



Cognitive-based method and interactive environment for teaching-learning computer programming

AB KABO MBIADA

 **orcid.org/0000-0003-4614-1370**

Thesis accepted in fulfilment of the requirements for the degree Doctor of Philosophy in Computer and Information Science with Computer Science and Information Systems at the North-West University

Supervisor: Prof B Isong
Co-supervisor: Prof F Lugayizi
Prof A Abu- Mahfouz

Graduation: November 2024

DECLARATION

I, ALAIN BLAISE KABO MBIADA, declare that “**Cognitive-based method and interactive environment for teaching-learning computer programming**” is my work which has never been presented for the award of any degree at any university. All sources of information used in this research have been deservedly acknowledged in text and references.

Signature:



Date:

31/10/2024

APPROVAL

Supervisor:

Prof. Bassey Isong

Department of Computer Science

Faculty of Natural and Agricultural Science

North-West University

South Africa

Signature:



Date:

31/10/2024

DEDICATION

This work is foremost dedicated to the memory of Mbiada Fidèle and Mbiada Esther, my late parents. I have always had the strength and drive to go forward in life, and in this work specifically, because of what they taught me. From wherever you are, please find all of my love and thanks in this work.

ACKNOWLEDGEMENTS

First and foremost, I thank God for his divine support, as well as the strength and patience He gave me throughout this work.

I express my gratitude to my supervisor, Prof. B. Isong, for his invaluable guidance, assistance, and continuous motivation. Your insightful comments and guidance have enhanced this work and consistently assisted me in learning. You taught me a lot, and I pray that God continues to richly reward you and give you the strength you need to carry out your admirable job.

I express my sincere thanks to Mr. Thusoyaone Joseph Moemi and Dr. Eric Kameni for their cooperation with the experiments conducted for this study.

Many thanks to North-West University for providing me with the necessary resources and support to complete my research.

I sincerely thank Prof. M. Fouda Ndjodo for his pieces of advice and encouragement, without which I could not have completed the majority of my training.

I am truly grateful to Prof. C. Oukouomi for the opportunities he has always offered.

I am deeply thankful to my brothers and sisters, for their prayers, love, encouragement, and moral support.

I must especially acknowledge Carlos Tchawouo and Sandrine Tchawouo's unfailing love and support throughout the years. Without your support and prayers, this project would not have been possible. May God bless you all.

It is almost impossible to put into words how much I appreciate and thank you, my wife, Josiane, and the kids. You always understood how to lift my spirits and inspire me, and you were there for me when I was doubting myself. The outcome of your endurance and selflessness over the past few years of research in this work. Thank you for always being at my side, and may God bless you all.

I am also thankful to all my friends, Dr Ebode, Mr Roussel, Mr Youdom Merlin, Mr Dassi, Mr Ella, Mr Nsotou, Ms Guipo, Ms Tatchuek, M. Nouyep, etc.

ABSTRACT

Non-computing students and novices often struggle with motivation and understanding of programming language syntax in introductory programming modules. The current pedagogical strategies and instructional resources are primarily built on constructivist theories and are designed for computer science students. They often overlook computer programming syntax issues faced by non-computing students. Therefore, this study highlights the need for a cognitive teaching approach and tool tailored to non-computer science students. The study first conducted both online and face-to-face surveys, using questionnaires to elicit the computer programming challenges faced by first-year students. Then, to increase the motivation and engagement of non-computing students, this study leverages the design science research approach to propose an explicit instructional methodology for teaching introductory programming. The study also introduces PyLe, an interactive learning environment for Python programming modules, to solve programming syntax issues. Analysis of the data collected from the first-year students showed that both computer and non-computer science students face several programming difficulties ranging from recursion, arrays, error handling, and functions/procedures or methods. Evaluating the effectiveness of the novel teaching method, a post-experiment data analysis showed positive outcomes for students' motivation, including attendance, relevance, confidence, and satisfaction. Moreover, testing PyLe at two universities improved students' understanding of programming syntax and maintained high usability. The study, therefore, recommends the new method and PyLe for teaching programming to non-computer students and novices. This study improves the teaching and learning of computer programming, particularly for non-computing and novice students, by introducing specialized methodologies and tools to tackle their unique challenges.

