formative assessments (Nguyen & Allan, 2024). Whereas metacognitive scaffold components (planning, monitoring, and reflection) are either implicit or absent, the majority of LLM feedback concentrates on explaining errors, suggesting fixes or giving. Despite these developments, the majority of LLM-based systems concentrate on single-submission feedback and do not modify feedback techniques in response to longitudinal learner behavior, such as persistent misconceptions or recurrent errors (Lohr, Keuning, & Kiesler, 2024; Pădurean et al., 2025; Silva & Costa, 2025). Generating questions according to difficulty levels is also a major potential that AI provides for adaptive formative assessment in programming. The majority of researchers focus on leveraging AI to create programming questions. LLM feedback has no influence on task difficulty, question sequence, or assessment pathways (Scholz et al., 2025). Although LLMs can generate a variety of feedback categories, the system does not determine which type to utilize when (Lohr, Keuning, & Kiesler, 2024). However, no work has been done to categorize the questions based on their degree of difficulty. As a result, this study recommends using AI to categorize the questions based on their degree of difficulty. In addition, there has been some research to date on this approach, and adaptive assessment has not yet been implemented (Cucuiat & Waite, 2024).

7.4.2 Formative feedback using SMT solvers

As discussed in chapter 2, SMT solvers (e.g., Z3) can encode the intended solution and the student's code as logical constraints and check whether they are semantically

equivalent (Farias et al., 2024). If the student's code fails equivalence, the solver can identify specific conditions (input scenarios) under which the code fails. Future work could integrate SMT solvers into adaptive formative assessment to deliver highly targeted, interactive feedback. Using SMT solvers, systems could generate runnable, personalised code examples for students to experiment with, provide tiered hints that progressively guide learners from conceptual cues to full solutions, and implement error-driven learning by dynamically creating micro-exercises that address the exact misconceptions detected in a student's incorrect answer. This approach would enhance adaptivity, promote deeper conceptual understanding, and accelerate mastery in introductory programming.

7.4.3 Categorise the difficulty levels

A key factor in adaptive formative assessment is programming complexity. Two well-known studies provided various classifications for the complexity: Bloom's taxonomy (Thompson et al., 2008) and Block model (Schulte, 2008). However, the complexity may vary among students or across programming languages because students' opinions on what they have learned are demonstrably different from what they actually learn (McCall & Kölling, 2014). Thus, this study suggests doing a methodical investigation to categorize programming complexity that can be applied to adaptive formative assessment and explore the impact of tighter integration with a programming module. IRT b-parameter could be used to investigate the complexity of programming concepts (Alves et al., 2021; Mohamad & Omar, 2017). Additionally, the difficulty and order of programming problems can have an impact on students' ability to learn programming, according to a recent study (J. Wang et al., 2023).

7.4.4 Enhancing adaptivity in Formative Assessment

Dynamic Difficulty Adjustment is a technique—often used in games and adaptive learning systems—where the difficulty level of tasks is automatically modified in

real time based on the learner's performance (Zheng, 2024; Zohaib, 2018). As an alternative to simply labeling each question as easy, moderate, or difficult, Future research can investigate the efficacy of multi-level question banks that dynamically adjust the difficulty of the questions based on learners' performance in real time. It can change the order of questions according to areas of weakness rather than just the difficulty of a particular topic.

Although AI-driven adaptive learning systems can modify learning tactics and content in response to learners' interactions and performance, this flexibility is frequently restricted to content sequencing or feedback delivery rather than dynamic task difficulty regulation customized to learners' progress in programming environments. AI customization generally concentrates on customizing teaching based on learner profiles and performance indicators, but it does not consistently extend to dynamic difficulty level adjustment across tasks, according to research on adaptive learning (Kolluru et al., 2018).

7.5 Closing statement

The experience presented in this thesis described the use of adaptive formative assessment in introductory programming to help novice students overcome the challenges of learning to program, particularly in increasing self-confidence in identifying and understanding more frequent errors. The results reinforce that this approach helped novice students become more confident in programming and dispel some of the misconceptions surrounding it. As a conclusion, we argue that adaptive formative assessment quizzes motivate students to evaluate and learn from their mistakes, which in turn encourages them to learn computer programming. It can be a viable teaching and learning tool for computer programming. These quizzes make it easier to comprehend the basic concepts. The students claim that learning through formative assessment quizzes has improved their comprehension and increased their confidence in learning programming. They also claim to be satisfied and indicate that they would repeat this teaching technique again, as evidenced by their high level

of engagement. This study recommends the adaptive formative assessment quizzes could help novice students learn more effectively in other programming languages as well as in other languages like Irish, Portuguese and Spanish.

Bibliography

Abreu, P. H., Silva, D. C., & Gomes, A. (2018). Multiple-choice questions in programming courses: Can we use them and are students motivated by them? *ACM Transactions on Computing Education (TOCE)*, *19*(1), 1–16.

Acosta-Gonzaga, E., & Ramirez-Arellano, A. (2022). Scaffolding Matters? Investigating Its Role in Motivation, Engagement and Learning Achievements in Higher Education. *Sustainability*, *14*(20). https://doi.org/10.3390/su142013419

Adamopoulos, F. A. (2019). Learning Programming, Student Motivation. In A. Tatnall (Ed.), *Encyclopedia of education and information technologies* (pp. 1–10). Springer International Publishing. https://doi.org/10.1007/978-3-319-60013-0_182-1

Ahadi, A., Lister, R., Lal, S., & Hellas, A. (2018). Learning Programming, Syntax Errors and Institution-Specific Factors. *Proceedings of the 20th Australasian Computing Education Conference*, 90–96. https://doi.org/10.1145/3160489.3160490

Aleven, V., Roll, I., Mclaren, B., & Koedinger, K. (2016). Help Helps, But Only So Much: Research on Help Seeking with Intelligent Tutoring Systems. *International Journal of Artificial Intelligence in Education*, *26*, 1–19. https://doi.org/10.1007/s40593-015-0089-1

Aljowaed, M., & Alebaikan, R. A. (2018). Training needs for computer teachers to use and teach computational thinking skills. *International Journal for Research in Education*, *42*(3), 237–284.

Alves, N. D. C., Wangenheim, C. V., Hauck, J. R., & Borgatto, A. F. (2021). An Item Response Theory Analysis of Algorithms and Programming Concepts in App Inventor Projects. *Anais do Simpósio Brasileiro de Educação em Computação*, 01–11. https://doi.org/10.5753/educomp.2021.14466

Alzahrani, N., & Vahid, F. (2021). Common Logic Errors for Programming Learners: A Three-decade Literature Survey [https://peer.asee.org/36814]. *2021 ASEE Virtual Annual Conference Content Access*. https://doi.org/10.18260/1-2--36814

Andrade, H., & Heritage, M. (2017, July). *Using Formative Assessment to Enhance Learning, Achievement, and Academic Self-Regulation*. https://doi.org/10.4324/9781315623856

Anfurrutia, F. I., Álvarez, A., Larrañaga, M., & López-Gil, J.-M. (2018). Integrating Formative Feedback in Introductory Programming Modules. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, *13*(1), 3–10. https://doi.org/10.1109/RITA.2018.2801898

Asamoah, D. (2019). Traditional assessment procedures, and performance and portfolio assessment procedures: An in-depth comparison. *1*, 28–30.

Athinaiou, M. (2023, November). *Teaching programming: A critical perspective on pedagogy*. https://doi.org/10.13140/RG.2.2.19426.91842

Azaiz, I., Deckarm, O., & Strickroth, S. (2023). AI-Enhanced Auto-Correction of Programming Exercises: How Effective is GPT-3.5? *International Journal of Engineering Pedagogy (iJEP)*, *13*(8), 67–83. https://doi.org/10.3991/ijep.v13i8.45621

Azaiz, I., Kiesler, N., & Strickroth, S. (2024). Feedback-Generation for Programming Exercises With GPT-4, 31–37. https://doi.org/10.1145/3649217.3653594

Azcona, D. (2017). Educational Analytics in Computer Science. Using Learning Analytics to Support the Enhancement of Teaching and Learning in Higher Education, National Forum for the Enhancement of Teaching and Learning in Higher Education.

Azcona, D., Hsiao, I.-H., & Smeaton, A. F. (2018). Personalizing Computer Science Education by Leveraging Multimodal Learning Analytics. *2018 IEEE Frontiers in Education Conference (FIE)*, 1–9. https://doi.org/10.1109/FIE.2018.8658596

Balderas, A., Palomo-Duarte, M., Dodero, J. M., Ibarra-Sáiz, M., & Rodríguez-Gómez, G. (2018). Scalable authentic assessment of collaborative work assignments in wikis. *International Journal of Educational Technology in Higher Education*, *15*. https://doi.org/10.1186/s41239-018-0122-1

Barana, A., Fissore, C., & Marchisio, M. (2020). From Standardized Assessment to Automatic Formative Assessment for Adaptive Teaching. *Proceedings of the 12th International Conference on Computer Supported Education*, 285–296. https://doi.org/10.5220/0009577302850296

Barra, E., López-Pernas, S., Alonso, Á., Sánchez-Rada, J. F., Gordillo, A., & Quemada, J. (2020). Automated Assessment in Programming Courses: A Case Study during the COVID-19 Era. *Sustainability*, *12*(18). https://doi.org/10.3390/su12187451

Bauer, E., Richters, C., Pickal, A., Klippert, M., Sailer, M., & Stadler, M. (2025). Effects of AI-generated adaptive feedback on statistical skills and interest in statistics: A field experiment in higher education. *British Journal of Educational Technology*, *56*, 1735–1757. https://doi.org/10.1111/bjet.13609

Becker, B. (2016). An Effective Approach to Enhancing Compiler Error Messages. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 126–131. https://doi.org/10.1145/2839509.2844584

Becker, B. (2019). A survey of Introductory Programming Courses in Ireland. *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, 58–64. https://doi.org/10.1145/3304221.3319752

Becker, B., & Fitzpatrick, T. (2019). What Do CS1 Syllabi Reveal About Our Expectations of Introductory Programming Students?, 1011–1017. https://doi.org/10.1145/3287324.3287485

Becker, B., & Glanville, G. (2016). Effective Compiler Error Message Enhancement for Novice Programming Students. *Faculty Research*, *1*. https://arc.cct.ie/fac%5C_research/1

Becker, B., Glanville, G., Iwashima, R., Mcdonnell, C., Goslin, K., & Mooney, C. (2016). Effective compiler error message enhancement for novice programming students. *Computer Science Education*, 1–28. https://doi.org/10.1080/08993408.2016.1225464

Bengtsson, M. (2016). How to plan and perform a qualitative study using content analysis. *NursingPlus Open*, *2*, 8–14. https://doi.org/10.1016/j.npls.2016.01.001

Benotti, L., Martnez, M. C., & Schapachnik, F. (2018). A tool for introducing computer science with automatic formative assessment. *IEEE Transactions on Learning Technologies*, *11*(2), 179–192. https://doi.org/10.1109/TLT.2017.2682084

Ben-Yaacov, A., & Hershkovitz, A. (2022). Types of Errors in Block Programming: Driven by Learner, Learning Environment. *Journal of Educational Computing Research*. https://doi.org/10.1177/07356331221102312

Bey, A., Jermann, P., & Dillenbourg, P. (2018). A Comparison between Two Automatic Assessment Approaches for Programming: An Empirical Study on MOOCs. *Journal of Educational Technology & Society*, *21*(2), 259–272. Retrieved April 11, 2023, from http://www.jstor.org/stable/26388406

Biggs, J. (2003, January). *Teaching for Quality Learning at University*.

Bimpeh, Y. (2024). AI Powered Adaptive Formative Assessment: Validity and Reliability Evaluation. In P. Ilic, I. Casebourne, & R. Wegerif (Eds.), *Artificial intelligence in education: The intersection of technology and*

pedagogy (pp. 127–144). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-71232-6_8

Bockmon, R., Cooper, S., Gratch, J., & Dorodchi, M. (2020). Validating a CS attitudes instrument, 899–904. https://doi.org/10.1145/3328778.3366830

Bosch, N., & D'Mello, S. (2015). The Affective Experience of Novice Computer Programmers. *International Journal of Artificial Intelligence in Education*, *27*. https://doi.org/10.1007/s40593-015-0069-5

Bosse, Y., & Gerosa, M. A. (2017). Why is programming so difficult to learn? Patterns of Difficulties Related to Programming Learning Mid-Stage. *SIGSOFT Softw. Eng. Notes*, *41*(6), 1–6. https://doi.org/10.1145/3011286.3011301

Bosse, Y., Redmiles, D., & Gerosa, M. A. (2019). Pedagogical Content for Professors of Introductory Programming Courses. *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, 429–435. https://doi.org/10.1145/3304221.3319776

Boud, D., & Dochy, F. (2010). Assessment 2020. Seven propositions for assessment reform in higher education. https://api.semanticscholar.org/CorpusID:156837742

Bowman, N. A., Jarratt, L., Culver, K., & Segre, A. M. (2019). How prior programming experience affects students' pair programming experiences and outcomes. *Proceedings of the 2019 ACM Conference on innovation and technology in computer science education*, 170–175.

Brenner, C. (2022). Self-regulated learning, self-determination theory and teacher candidates' development of competency-based teaching practices. *Smart Learning Environments*, *9*. https://doi.org/10.1186/s40561-021-00184-5

Brown, N. C. C., & Wilson, G. (2018). Ten quick tips for teaching programming. *PLOS Computational Biology*, *14*(4), 1–8. https://doi.org/10.1371/journal.pcbi.1006023

Broy, M., Brucker, A. D., Fantechi, A., Gleirscher, M., Havelund, K., Kuppe, M. A., Mendes, A., Platzer, A., Ringert, J. O., & Sullivan, A. (2024). Does Every Computer Scientist Need to Know Formal Methods? *Form. Asp. Comput.*, *37*(1). https://doi.org/10.1145/3670795

Burgess, A. W., Roberts, C., Black, K. I., & Mellis, C. (2013). Senior medical student perceived ability and experience in giving peer feedback in formative long case examinations. *BMC Medical Education*, *13*(1), 07356331221102312. https://doi.org/10.1186/1472-6920-13-79

Cardoso, M., de Castro, A. V., Rocha, Á., Silva, E., & Mendonca, J. C. F. (2020). Use of Automatic Code Assessment Tools in the Programming Teaching Process. *International Computer Programming Education Conference*. https://api.semanticscholar.org/CorpusID:221593429

Carless, D. (2014). Exploring learning-oriented assessment processes. *Higher Education*, *69*. https://doi.org/10.1007/s10734-014-9816-z

CERI, for Educational Research, C., & Innovation. (2012). Assessment for learning formative assessment. *OECD/CERI International Conference "Learning in the 21st Century: Research, Innovation and Policy"*, 25. https://www.oecd.org/site/educeri21st/40600533.pdf

Charles, T., & Gwilliam, C. (2023). The Effect of Automated Error Message Feedback on Undergraduate Physics Students Learning Python: Reducing Anxiety and Building Confidence. *Journal for STEM Education Research*, *6*(2), 326–357. https://doi.org/10.1007/s41979-022-00084-4

Chassignol, M., Khoroshavin, A., Klimova, A., & Bilyatdinova, A. (2018). Artificial Intelligence trends in education: a narrative overview. *Procedia Computer Science*, *136*, 16–24. https://doi.org/10.1016/j.procs.2018.08.233

Chatzopoulou, D., & Economides, A. (2010). Adaptive assessment of student's knowledge in programming courses. *J. Comp. Assisted Learning*, *26*, 258–269. https://doi.org/10.1111/j.1365-2729.2010.00363.x

Check, J., & Schutt, R. (2012). *Research Methods in Education*. https://doi.org/10.4135/9781544307725

Chen, X., Breslow, L., & DeBoer, J. (2018). Analyzing productive learning behaviors for students using immediate corrective feedback in a blended learning environment. *Computers & Education*, *117*, 59–74. https://doi.org/https://doi.org/10.1016/j.compedu.2017.09.013

Choi, Y., & McClenen, C. (2020). Development of Adaptive Formative Assessment System Using Computerized Adaptive Testing and Dynamic Bayesian Networks. *Applied Sciences*, *10*(22). https://doi.org/10.3390/app10228196

Cipriano, B. P., Fachada, N., & Alves, P. (2022). Drop Project: An automatic assessment tool for programming assignments. *SoftwareX*, *18*, 101079. https://doi.org/https://doi.org/10.1016/j.softx.2022.101079

Combéfis, S. (2022). Automated Code Assessment for Education: Review, Classification and Perspectives on Techniques and Tools. *Software*, *1*(1), 3–30. https://doi.org/10.3390/software1010002

Conley, D. T., & French, E. M. (2014). Student Ownership of Learning as a Key Component of College Readiness. *American Behavioral Scientist*, *58*(8), 1018–1034. https://doi.org/10.1177/0002764213515232

Cooper, C., Booth, A., Campbell, J., Britten, N., & Garside, R. (2018). Defining the process to literature searching in systematic reviews:A literature review of guidance and supporting studies. *BMC Medical Research Methodology*, *18*. https://doi.org/10.1186/s12874-018-0545-3

Cosyn, E., Uzun, H., Doble, C., & Matayoshi, J. (2021). A practical perspective on knowledge space theory: ALEKS and its data. *Journal of Mathematical Psychology*, *101*, 102512. https://doi.org/https://doi.org/10.1016/j.jmp.2021.102512

Creswell, J. W. (2014). Research design: Qualitative, quantitative, and mixed methods approaches.