Key terms: Computer programming, Teaching methodology, Students' motivation, Explicit instruction, Programming syntax, PyLe, Teaching/learning environments.

LIST OF ABBREVIATIONS

ADDIE	Analysis, Design, Development, Implementation, and Evaluation
ADRI	Approach, Deployment, Result, and Improvement
ASSURE	Analyzing learners, Stating objectives, Selecting methods or strategies, technology, and materials, Using the available resources, Requiring learner participation, Evaluating and revising
CBLS	Cloud-Based Learning System
CI	Concept Inventory
CS	Computer Science
CSE	Computer Science Education
DSDM	Dynamic Systems Development Method
DSR	Design Science Research
EI	Explicit Instruction
ES	Educational Software
EDS	Educational Software Design
FNAS	Faculty of Natural and Agricultural Sciences
HTTC	Higher Teacher Training College
IIADPIE	Initial, Instructional Orientation, Analysis, Design, Production, Integration and Implementation, and Evaluation
ID	Instructional Design
IDM	Instructional Design Methods
IP	Introductory Programming
NWU	North-West University
UDS	University of Dschang
UY 1	University of Yaoundé 1

TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	1
1.1	Chapter Outline.....	1
1.2	Introduction.....	1
1.3	Background Information	2
1.3.1	Students' Computer Programming Challenges	2
1.3.2	Misconception and Concept Inventories in IP	4
1.3.3	Motivation and Engagement in Computing Programming	5
1.3.4	Learning Theory	6
1.3.5	Explicit Instruction Method	7
1.4	Problem Statement	8
1.5	Research Motivation.....	9
1.6	Research Questions	9
1.7	Research Aim.....	10
1.8	Research Objectives	10
1.9	Research Significance.....	11
1.10	Research Contributions	12
1.11	Proposed Approach and Solution	12
1.12	Research Scope and Limitation.....	13
1.13	Structure of the Thesis.....	13
1.14	Publications From this Thesis	15
1.15	Chapter Summary.....	16

CHAPTER 2	LITERATURE REVIEW	17
2.1	Chapter Outline.....	17
2.2	Introduction.....	17
2.3	State-of-the-Practice in IP Learning and Instructional Techniques and Resources	17
2.3.1	Related Studies	18
2.3.2	Deliver-based Approaches.....	19
2.3.3	Content-based Approaches	23
2.3.4	Tool-based Approaches.....	24
2.3.5	Assessment-based Approaches	30
2.3.6	Discussion	30
2.4	Agile Methods	32
2.5	Instructional Design Models	35
2.6	Chapter Summary.....	37
CHAPTER 3	RESEARCH METHODOLOGY AND DESIGN	39
3.1	Chapter Outline.....	39
3.2	Introduction.....	39
3.3	Research Methodology	41
3.4	Research Design.....	43
3.5	Chapter Summary.....	49
CHAPTER 4	FRAMEWORK DEVELOPMENT.....	50
4.1	Chapter Outline.....	50

4.2	Introduction.....	50
4.3	The Proposed Explicit Instruction-based Framework for Teaching IP	51
4.4	The Proposed Framework for Efficient Educational Software Development	54
4.4.1	Justification.....	54
4.4.2	IIADPIE Process.....	55
4.5	PyLe: The Proposed Interactive Learning Environment	66
4.5.1	Design Goals.....	67
4.5.2	PyLe Infrastructure	68
4.5.3	User Requirements.....	71
4.5.4	Class Diagram	72
4.5.5	Student Profile in PyLe	74
4.5.6	Resources in PyLe.....	75
4.5.7	Activities in PyLe.....	77
4.6	Analysis.....	78
4.7	Chapter Summary.....	84
CHAPTER 5	RESULTS AND ANALYSIS	85
5.1	Chapter Outline.....	85
5.2	Introduction.....	85
5.3	Phase 1: IP Challenges for First-Year Students in CS and Non-CS Fields	8
5.4	Phase 2: IP-EI Framework Evaluation	86
5.5	Phase 3: PyLe’s Evaluations.....	87
5.6	Results and Analysis.....	89