Cucuiat, V., & Waite, J. (2024). Feedback Literacy: Holistic Analysis of Secondary Educators' Views of LLM Explanations of Program Error Messages. https://doi.org/https://doi.org/10.1145/3649217.3653595

Culligan, N., & Casey, K. (2018). Building an Authentic Novice Programming Lab Environment. *Irish Conference on Engaging Pedagogy*, 13–14.

Cutting, V., & Stephen, N. (2021). A Review on using Python as a Preferred Programming Language for Beginners. *8*, 4258–4263.

Dāboliņš, J., & Grundspenkis, J. (2013). The Role of Feedback in Intelligent Tutoring System. *Applied Computer Systems*, *14*, 88–93. https://doi.org/10.2478/acss-2013-0011

Darrell Evans, R. S., Paul Zeun. (2013). Motivating student learning using a formative assessment journey. *J Anat*, *224*(3), 296–303.

Das Roy, R., Das Gupta, S., & Das, D. (2025). Chat GPT, Gemini or Meta AI: A comparison of AI platforms as a tool for answering higher-order questions in microbiology. *Journal of Postgraduate Medicine*, *71*. https://doi.org/10.4103/jpgm.jpgm_775_24

Dawson, P., Henderson, M., Mahoney, P., Phillips, M., Ryan, T., Boud, D., & Molloy, E. (2018). What makes for effective feedback: Staff and student perspectives. *Assessment & Evaluation in Higher Education*, *44*. https://doi.org/10.1080/02602938.2018.1467877

Delgado-Pérez, P., & Medina-Bulo, I. (2020). Customizable and scalable automated assessment of C/C++ programming assignments. *Computer Applications in Engineering Education*, *28*, 1449–1466.

Denny, P., Prather, J., Becker, B. A., Mooney, C., Homer, J., Albrecht, Z. C., & Powell, G. B. (2021). On Designing Programming Error Messages for Novices: Readability and Its Constituent Factors. *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3411764.3445696

Dhakshina Moorthy, A., & Dhakshina Moorthy, K. (2020). Online coding event as a formative assessment tool in Introductory Programming and algorithmic courses—An exploration study. *Computer Applications in Engineering Education*, *28*, 1580–1590.

Dietrich, S. W., Goelman, D., Broatch, J., Crook, S. M., Ball, B., Kobojek, K., & Ortiz, J. (2020). Using Formative Assessment for Improving Pedagogy: Reflections on Feedback Informing Database Visualizations. *ACM Inroads*, *11*(4), 27–34. https://doi.org/10.1145/3430766

Dirzyte, A., Perminas, A., Kaminskis, L., Zebrauskas, G., Sederevičiūtė-Pačiauskienė, Ž., Šliogerienė, J., Suchanova, J., Rimasiute-Knabikiene, R., Patapas, A., & Gajdosikienė, I. (2023). Factors contributing to dropping out of adults' programming e-learning. *Heliyon*, *9*, e22113. https://doi.org/10.1016/j.heliyon.2023.e22113

Djenno, M., Insua, G. M., & Pho, A. (2022). From paper to pixels: using Google Forms for collaboration and assessment. *32*(4), 9–13. https://doi.org/10.1108/LHTN-12-2014-0105

Ettles, A., Luxton-Reilly, A., & Denny, P. (2018). Common Logic Errors Made by Novice Programmers. *Proceedings of the 20th Australasian Computing Education Conference*, 83–89. https://doi.org/10.1145/3160489.3160493

Evans, C. A., & Wilson, S. (2020). Development of an automated assessment and feedback platform.

Faherty, R., Quille, K., Vivian, R., McGill, M. M., Becker, B. A., & Nolan, K. (2021). Comparing programming self-esteem of upper secondary school teachers to CS1 students. *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, 554–560. https://doi.org/10.1145/3430665.3456372

Falmagne, J.-C., Albert, D., Doble, C., Eppstein, D., & Hu, X. (2013, January). *Knowledge Spaces: Applications in Education*. https://doi.org/10.1007/978-3-642-35329-1

Faradilla, T., Abdul Rahman, T. F., Anuar, N., Mohd Said, R. F., Said, M., & Safiai, S. (2019). How a proposed ratio of Bloom's taxonomy enhances learning in C programming.

Farias, B., Menezes, R., de Lima Filho, E. B., Sun, Y., & Cordeiro, L. C. (2024). ESBMC-Python: A Bounded Model Checker for Python Programs. *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 1836–1840. https://doi.org/10.1145/3650212.3685304

Feudel, F., & Unger, A. (2022). Students' Strategic Usage of Formative Quizzes in an Undergraduate Course in Abstract Algebra. *International Journal of Research in Undergraduate Mathematics Education*, *10*, 1–29. https://doi.org/10.1007/s40753-022-00194-9

Field, A. P. (2017). Discovering statistics using ibm spss statistics. https://dl.acm.org/doi/abs/10.5555/2502692

Figueiredo, J., & García-Peñalvo, F. (2020). Increasing student motivation in computer programming with gamification, 997–1000. https://doi.org/10.1109/EDUCON45650.2020.9125283

Fisher Jr, W. P. (2000). Survey design recommendations. *Popular Measurement*, *3*(1), 58–60.

Fuller, U., Johnson, C. G., Ahoniemi, T., Cukierman, D., Hernán-Losada, I., Jackova, J., Lahtinen, E., Lewis, T. L., Thompson, D. M., Riedesel, C., & Thompson, E. (2007). Developing a computer science-specific learning taxonomy. *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, 152–170. https://doi.org/10.1145/1345443.1345438

Gabbay, H., & Cohen, A. (2022). Investigating the effect of Automated Feedback on learning behavior in MOOCs for programming. *Proceedings of the 15th International Conference on Educational Data Mining*, 376–383. https://doi.org/10.5281/zenodo.6853125

Ghosh, S., Brooks, B., Ranmuthugala, D., & Bowles, M. (2020). Authentic Versus Traditional Assessment: An Empirical Study Investigating the Difference in Seafarer Students' Academic Achievement. *Journal of Navigation*, *73*, 1–16. https://doi.org/10.1017/S0373463319000894

Glavind, J. G., Oca, L. M. D., Pechmann, P., Sejersen, D. B., & Iskov, T. (2023). Student-centred learning and teaching: A systematic mapping review of empirical research. *Journal of Further and Higher Education*, *47*(9), 1247–1261. https://doi.org/10.1080/0309877X.2023.2241391

Goel, A., & Joyner, D. (2016). Formative assessment and implicit feedback in online learning. https://www.davidjoyner.net/blog/formative-assessment-and-implicit-feedback-in-online-learning/

González, R. (2017). ICE: An Automated Tool for Teaching Advanced C Programming. *International Association for Development of the Information Society*.

Gordillo, A. (2019). Effect of an Instructor-Centered Tool for Automatic Assessment of Programming Assignments on Students' Perceptions and Performance. *Sustainability*, *11*(20). https://doi.org/10.3390/su11205568

Goto, T., Kano, K., & Shiose, T. (2023). Students' acceptance on computer-adaptive testing for achievement assessment in japanese elementary and secondary school. *Frontiers in Education*, *8*. https://doi.org/10.3389/feduc.2023.1107341

Grant, P. F. (2017). Formative test-driven development for programming practicals. *AISHE-J: The All Ireland Journal of Teaching and Learning in Higher Education*, *9*.

Grawemeyer, B., Halloran, J., England, M., & Croft, D. (2022). Feedback and engagement on an introductory programming module. *CoRR*, *abs/2201.01240*. https://arxiv.org/abs/2201.01240

Grover, S. (2021). Toward A Framework for Formative Assessment of Conceptual Learning in K-12 Computer Science Classrooms, 31–37. https://doi.org/10.1145/3408877.3432460

Gupta, S., & Gupta, A. (2018). E-Assessment Tools for Programming Languages: A Review. *International Conference on Information Technology and Knowledge Management*.

Hainey, T., Baxter, G., Black, J., Yorke, K., Bernikas, J., Chrzanowska, N., & McAulay, F. (2022). Serious games as innovative formative assessment tools for programming in higher education. *ECGBL, 16th European Conference on Games Based Learning, 6 - 7 October 2022, Lisbon, Portugal*. https://www.academic-conferences.org/conferences/ecgbl/

Hamamoto Filho, P., Ii, E., Ribeiro, Z., Hafner, M., Cecilio Fernandes, D., & Bicudo, A. (2020). Relationships between Bloom's taxonomy, judges' estimation of item difficulty and psychometric properties of items from a progress test: a prospective observational study. *Sao Paulo Medical Journal*, *138*, 33–39. https://doi.org/10.1590/1516-3180.2019.0459.R1.19112019

Hao, Q., Smith, D., Ding, L., Ko, A., Ottaway, C., Wilson, J., Arakawa, K., Turcan, A., Poehlman, T., & Greer, T. (2021). Towards understanding the effective design of automated formative feedback for programming assignments. *Computer Science Education*. https://doi.org/10.1080/08993408.2020.1860408

Hao, Q., Wilson, J., Ottaway, C., Iriumi, N., Arakawa, K., & Smith, D. (2019). Investigating the Essential of Meaningful Automated Formative Feedback for Programming Assignments. https://doi.org/10.1109/VLHCC.2019.8818922

Hasan, M. M., Adnan, M., & Waheed, B. (2021). Use of an Adaptive e-Learning Platform as a Formative Assessment Tool in the Cardiovascular System Course Component of an MBBS Programme [Letter]. *Advances in Medical Education and Practice*, *12*, 111–112. https://doi.org/10.2147/AMEP.S299759

Hattie, J., & Timperley, H. (2007). The Power of Feedback. *Review of Educational Research*, *77*, 81–112. https://doi.org/10.3102/003465430298487

Heck, D. W., & Noventa, S. (2020). Representing probabilistic models of knowledge space theory by multinomial processing tree models. *Journal of Mathematical Psychology*, *96*, 102329. https://doi.org/https://doi.org/10.1016/j.jmp.2020.102329

Heitmann, S., Grund, A., Berthold, K., Fries, S., & Roelle, J. (2018). Testing Is More Desirable When It Is Adaptive and Still Desirable When Compared to Note-Taking. *Frontiers in Psychology*, *Volume 9 - 2018*. https://doi.org/10.3389/fpsyg.2018.02596

Hertz, M. (2010). What do "CS1" and "CS2" mean? Investigating differences in the early courses. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 199–203. https://doi.org/10.1145/1734263.1734335

Hoffman, H. J., & Elmi, A. F. (2021). Do Students Learn More from Erroneous Code? Exploring Student Performance and Satisfaction in an Error-Free Versus an Error-full SASÂ® Programming Environment. *Journal of Statistics and Data Science Education*, *29*(3), 228–240. https://doi.org/10.1080/26939169.2021.1967229

Hogan, E., Li, R., & Soosai Raj, A. G. (2023). CS0 vs. CS1: Understanding fears and confidence amongst non-majors in introductory CS courses. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 25–31. https://doi.org/10.1145/3545945.3569865

Holmes, W., Bialik, M., & Fadel, C. (2019, March). *Artificial Intelligence in Education. Promise and Implications for Teaching and Learning.*

Howland, J., Wright, T., Boughan, R., & Roberts, B. (2009). How Scholarly Is Google Scholar? A Comparison to Library Databases. *College & Research Libraries*, *70*, 227–234. https://doi.org/10.5860/crl.70.3.227

Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Börstler, J., Edwards, S. H., Isohanni, E., Korhonen, A., Petersen, A., Rivers, K., Rubio, M. Á., Sheard, J., Skupas, B., Spacco, J., Szabo, C., & 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

Ihichr, A., Oustous, O., Idrissi, Y., & Ait Lahcen, A. (2024). A Systematic Review on Assessment in Adaptive Learning: Theories, Algorithms and Techniques. *International Journal of Advanced Computer Science and Applications*, *15*. https://doi.org/10.14569/IJACSA.2024.0150785

Inventado, P. S. (2019). Promoting Mastery Learning in an Introductory Programming Course. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 1285. https://doi.org/10.1145/3287324.3293780

Islam, N., Sheikh, G., Fatima, R., & Alvi, F. (2019). A Study of Difficulties of Students in Learning Programming. *Journal of Education & Social Sciences*, *7*, 38–46. https://doi.org/10.20547/jess0721907203

Ismail, S. M., Rahul, D. R., Patra, I., & Rezvani, E. (2022). Formative vs. summative assessment: Impacts on academic motivation, attitude toward learning, test anxiety, and self-regulation skill. *Language Testing in Asia*, *12*(1), 40. https://doi.org/10.1186/s40468-022-00191-4

Jansen, J., Oprescu, A., & Bruntink, M. (2017). The impact of automated code quality feedback in programming education. *Post-proceedings of the Tenth Seminar on Advanced Techniques and Tools for Software Evolution (SATToSE)*, *210*.

Jha, M., Shivshankar, S., Thakur, S., & Jha, S. (2023). Changes Formative Assessments for Programming Units. *Universal Journal of Educational Research*, *11*, 148–158. https://doi.org/10.13189/ujer.2023.110802

Johnson, F., McQuistin, S., & O'Donnell, J. (2020). Analysis of Student Misconceptions Using Python as an Introductory Programming Language. *Proceedings of the 4th Conference on Computing Education Practice*. https://doi.org/10.1145/3372356.3372360

Jones, T., Baxter, M., & Khanduja, V. (2013). A quick guide to survey research. *Annals of the Royal College of Surgeons of England*, *95*, 5–7. https://doi.org/10.1308/003588413X13511609956372

Kabir, S. M. (2016). Methods of data collection.

Kadar, R., Wahab, N., Othman, J., Shamsuddin, M., & Mahlan, S. (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*. https://doi.org/10.6007/IJARPED/v10-i3/11100

Kalelioğlu, F., & Gülbahar, Y. (2014). The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners' Perspective. *Informatics Educ.*, *13*, 33–50. https://api.semanticscholar.org/CorpusID:7458490

Kallia, M. (2017). Assessment in Computer Science courses: A Literature Review.

Katsaris, I., & Vidakis, N. (2021). Adaptive e-learning systems through learning styles: A review of the literature. *Advances in Mobile Learning Educational Research*, *1*, 124–145. https://doi.org/10.25082/AMLER.2021.02.007

Keuning, H., Alpizar-Chacon, I., Lykourentzou, I., Beehler, L., Köppe, C., de Jong, I., & Sosnovsky, S. (2024). Students' Perceptions and Use of Generative AI Tools for Programming Across Different Computing Courses. https://arxiv.org/abs/2410.06865

Keuning, H., Jeuring, J., & Heeren, B. (2018). A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Transactions on Computing Education*, *19*, 1–43. https://doi.org/10.1145/3231711

King, A. (2017). Using Kahoot! *The Australian Mathematics Teacher*, *73*(4), 35–36. https://search.informit.org/doi/10.3316/informit.298530223534343

Kitchenham, B., & Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. *2*.

Kolluru, V., Mungara, S., & Chintakunta, A. N. (2018). Adaptive Learning Systems: Harnessing AI for Customized Educational Experiences. *International Journal of Computational Science and Information Technology*, *6*, 13–26. https://doi.org/10.5121/ijcsity.2018.6302

Koorsse, M., Cilliers, C., & Calitz, A. P. (2015). Programming assistance tools to support the learning of IT programming in South African secondary schools. *Comput. Educ.*, *82*, 162–178.

Kotronoulas, G., Miguel, S., Dowling, M., Fernández-Ortega, P., Colomer-Lahiguera, S., Gülcan Bağçivan, Pape, E., Drury, A., Semple, C., Dieperink, K. B., & Papadopoulou, C. (2023). An Overview of the Fundamentals of Data Management, Analysis, and Interpretation in Quantitative Research [Special Issue on Research Scholarship]. *Seminars in Oncology Nursing*, *39*(2), 151398. https://doi.org/https://doi.org/10.1016/j.soncn.2023.151398

Koulouri, T., Lauria, S., & Macredie, R. D. (2015). Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches. *ACM Trans. Comput. Educ.*, *14*(4). https://doi.org/10.1145/2662412

Koutcheme, C., Dainese, N., Sarsa, S., Hellas, A., Leinonen, J., Ashraf, S., & Denny, P. (2025). Evaluating language models for generating and judging programming feedback. *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, 624–630. https://doi.org/10.1145/3641554.3701791

Koutcheme, C., & Hellas, A. (2024). Propagating Large Language Models Programming Feedback. *Proceedings of the Eleventh ACM Conference on Learning @ Scale*, 366–370. https://doi.org/10.1145/3657604.3664665

Kővári, A., & Katona, J. (2023). Effect of software development course on programming self-efficacy. *Education and Information Technologies*, *28*(9), 10937–10963. https://doi.org/10.1007/s10639-023-11617-8

Le, N.-T. (2016). A Classification of Adaptive Feedback in Educational Systems for Programming. *Systems*, *4*(2). https://doi.org/10.3390/systems4020022

Leino, K. R. M., & Monahan, R. (2007). Automatic verification of textbook programs. *ECOOP workshop on Formal Techniques for Java-like Programs*.

Ling, H.-C., Hsiao, K.-L., & Hsu, W.-C. (2021). Can Students' Computer Programming Learning Motivation and Effectiveness Be Enhanced by Learning Python Language? A Multi-Group Analysis. *Frontiers in Psychology*, *11*, 600814. https://doi.org/10.3389/fpsyg.2020.600814

Lishinski, A., & Yadav, A. (2021). Self-evaluation Interventions: Impact on Self-efficacy and Performance in Introductory Programming. *ACM Transactions on Computing Education (TOCE)*, *21*, 1–28.