5.6.1	Phase 1's Results and Analysis	89
5.6.2	Phase's 2 Results and Analysis	96
5.6.3	Phase's 3 Results and Analysis	100
5.7	Discussions	104
5.7.1	Phase 1 Evaluation.....	105
5.7.2	Phase 2 Evaluation.....	105
5.7.3	Phase 3 Evaluation.....	107
5.8	Validity Threats	108
5.8.1	Phase 1	109
5.8.2	Phase 2	109
5.8.3	Phase 3	109
5.9	Chapter Summary.....	110
CHAPTER 6	SUMMARY, CONCLUSION, AND RECOMMENDATIONS	111
6.1	Summary	111
6.2	Conclusion.....	113
6.3	Recommendations.....	115
References		117

LIST OF TABLES

Table 2.1: Comparing Scrum with other Agile Methods [145]	34
Table 4.1: Student Profile in PyLe [21]	74
Table 4.2: A Suggestion for the order of PyLe's Lessons [21].....	75
Table 4.3: Comparison of IPEI with some IP Teaching Methods	80
Table 4.4: IIADPIE Analysis [81].....	81
Table 4.5: Theoretical Comparison of PyLe with Some Existing Learning Programming Environments [21]	83
Table 5.1: Mann-Whitney U Test Statistic [20]	96
Table 5.2: Outcomes of the Descriptive Analysis of Participants' Motivation	96
Table 5.3: Correlation between ARCS dimensions	100
Table 5.4: Descriptive Analysis of Grade and TT [21]	101
Table 5.5: Descriptive Analysis Result of PyLe's Usability [21]	103

LIST OF FIGURES

Figure 1.1: Predicted Anatomy of a Concept Inventory	5
Figure 1.2: Mapping Research Questions to Research Objectives	11
Figure 1.3: Structure of the Thesis.....	15
Figure 2.1: OGITS Main Page [116]	25
Figure 2.2: Sample View of Python Grid System [124]	28
Figure 2.3: Summary of the Learning and Instructional Approaches and Materials for IP Modules [42].....	32
Figure 2.4: Agile Method's Values and Principles [141]	33
Figure 2.5: DSDM Components and Process Diagram [146]	35
Figure 2.6: Component of ADDIE's Model [149]	37
Figure 3.1: Overview of the Research Process.....	40
Figure 3.2: Research Design of the Study	49
Figure 4.1: IPEI Framework.....	52
Figure 4.2: IIADPIE Process [81]	56
Figure 4.3: Initial Phase of IIADPIE's Process.	57
Figure 4.4: Instructional and Orientation Phase of IIADPIE's Process.	58
Figure 4.5: Analysis Phase of IIADPIE's Process.	61
Figure 4.6: Design Phase of IIADPIE's Process.	64
Figure 4.7: Production Phase of IIADPIE's Process.	65
Figure 4.8: Integration and Implementation Phase of IIADPIE's Process	66
Figure 4.9: PyLe Architecture [21]	69

Figure 4.10: Sign-in Sequence Diagram	70
Figure 4.11: Learner Use Cases Diagram	71
Figure 4.12: Class Diagram	73
Figure 4.13: PyLe Lesson Overview [21]	76
Figure 4.14: Sample of Interactive Visualization [21]	76
Figure 4.15: Overview of Activities [21].....	78
Figure 5.1: Experiments and Assessment Workflow	88
Figure 5.2: Repartition of NWU Respondents by Field of Study [20]	90
Figure 5.3: Students' Background Features [20]	91
Figure 5.4: Students' Preferred Situational Learning [20].....	92
Figure 5.5: Students' Preferred Learning Materials [20].....	93
Figure 5.6: Students' General Opinion of Computing Programming [20]	94
Figure 5.7: Students' Proficiency in some Key Concepts of Programming [20]	95
Figure 5.8: Selecting frequency for certain programming concepts [20].....	95
Figure 5.9: Attention	98
Figure 5.10: Relevance	98
Figure 5.11: Confidence	99
Figure 5.12: Satisfaction	100
Figure 5.13: Post-test Results for Cameroon [21]	101
Figure 5.14: Post-test Results for South Africa [21]	101
Figure 5.15: Independent-samples Mann-Whitney U Test on GRADE [21].....	102
Figure 5.16: Independent-samples Mann-Whitney U Test on TT [21]	103

Figure 5.17: ARCS-model Summary Results.....	107
Figure 5.18: Overview of a Student Progress Report [21]	108
Figure 5.19: Student's Activities Report [21]	108
Figure 6.1: Mapping Research Questions to Research Objectives, and Chapters	113

CHAPTER 1 INTRODUCTION

Chapter Outline

This chapter presents an overview of the research and some background information, detailing the research statement, motivation, questions, aims, objectives, significance, contribution, proposed approach and solution, research scope, thesis structure, and publications.

Introduction

The study undertaken in this research aims to improve the quality of teaching and learning computer science by questioning the quality of teaching practices and by finding the best way to facilitate the student's construction of knowledge. Fincher and Petre had identified ten research areas in Computer Science Education (CSE) [1]: *"student understanding, visualization systems, teaching methods, assessment, educational technology, transfer of professional practice into the classroom, the incorporation of new development and new technologies into the classroom, e-learning, recruitment and retention of students, and the construction of the discipline"*. In terms of the identified research areas, student understanding, for instance, relates to students' conceptual and mental models, insights, and misconceptions. This is to say that the research is on *"Learning to Design Software as well as Misconceptions and Attitudes that Interfere with Learning to Program"* [1]. Teaching methods as a research area, on the other hand, focuses on the structuring of teaching contents in a way to produces a positive and improves students' learning outcomes. Furthermore, educational technology refers to the application of technology (instruments, programmes and techniques) in education to improve instruction, learning and administrative practices. Previous works have shown the benefit of emerging devices and technologies in education [2], [3].

Nowadays, the diverse use of technology has led education officials to integrate CS courses and technology into the curricula of most university programmes. As a result, CS is open to all students regardless of their qualifications, which has increased the number of students learning CS. Hence, providing fundamental computing skills to students to ease their integration in this digital era. Among the modules offered in CS, algorithms, data structures and programming are becoming global skills, particularly as the computer literacy movement gains momentum [4]. However, learning computer programming is challenging for most beginners at the university level [5], as the tedious process involved in learning makes it unattractive [6]. Learning computer

programming requires a diversity of logical tasks and mental representations linked to designing, understanding, modifying, bug-fixing and documenting [7]–[9]. According to the actual programming teaching context, some learners who attend classes without a scientific background are more familiar with the analysis, discussion and interpretation aspects but less likely to be interested in topics such as programming, algorithms and data structures [10]. The divergent backgrounds of students in introductory programming courses (CS1) variably impede students' performance and motivation [11]. To alleviate this problem, several teaching methods and tools such as gamification, problem-based learning, inductive teaching methods, pair programming, visual programming tools, etc. have been proposed [12]. Despite the promising results of these methods and tools, there is still no explanation as to why not all students complete their computer programming courses [13]. This is compounded by the fact that students show little interest in programming courses after they complete their modules [14]–[17]. Furthermore, most students remain at a reproductive level of education even after completing the course [16]. This might explain why a substantial number of students fail to master the basic abilities of IP [17]. Even worse, many students who complete such courses possess minimal confidence in their programming skills [14]. They often struggle with developing a simple application or even interpreting small fragments of code [15]. In light of the preceding findings, it becomes imperative to build innovative approaches to teaching and learning IP to improve students' basic skills in this area [5], [7], [12], [18]. The study emphasizes that the existing CS teaching methods and learning tools are limited to CS students. They, therefore, do not cater for slow-paced and non-computing students who usually experience more difficulties in IP modules than CS students, as they are typically both beginners with an interest in another field [19], [20]. It is in this dynamic that this study fills the gap and proposes a novel teaching method and a fully interactive learning environment for introductory computer programming for non-computer science students.