Lobb, R., & Harlow, J. (2016). Coderunner: A Tool for Assessing Computer Programming Skills. *ACM Inroads*, *7*(1), 47–51. https://doi.org/10.1145/2810041

Lohiniva, M., & Isomöttönen, V. (2021). Novice Programming Students' Reflections on Study Motivation during COVID-19 Pandemic, 1–9. https://doi.org/10.1109/FIE49875.2021.9637367

Lohr, D., Berges, M., Chugh, A., & Striewe, M. (2024). Adaptive Learning Systems in Programming Education: A Prototype for Enhanced Formative Feedback. https://doi.org/10.18420/delfi2024_57

Lohr, D., Keuning, H., & Kiesler, N. (2024). You're (Not) My Type – Can LLMs Generate Feedback of Specific Types for Introductory Programming Tasks? https://arxiv.org/abs/2412.03516

Loksa, D., Ko, A. J., Jernigan, W., Oleson, A., Mendez, C. J., & Burnett, M. M. (2016). Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance. *Proceedings of the 2016 CHI Conference on Human*

Factors in Computing Systems, 1449–1461. https://doi.org/10.1145/2858036.2858252

Loksa, D., Margulieux, L., Becker, B. A., Craig, M., Denny, P., Pettit, R., & Prather, J. (2022). Metacognition and Self-Regulation in Programming Education: Theories and Exemplars of Use. *ACM Trans. Comput. Educ.*, *22*(4). https://doi.org/10.1145/3487050

Lombardi, M. (2022). Making the Grade: The Role of Assessment in Authentic Learning.

Louhab, F. E., Bahnasse, A., & Talea, M. (2018). Towards an Adaptive Formative Assessment in Context-Aware Mobile Learning [The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life]. *Procedia Computer Science*, *135*, 441–448. https://doi.org/https://doi.org/10.1016/j.procs.2018.08.195

Lukose, J. M., & Mammen, K. J. (2018). Enhancing Academic Achievement in an Introductory Computer Programming Course through the Implementation of Guided Inquiry-Based Learning and Teaching. *Asia-Pacific Forum on Science Learning and Teaching*, *19*(2).

Luo, J., Zheng, C., Yin, J., & Teo, H. (2025). Design and assessment of AI-based learning tools in higher education: A systematic review. *International Journal of Educational Technology in Higher Education*, *22*. https://doi.org/10.1186/s41239-025-00540-2

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., & Szabo, C. (2018). Introductory programming: A systematic literature review. *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, 55–106. https://doi.org/10.1145/3293881.3295779

Luxton-Reilly, A., Tempero, E., Arachchilage, N., Chang, A., Denny, P., Fowler, A., Giacaman, N., Kontorovich, I., Lottridge, D., Manoharan, S., Sindhwani, S., Singh, P., Speidel, U., Stephen, S., Terragni, V., Whalley, J., Wuensche, B., & Ye, X. (2023). Automated Assessment: Experiences From the Trenches. *Proceedings of the 25th Australasian Computing Education Conference*, 1–10. https://doi.org/10.1145/3576123.3576124

Luxton-Reilly, A., Whalley, J., Becker, B., Yingjun, C., McDermott, R., Mirolo, C., Mühling, A., Petersen, A., & Sanders, K. (2017). Developing Assessments to Determine Mastery of Programming Fundamentals, 47–69. https://doi.org/10.1145/3174781.3174784

Maggin, D. M., Barton, E., Reichow, B., Lane, K. L., & Shogren, K. A. (2022). Commentary on the What Works Clearinghouse Standards and Procedures Handbook (v. 4.1) for the Review of Single-Case Research. *Remedial and Special Education*, *43*(6), 421–433. https://doi.org/10.1177/07419325211051317

Mahmoud, A. T., Mohammed, A. A., Ayman, M., Medhat, W., Selim, S., Zayed, H., Yousef, A. H., & Elaraby, N. (2024). Formal Verification of Code Conversion: A Comprehensive Survey. *Technologies*, *12*(12). https://doi.org/10.3390/technologies12120244

Mahon, J., Mac Namee, B., & Becker, B. A. (2024). Guidelines for the Evolving Role of Generative AI in Introductory Programming Based on Emerging Practice. *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, 10–16. https://doi.org/10.1145/3649217.3653602

Malik, S. I., & Coldwell-Neilson, J. (2017). A model for teaching an introductory programming course using adri. *Education and Information Technologies*, *22*, 1089–1120.

Marchisio, M., Tiziana, M., & Sacchet, M. (2020). Automatic Formative Assessment in Computer Science: Guidance to Model-Driven Design, 201–206. https://doi.org/10.1109/COMPSAC48688.2020.00035

Margulieux, L. E., Morrison, B. B., & Decker, A. (2020). Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples. *International Journal of STEM Education*, *7*, 1–16.

Marwan, S., Akram, B., Barnes, T., & Price, T. (2022). Adaptive Immediate Feedback for Block-Based Programming: Design and Evaluation. *IEEE Transactions on Learning Technologies*, *15*, 1–15. https://doi.org/10.1109/TLT.2022.3180984

Marwan, S., Gao, G., Fisk, S., Price, T. W., & Barnes, T. (2020). Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science. *Proceedings of the 2020 ACM Conference on International Computing Education Research*, 194–203. https://doi.org/10.1145/3372782.3406264

Matthews, R. (2025). Transforming Formative Assessment Method in Introductory Programming Course With ChatGPT, 463–475. https://doi.org/10.22492/issn.2435-5240.2025.39

McCall, D., & Kölling, M. (2014). Meaningful Categorisation of Novice Programmer Errors. *Proceedings - Frontiers in Education Conference, FIE*, *2015*. https://doi.org/10.1109/FIE.2014.7044420

McMillan, J., & Schumacher, S. (2014). *Research in Education: Evidence-Based Inquiry*. Pearson Education. https://books.google.ie/books?id=JSavAgAAQBAJ

McSweeney, K. (2012). Assessment for Learning: from Theory to Practice. https://www.teachingcouncil.ie/en/_fileupload/research/bursary-summaries/web-final-teaching-council-report-2012-kathryn-mc-sweeney.pdf

Mejeh, M. (2024). Effects of Adaptive Feedback Through a Digital Tool: A Mixed-Methods Study on the Course of Self-Regulated Learning (Poster 41). https://doi.org/10.3102/IP.24.2107031

Mekterovic, I., Brkic, L., Milašinović, B., & Baranovic, M. (2020). Building a Comprehensive Automated Programming Assessment System. *IEEE Access*, *PP*, 1–1. https://doi.org/10.1109/ACCESS.2020.2990980

Meline, T. (2006). Selecting Studies for Systemic Review: Inclusion and Exclusion Criteria. *Contemporary Issues in Communication Science and Disorders*, *33*(Spring), 21–27. https://doi.org/10.1044/cicsd_33_S_21

Mertens, D. (2019). *Research and Evaluation in Education and Psychology: Integrating Diversity with Quantitative, Qualitative, and Mixed Methods 5th edition.*

Misirli, A., & Komis, V. (2023). Computational thinking in early childhood education: The impact of programming a tangible robot on developing debugging knowledge. *Early Childhood Research Quarterly*, *65*, 139–158. https://doi.org/https://doi.org/10.1016/j.ecresq.2023.05.014

Mitrovic, A., Ohlsson, S., & Barrow, D. (2013). The effect of positive feedback in a constraint-based intelligent tutoring system. *Computers & Education*, *60*, 264–272. https://doi.org/10.1016/j.compedu.2012.07.002

Mohamad, M., & Omar, A. (2017). Measuring Cognitive Performance on programming Knowledge: Classical Test Theory versus Item Response Theory, 361–365. https://doi.org/10.1109/WEEF.2017.8467047

Mohamed Shuhidan, S., Hamilton, M., & D'Souza, D. (2010). Instructor perspectives of multiple-choice questions in summative assessment for novice programmers. *Computer Science Education*, *20*, 229–259. https://doi.org/10.1080/08993408.2010.509097

Moon, H., Cheon, J., & Kwon, K. (2022). Difficult Concepts and Practices of Computational Thinking Using Block-based Programming. *International Journal of Computer Science Education in Schools*, *5*, 3–16. https://doi.org/10.21585/ijcses.v5i3.129

Moorhouse, B. L., Yeo, M. A., & Wan, Y. (2023). Generative AI tools and assessment: Guidelines of the world's top-ranking universities. *Computers*

and Education Open, *5*, 100151. https://doi.org/https://doi.org/10.1016/j.caeo.2023.100151

Näsström, G., Andersson, C., Granberg, C., Palm, T., & Palmberg, B. (2021). Changes in student motivation and teacher decision making when implementing a formative assessment practice. *Frontiers in Education*, *6*. https://doi.org/10.3389/feduc.2021.616216