Background Information

This section discusses some relevant concepts such as computing programming challenges, students' misconceptions, motivation and engagement, learning theory, and explicit instruction methods. Some of these concepts contribute to understanding the frameworks designed in this study.

1.1.1 Students' Computer Programming Challenges

We should acknowledge that this subsection was published [20], [21]. The process of writing executable, computer-readable instructions to solve a problem is known as computer

programming. It includes a series of deftly, rational, and critical thinking exercises that span from program maintenance to analysis and design [8]. Students can create instructions by utilizing text editors, Integrated Development Environments (IDEs), and other tools, along with programming languages like Java, C, C++, Python, PHP, JavaScript, etc. With its many applications and crucial role in the development of technology, computer programming has become an essential ability for students studying computers and related fields. It enables them to design innovative technologies and develop software. Competence in computer programming is now essential due to the rising need for technology in many sectors. The duties are complex and necessitate logical reasoning, problem-solving skills, and meticulous attention to detail. Hence, acquiring proficiency in programming logic can prove to be arduous and often exasperate novice students, resulting in diminished learning drive, academic setbacks, and eventual withdrawal. The challenges in understanding IP modules may arise from a variety of sources [8], [22]–[24]. These include inappropriate teaching methods, disregarding the prior knowledge of beginners, using expert-level content explanations, subject matter expertise of the instructor, learners being preoccupied with other classes, students who are accustomed to memorization rather than comprehension, large class sizes, challenges with using line-by-line approaches to tackle programming problems, misunderstandings between programming syntax and native languages, modules that are too short, and a dearth of beginner-friendly computer mental models [25]. Even though learning programming can be challenging, it is doable, and developing and putting into practice effective teaching and learning methodologies for programming requires an understanding of the root causes of the difficulty. Some of these difficulties include program construction, error correction, recursion, arrays, pointers, error handling, poor library usage, poor mental model representation, unreferenced classes, issues related to class attributes and cohesion, and more [26]–[30].

On the other hand, the programmer must adhere to specific rules and semantics specific to the used programming language to solve a problem appropriately. The rules are a reference to the syntax of the programming language, which includes the standards for punctuation, symbols, and each word. Beginners always begin their programming endeavours with syntactical errors, and after some time has passed, semantical errors become an increasingly prevalent element of all of their programming time [31]. Learning to program is an activity that is full of mistakes. Therefore, basic programming modules often contain syntax mistakes, one of the most common types of errors. These errors are responsible for student frustration, high failure rates, and the drop-out of modules [32]. Non-computing students struggle far more than their computer science counterparts [20]. Students who spend a significant amount of time striving to correct particular mistakes may greatly benefit from receiving specialized guidance.

1.1.2 Misconception and Concept Inventories in IP

Misconceptions and concept inventories (CI) are critical to providing the right interventions to improve the learning process. Misconceptions are academic concepts that are challenging and well-rooted [33]. Learners most often build huge misconceptions when it comes to programming. In learning programming, students' misconceptions involve syntax errors, improper semantic comprehension, and other challenges in the attempt to write correct programs [34]. Moreover, the age of the student, the programming language, the context of programming, the challenges of switching from a native language to a programming language, a textbook that includes wrong information and mistakes, previous learning experiences, and insufficient insight into the topic by the teacher are some factors that can develop or prevent students' misconceptions [32], [35]. Previous studies developed some misconceptions [35]–[38] that could help teachers develop their understanding (or lists or data banks) of their students' misconceptions. For example, Altadmri and Brown [39] in their study identified the 18 most common mistakes made by a wide range of beginners worldwide in the Java programming language and classified them into three main types: misunderstanding or forgetting the syntax, type errors, and semantic errors. In addition, Johnson et al. [36] identified seven first-year students' misconceptions of Python programming languages. Developing misconceptions is an important skill for teachers. To this end, this study strongly recommends that teaching strategies for IP courses that include students' misconceptions of the content, is an important measure to reduce students' frustration, failure and dropout rates. It might be also helpful in designing an effective learning programming environment.