Nguyen, H., & Allan, V. (2024). Using GPT-4 to Provide Tiered, Formative Code Feedback. *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 958–964. https://doi.org/10.1145/3626252.3630960

Nunes, R., Cruz, G., Pedrosa, D., Maia, A., Morgado, L., Paredes, H., Cravino, J., & Martins, P. (2021). Motivating Students to Learn Computer Programming in Higher Education: The SimProgramming Approach. https://doi.org/10.1007/978-3-030-73988-1_41

Orozco-Garcia, L., Gonzalez, C., Montano, J., Mondragon, C., & Tobar-Munoz, H. (2019). A formative assessment tool to support computational thinking in the classroom. *2019 International Conference on Virtual Reality and Visualization (ICVRV)*, 185–188. https://doi.org/10.1109/ICVRV47840.2019.00043

Pădurean, V.-A., Phung, T., Kotalwar, N., Liut, M., Leinonen, J., Denny, P., & Singla, A. (2025). Humanizing Automated Programming Feedback: Fine-Tuning Generative Models with Student-Written Feedback. https://arxiv.org/abs/2509.10647

Pankiewicz, M. (2021). Assessing the Cold Start Problem in Adaptive Systems. *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 2*, 650. https://doi.org/10.1145/3456565.3460053

Papanastasiou, E. (2021). Adaptive Assessment. In R. Gunstone (Ed.), *Encyclopedia of science education* (pp. 1–2). Springer Netherlands. https://doi.org/10.1007/978-94-007-6165-0_3-4

Petrillo, F., Spritzer, A., Freitas, C., & Pimenta, M. (2011). Interactive analysis of likert scale data using a multichart visualization tool, 358–365.

Pettit, R., Homer, J., Holcomb, K., Simone, N., & Mengel, S. (2015). Are automated assessment tools helpful in programming courses? *ASEE Annual Conference and Exposition, Conference Proceedings*, *122*.

Pfitzner-Eden, F. (2016). Why Do I Feel More Confident? Bandura's Sources Predict Preservice Teachers' Latent Changes in Teacher Self-Efficacy. *Frontiers in Psychology*, *Volume 7 - 2016*. https://doi.org/10.3389/fpsyg.2016.01486

Phelan, J. (2019). Review of Adaptive Testing, Mastery-based Learning and their Educational Applications. http : / / download . lww . com / efficacy / WP _ Review_of_Adaptive_Testing.pdf

Piteira, M., & Costa, C. (2013). Learning computer programming: Study of difficulties in learning programming. *Proceedings of the 2013 International Conference on Information Systems and Design of Communication*, 75–80. https://doi.org/10.1145/2503859.2503871

Polito, G., Temperini, M., & Sterbini, A. (2019). 2TSW: Automated Assessment of Computer Programming Assignments, in a Gamified Web Based System. *2019 18th International Conference on Information Technology Based Higher Education and Training (ITHET)*, 1–9. https : / / doi . org / 10 . 1109 / ITHET46829.2019.8937377

Prather, J. E. (2018). Beyond Automated Assessment: Building Metacognitive Awareness in Novice Programmers in CS1.

Qian, Y., & D.Lehman, J. (2021). Using an automated assessment tool to explore difficulties of middle school students in introductory programming. *Journal of Research on Technology in Education*, *54*, 1–17. https://doi.org/10.1080/15391523.2020.1865220

Qian, Y., & Lehman, J. D. (2019). Using Targeted Feedback to Address Common Student Misconceptions in Introductory Programming: A Data-Driven

Approach. *SAGE Open*, *9*(4), 2158244019885136. https://doi.org/10.1177/2158244019885136

Raj, A. G. S., Patel, J. M., Halverson, R., & Halverson, E. R. (2018). Role of Live-coding in Learning Introductory Programming. *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*. https://doi.org/10.1145/3279720.3279725

Ralabate, P. K. (2011). Universal Design for Learning: Meeting the Needs of All Students. *The ASHA Leader*, *16*, 14–17.

Ramalingam, V., & Wiedenbeck, S. (1998). Development and Validation of Scores on a Computer Programming Self-Efficacy Scale and Group Analyses of Novice Programmer Self-Efficacy. *Journal of Educational Computing Research*, *19*(4), 367–381. https://doi.org/10.2190/C670-Y3C8-LTJ1-CT3P

Reihanian, I., Hou, Y., Chen, Y., & Zheng, Y. (2025, June). A Review of Generative AI in Computer Science Education: Challenges and Opportunities in Accuracy, Authenticity, and Assessment. https://doi.org/10.48550/arXiv.2507.11543

Reiser, B., & Tabak, I. (2014). Scaffolding [Publisher Copyright: © Cambridge University Press 2006, 2014.]. In R. Sawyer (Ed.), *The cambridge handbook of the learning sciences, second edition* (pp. 44–62). Cambridge University Press. https://doi.org/10.1017/CBO9781139519526.005

Renske Weeda, S. S., & Barendsen, E. (2023). Unraveling novices' code composition difficulties. *Computer Science Education*, *0*(0), 1–28. https://doi.org/10.1080/08993408.2023.2169067

Rich, J., Colon, A., Mines, D., & Jivers, K. (2014). Creating learner-centered assessment strategies for promoting greater student retention and class participation. *Frontiers in Psychology*, *5*, 595. https://doi.org/10.3389/fpsyg.2014.00595

Riese, E., & Stenbom, S. (2020). Experiences of Assessment in Introductory Programming From the Perspective of Non-Computer Science Majors. *2020*

IEEE Frontiers in Education Conference (FIE), 1–9. https://doi.org/10.1109/FIE44824.2020.9274060

Rodríguez-Del-Pino, J. C., Royo, E. R., & Figueroa, Z. J. H. (2012). A Virtual Programming Lab for moodle with automatic assessment and anti-plagiarism features.

Rong, Q., Kong, W., Xiao, Y., & Gao, X. (2023). An Adaptive Testing Approach for Competence Using Competence-Based Knowledge Space Theory. *Smart Learning for A Sustainable Society: Proceedings of the 7th International Conference on Smart Learning Environments*, 158–163. https://doi.org/10.1007/978-981-99-5961-7_18

Rosminah, S., & Ali, M. (2012). Difficulties in learning programming: Views of students. https://doi.org/10.13140/2.1.1055.7441

Ross, B., Chase, A.-M., Robbie, D., Oates, G., & Absalom, Y. (2018). Adaptive quizzes to increase motivation, engagement and learning outcomes in a first year accounting unit. *International Journal of Educational Technology in Higher Education*, *15*(1), 30.

Rubin, M. J. (2013). The effectiveness of live-coding to teach introductory programming. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 651–656. https://doi.org/10.1145/2445196.2445388

Rum, S. N. M., & Ismail, M. A. B. (2017). Metacognitive Support Accelerates Computer Assisted Learning for Novice Programmers. *J. Educ. Technol. Soc.*, *20*, 170–181.

Ryan, R. M., & Deci, E. L. (2000). Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions. *Contemporary Educational Psychology*, *25*(1), 54–67. https://doi.org/https://doi.org/10.1006/ceps.1999.1020

Sahu, V., Gupta, G. R., Borikar, R., & Mane, N. (2025). Autograder+: A Multi-Faceted AI Framework for Rich Pedagogical Feedback in Programming Education. https://arxiv.org/abs/2510.26402

Saunders, M., Lewis, P., Thornhill, A., & Bristow, A. (2019). "Research Methods for Business Students" Chapter 4: Understanding research philosophy and approaches to theory development.

Schellekens, L. H., Bok, H. G., de Jong, L. H., Van der Schaaf, M. F., Kremer, W. D., & Van der Vleuten, C. P. (2021). A scoping review on the notions of Assessment as Learning (AaL), Assessment for Learning (AfL), and Assessment of Learning (AoL). *Studies in Educational Evaluation*, *71*, 101094. https://doi.org/https://doi.org/10.1016/j.stueduc.2021.101094

Scholz, N., Nguyen, M. H., Singla, A., & Nagashima, T. (2025, September). Partnering with AI: A Pedagogical Feedback System for LLM Integration Into Programming Education. In *Two decades of tel. from lessons learnt to challenges ahead* (pp. 243–248). Springer Nature Switzerland. https://doi.org/10.1007/978-3-032-03873-9_31

Schulte, C. (2008). Block Model: An Educational Model of Program Comprehension as a Tool for a Scholarly Approach to Teaching. *Proceedings of the Fourth International Workshop on Computing Education Research*, 149–160. https://doi.org/10.1145/1404520.1404535

Scott, G. W. (2017). Active engagement with assessment and feedback can improve group-work outcomes and boost student confidence. *Higher Education Pedagogies*, *2*(1), 1–13. https://doi.org/10.1080/23752696.2017.1307692

Selvaraj, A., Zhang, R. E., Porter, L., & Soosai Raj, A. G. (2021). Live Coding: A Review of the Literature, 164–170. https://doi.org/10.1145/3430665.3456382

Shanley, N., Martin, F., Collins, N., Perez-Quinones, M., Ahlgrim-Delzell, L., Pugalee, D., & Hart, E. (2022). Teaching Programming Online: Design, Facilitation and Assessment Strategies and Recommendations for High School Teachers. *TechTrends*, *66*. https://doi.org/10.1007/s11528-022-00724-x

Shareghi Najar, A., & Mitrovic, A. (2012). Using Examples in Intelligent Tutoring Systems. *Intelligent Tutoring Systems*, 579–581.

Sharmin, S., Zingaro, D., Zhang, L., & Brett, C. (2019). Impact of Open-Ended Assignments on Student Self-Efficacy in CS1, 215–221. https://doi.org/10.1145/3300115.3309532

Sharpnack, J., Hao, K., Mulcaire, P., Bicknell, K., LaFlair, G., Yancey, K., & von Davier, A. A. (2024). BanditCAT and AutoIRT: Machine Learning Approaches to Computerized Adaptive Testing and Item Calibration. https://arxiv.org/abs/2410.21033

Shi, Y., Chi, M., Barnes, T., & Price, T. (2022). Code-DKT: A Code-based Knowledge Tracing Model for Programming Tasks. https://arxiv.org/abs/2206.03545

Shute, V. J. (2008). Focus on Formative Feedback. *Review of Educational Research*, *78*(1), 153–189. https://doi.org/10.3102/0034654307313795

Silva, P., & Costa, E. (2025). Assessing Large Language Models for Automated Feedback Generation in Learning Programming Problem Solving. https://arxiv.org/abs/2503.14630

Simon-Campbell, E., & Phelan, J. (2018). Effectiveness of an Adaptive Quizzing System as a Self-Regulated Study Tool to Improve Nursing Students' Learning. *International Journal of Nursing & Clinical Practices*, *5*. https://doi.org/10.15344/2394-4978/2018/290

Singh, S. (2022). Identifying learning challenges faced by novice/beginner computer programming students: An action research approach. In H. Lichter, S. Aydin, T. Sunetnanta, T. Anwar, E. L. Ouh, B. Wadhwa, S. Chawla, K. Kumar, B. Suri, & B. Gan (Eds.), *Joint proceedings of the 10th international workshop on quantitative approaches to software quality (quasoq 2022) & the 6th software engineering education workshop (SEED 2022) co-located with 29th asia pacific software engineering conference 2022, virtual, december 6, 2022* (pp. 63–71, Vol. 3330). CEUR-WS.org. https://ceur-ws.org/Vol-3330/Paper-09-SEED.pdf

Snyder, H. (2019). Literature review as a research methodology: An overview and guidelines. *Journal of Business Research*, *104*, 333–339. https://doi.org/https://doi.org/10.1016/j.jbusres.2019.07.039

Sobral, S. (2021). Bloom's Taxonomy to Improve Teaching-Learning in Introduction to Programming. *International Journal of Information and Education Technology*, *11*, 148–153. https://doi.org/10.18178/ijiet.2021.11.3.1504

Stanja, J., Gritz, W., Krugel, J., Hoppe, A., & Dannemann, S. (2023). Formative assessment strategies for students' conceptions—the potential of learning analytics. *British Journal of Educational Technology*, *54*(1), 58–75. https://doi.org/10.1111/bjet.13288

Stefanutti, L., & de Chiusole, D. (2017). On the assessment of learning in competence based knowledge space theory. *Journal of Mathematical Psychology*, *80*, 22–32. https://doi.org/https://doi.org/10.1016/j.jmp.2017.08.003

Steiner, C. M., Nussbaumer, A., & Albert, D. (2009). Supporting Self-Regulated Personalised Learning through Competence-Based Knowledge Space Theory. *Policy Futures in Education*, *7*(6), 645–661. https://doi.org/10.2304/pfie.2009.7.6.645

Steinmayr, R., Weidinger, A. F., Schwinger, M., & Spinath, B. (2019). The Importance of Students' Motivation for Their Academic Achievement – Replicating and Extending Previous Findings. *Frontiers in Psychology*, *10*. https://doi.org/10.3389/fpsyg.2019.01730

Strickroth, S. (2024). Exploring Students' Self-Confidence in Their Programming Solutions. *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, 415–421. https://doi.org/10.1145/3649217.3653589

Strong, G., Higgins, C., Bresnihan, N., & Millwood, R. (2017, January). A Survey of the Prior Programming Experience of Undergraduate Computing and Engineering Students in Ireland. https://doi.org/10.1007/978-3-319-74310-3_48

Sullivan, G. M., & Artino, A. R., Jr. (2013). Analyzing and interpreting data from likert-type scales. *J Grad Med Educ*, *5*(4), 541–542.

Sun, Q., Wu, J., Rong, W., & Liu, W. (2019). Formative assessment of programming language learning based on peer code review: Implementation and experience report. *Tsinghua Science and Technology*, *24*, 423–434. https://doi.org/10.26599/TST.2018.9010109

Szabó, T., Engelmann, B., & Birken, K. (2017). An Overview of Program Analysis using Formal Methods With a Particular Focus on their Relevance for DSLs. https://api.semanticscholar.org/CorpusID:52063125

Taherdoost, H. (2022). What are Different Research Approaches? Comprehensive Review of Qualitative, Quantitative, and Mixed Method Research, Their Applications, Types, and Limitations. *Journal of Management Science & Engineering Research*, *5*. https://doi.org/10.30564/jmser.v5i1.4538

Thompson, E., Luxton-Reilly, A., Whalley, J. L., Hu, M., & Robbins, P. (2008). Bloom's taxonomy for CS assessment. *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78*, 155–161.

Tisza, G., Markopoulos, P., & Bekker, T. (2023). Learning to code: Interplay of attitude, emotions, and fun. *Humanities and Social Sciences Communications*, *10*. https://doi.org/10.1057/s41599-023-02235-3

Toti, G., & Chen, G. (2024). Teaching CS1 with a mastery learning framework: Changes in CS2 results and students' satisfaction. *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 1*, 221–227. https://doi.org/10.1145/3649165.3690105

Toti, G., Chen, G., & Gonzalez, S. (2023). Teaching CS1 with a mastery learning framework: Impact on students' learning and engagement, 540–546. https://doi.org/10.1145/3587102.3588844

Trumbull, E., & Lash, A. A. (2013). Understanding Formative Assessment Insights from Learning Theory and Measurement Theory.

Ullah, Z., Lajis, A., Jamjoom, M., Altalhi, A., Al-Ghamdi, A., & Saleem, F. (2018). The effect of automatic assessment on novice programming: Strengths and limitations of existing systems. *Computer Applications in Engineering Education*, *26*. https://doi.org/10.1002/cae.21974

Van der Kleij, F. M., Eggen, T. J., Timmers, C. F., & Veldkamp, B. P. (2012). Effects of feedback in a computer-based assessment for learning. *Computers & Education*, *58*(1), 263–272. https://doi.org/https://doi.org/10.1016/j.compedu.2011.07.020

Van der Kleij, F. S., & Adie, L. (2018). Formative assessment and feedback using information technology (G. K. Voogt, R. Christensen, & K. Lai, Eds.). *Second Handbook of Information Technology in Primary and Secondary*, 601–615. https://doi.org/https://doi.org/10.1007/978-3-319-71054-9

Velázquez-Iturbide, J. Á. (2021). An Analysis of the Formal Properties of Bloom's Taxonomy and Its Implications for Computing Education, 1–7. https://doi.org/10.1145/3488042.3488069

Velde, M., Sense, F., Borst, J., & Rijn, H. (2021). Alleviating the Cold Start Problem in Adaptive Learning using Data-Driven Difficulty Estimates. *Computational Brain & Behavior*, *4*. https://doi.org/10.1007/s42113-021-00101-6

Verkleij, M. (2019). *Teaching programming using industry tools* [B.S. thesis]. University of Twente.

Vesin, B., Mangaroska, K., Akhuseyinoglu, K., & Giannakos, M. (2022). Adaptive Assessment and Content Recommendation in Online Programming Courses: On the Use of Elo-rating. *ACM Trans. Comput. Educ.*, *22*(3). https://doi.org/10.1145/3511886

Vie, J.-J., Popineau, F., Bruillard, É., & Bourda, Y. (2017). A Review of Recent Advances in Adaptive Assessment. https://doi.org/10.1007/978-3-319-52977-6_4

Vorobyov, K., & Krishnan, P. (2010). Comparing Model Checking and Static Program Analysis: A Case Study in Error Detection Approaches.

Wang, J., Lin, P., Tang, Z., & Chen, S. (2023). How problem difficulty and order influence programming education outcomes in online judge systems. *Heliyon*, *9*(11). https://doi.org/10.1016/j.heliyon.2023.e20947

Wang, X.-M., Hwang, G.-J., Liang, Z.-Y., & Wang, H.-Y. (2017). Enhancing students' computer programming performances, critical thinking awareness and attitudes towards programming: An online peer assessment attempt. *Educational Technology and Society*, *20*, 58–68.

Wang, Y., Li, H., Feng, Y., Jiang, Y., & Liu, Y. (2012). Assessment of Programming Language Learning Based on Peer Code Review Model: Implementation and Experience Report. *Computers & Education*, *59*, 412–422. https://doi.org/10.1016/j.compedu.2012.01.007

Williams, L., Mccrickard, D., Layman, L., & Hussein, K. (2008). Agile 2008 Conference Eleven Guidelines for Implementing Pair Programming in the Classroom, 445–452. https://doi.org/10.1109/Agile.2008.12

Xinogalos, S., Pitner, T., Savić, M., & Ivanović, M. (2019). First Programming Language in Introductory Programming Courses, Role of. In A. Tatnall (Ed.), *Encyclopedia of education and information technologies* (pp. 1–11). Springer International Publishing. https://doi.org/10.1007/978-3-319-60013-0_217-1

Yan, H. (2020). Using Learning Analytics and Adaptive Formative Assessment to Support At-risk Students in Self-paced Online Learning. *2020 IEEE 20th International Conference on Advanced Learning Technologies (ICALT)*, 396–398. https://doi.org/10.1109/ICALT49669.2020.00125

Yan, Z., Panadero, E., Xiang, W., & Zhan, Y. (2023). A Systematic Review on Students' Perceptions of Self-Assessment: Usefulness and Factors Influencing Implementation. *Educational Psychology Review*, *35*, 81. https://doi.org/10.1007/s10648-023-09799-1

Yang, A. C., Flanagan, B., & Ogata, H. (2022). Adaptive formative assessment system based on computerized adaptive testing and the learning memory cycle for personalized learning. *Computers and Education: Artificial*

Intelligence, *3*, 100–104. https://doi.org/https://doi.org/10.1016/j.caeai.2022.100104

Yang, L., Sun, X., Li, H., Xu, R., & Wei, X. (2025). Difficulty aware programming knowledge tracing via large language models. *Scientific Reports*, *15*. https://doi.org/10.1038/s41598-025-96540-3

Yu, H., & Xie, Q. (2025). Generative AI vs. Teachers: Feedback Quality, Feedback Uptake, and Revision. *Language Teaching Research Quarterly*, *47*, 113–137. https://doi.org/10.32038/ltrq.2025.47.07

Zampirolli, F., Pisani, P., Borovina Josko, J., Kobayashi, G., Fraga, F., Goya, D., & Savegnago, H. (2020). Parameterized and automated assessment on an introductory programming course, 1573–1582. https://doi.org/10.5753/cbie.sbie.2020.1573

Zawacki-Richter, O., Marín, V., Bond, M., & Gouverneur, F. (2019). Systematic review of research on artificial intelligence applications in higher education -where are the educators? *International Journal of Educational Technology in Higher Education*, *16*, 1–27. https://doi.org/10.1186/s41239-019-0171-0

Zhai, X., Shi, L., & Nehm, R. (2021). A Meta-Analysis of Machine Learning- Based Science Assessments: Factors Impacting Machine-Human Score Agreements. *Journal of Science Education and Technology*, *30*. https://doi.org/10.1007/s10956-020-09875-z

Zheng, T. (2024). Dynamic difficulty adjustment using deep reinforcement learning: A review. *Applied and Computational Engineering*, *71*, 157–162. https://doi.org/10.54254/2755-2721/71/20241633

Zhou, Z., Wang, S., & Qian, Y. (2021). Learning From Errors: Exploring the Effectiveness of Enhanced Error Messages in Learning to Program. *Frontiers in Psychology*, *12*. https://doi.org/10.3389/fpsyg.2021.768962

Zohaib, M. (2018). Dynamic Difficulty Adjustment (DDA) in Computer Games: A Review. *Advances in Human-Computer Interaction*, *2018*, 1–12. https://doi.org/10.1155/2018/5681652

Appendix A

Plain statement and Sample quiz questions

A.1 Plain statement

Every quiz starts with a plain statement and some questions to track students' progress as follows:

Researchers' Names: Jagadeeswaran Thangaraj, Monica Ward, and Fiona O'Riordan

Project Title: Adaptive Formative Assessment Systems for Introductory Programming

Aim: By taking part in this quiz, you will learn more about the most frequent mistakes made when programming in Python.

Data Collection and Anonymity: Data from the survey will be collected anonymously through a Google Forms digital questionnaire. Data obtained from focus groups will be anonymised prior to analysis by the researchers, Jagadeeswaran Thangaraj, Monica Ward and Fiona O'Riordan. If you edit a Google Document used as part of the research process, your identity will not be revealed.

Data Storage and Confidentiality: All data collected will be securely stored within Dublin City University (DCU) applications using Google Drive and will

be accessible only to the researchers listed above. You will not be personally identified in any outputs resulting from this project. All data will be reported in an anonymised format, and no identifying information will be collected to ensure participant anonymity is maintained.

Consent: By proceeding with this study, you confirm that you have read and understood the information provided and consent to participate voluntarily. **Ethical Approval:** This study has received ethical approval from DCU's research ethics committee.

A.2 Sample Quiz Questions

A.2.1 Question-1 with Aligned Feedback

Question: What will the output be from the following code?

```
print "Hello world!"
```

Answer Choice	Corresponding Feedback
Hello world!	Incorrect. An opening parenthesis must follow the `print` keyword in Python.
SyntaxError	**Correct.** The `print` statement is missing parentheses, which results in a syntax error.
Hello world	Incorrect. While string literals are printed exactly as written, the syntax of the `print` statement is invalid.
`print "Hello world!"`	Incorrect. This option reproduces the source code rather than the program output.

Table A.1: Answer choices and adaptive feedback for Question-1

A.2.2 Question-2 with Aligned Feedback

Question: What will the output be from the following code?

```
x = 3

y = 4

average = x + y / 2

print(average)
```

Answer Choice	Corresponding Feedback
average	Incorrect! Here, average is supplied as a variable to print. It was the right response if it was enclosed in single or double quotations.
3.5	Incorrect! But there are no brackets around the expression. hence, the operator precedence varies.
5.0	**Correct**. Based on the operator precedence, this is the right outcome. `/, *` take precedence over `+, -`.
Nothing	Incorrect! The expression's outcome is printed. Try again when you have checked the expression.

Table A.2: Answer choices and adaptive feedback for Question-2

A.2.3 Question-3 with Aligned Feedback

Question: What will the output be from the following code?

```
lia_age = 18
if(lia_age > 18):
    print("Lia is eligible to vote")
else:
     print("Lia is not eligible to vote")
```

Answer Choice	Corresponding Feedback
Lia is eligible to vote	Incorrect! You are nearly there, however, it is effective for all values greater than 18. Consequently, all values starting at 19.
Lia is not eligible to vote	**Correct**. it is effective for all values starting at 19.
Syntax Error	Incorrect! It appears to be accurate. There is no error. Please double-check!.
None of the above	Incorrect! The code provided is valid. Based on the logical condition, it offers the outcome like whether the age greater than 18 or not.

Table A.3: Answer choices and adaptive feedback for Question-3

A.2.4 Question-4 with Aligned Feedback

Question: What will the output be from the following code?

```
number = 5
while (number > 0 ) :
    print(number)
    number=number-1
```

Answer Choice	Corresponding Feedback
4 3 2 1 0	Incorrect! You are nearly there. It prints 'number' prior to decrement. So the answer starts with 5 until it reaches 0.
5 4 3 2 1	**Correct**. It prints 'number' prior to decrement. So the answer starts with 5 until it reaches 0. So the answer is: 5 4 3 2 1.
Syntax Error	Incorrect! It appears to be accurate. There is no error. Please double-check!.
5	Incorrect! It prints 'number' prior to decrement. So the answer starts with 5 until it reaches 0.

Table A.4: Answer choices and adaptive feedback for Question-4

A.2.5 Question-5 with Aligned Feedback

Question: What is the error in the following function call?

```
def outer_fun(a, b):
    def inner_fun(c, d):
        return c + d
        return inner_fun(a, b)
    return a
result = outer_fun(5)
print(result)
```

Answer Choice	Corresponding Feedback
Invalid syntax error	Incorrect! Syntax are correct here. But the answer is 'missing the argument' in function call. Check the function call of $outer\_fun()$.
Missing argument error	**Correct**. Coder tries to invoke the function without passing the argument.
Indent error: (Wrong indents)	Incorrect! It follows proper indenting here. But the answer is 'missing the argument' in function call. Check the function call of $outer\_fun()$.
No error	Incorrect! There is an error. The answer is 'missing the argument' in function call. Check the function call of $outer\_fun()$.

Table A.5: Answer choices and adaptive feedback for Question-5