On the other hand, CI is a specialized multiple-choice test designed to assess students' understanding of the basic notions of a subject [40]. In this case, teachers and researchers are in accord that if students have a good understanding of the basic notions, they should certainly understand all of the rest of the concepts in the subject [41]. The CIs are not instructor assessments but are supplementary to the final exam. They reveal students' misconceptions and help the instructor to identify struggling students' fundamental concepts and build his instruction accordingly. The multi-choice item is a test constituted of distractors and correct choice options as shown in Figure 1.1 [42]. Distractors reflect students' misconceptions and come from analysis of results from the exercises during lectures as well as tutorial classes, or exams [42], [43]. Thus, we recommend the following process designed by Taylor et al. [40], for developing and validating reliable CIs: a) Identify core concepts; b) Identify student thinking; c) Create multi-choice survey

questions; d) Create forced-answer tests; e) Validate test questions through interviews; f) Administer and statistically analysis.

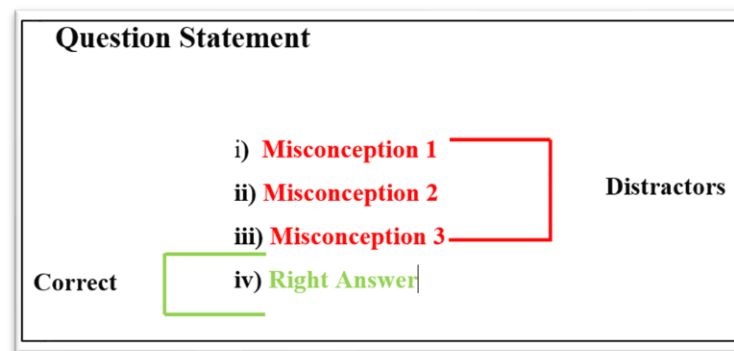


Figure 1.1: Predicted Anatomy of a Concept Inventory

1.1.3 Motivation and Engagement in Computing Programming

Motivation and engagement are important success factors in IP courses [44], and how to motivate and engage students is a major concern in all disciplines. A success factor represents certain skills, attitudes, or beliefs that, once a student has them, lead to the conclusion that he or she will have a good outcome in learning activities [45]. It is important for a teaching strategy to incorporate or include at least one success factor. When students understand their potential and understand that gaining these skills will lead to a good grade and the privileges that a good grade confers, they become motivated to learn or gain new skills [46]. To support motivation, teaching methods need to integrate techniques that deal with the attention, relevance, confidence, and satisfaction (ARCS) model [47]. The ARCS model is therefore an excellent way of measuring students' motivation. The ARCS model was developed by John Keller [48] as a result of a concern to provide more accurate approaches to the key influences on motivation to learn as well as systematic means of detecting and addressing problems associated with motivation to learn.

Furthermore, engagement is viewed as the ability to invest time and effort throughout the training program [49] and involves the initiation of action and active participation. It is defined as “a positive, fulfilling, work-related state of mind that is characterized by vigour, dedication, and absorption...” [50]. Vigor is described as high energy levels and mental toughness during work, a desire to work hard, and perseverance even in case of difficulties. Dedication is about being deeply immersed in one's task and feeling a sense of significance, enjoyment, pride, and challenge. Absorption is being completely focused and joyfully absorbed in one's task, so time flies by and it is hard to detach oneself from one's task. To support students' engagement in

learning, the teaching approach must include opportunities for control over the activities offered to develop a positive perception of competence; including tasks that correspond to the students' level of ability; provide self-assessment grids to allow students to assess their performance and thus foster the development of their autonomy. Schaufeli et al. [50] designed an assessment questionnaire known as the Utrecht Work Engagement Scale (UWES) for the measurement of engagement.

On the other hand, in teaching and learning IP modules, in addition to motivation and engagement, some studies pointed out other success factors such as grades from secondary school, prior programming experience, gender, mathematics or English background, learning style, self-efficacy, mental model, comfort level, achievement goals, teaching tools and methods, etc. [9], [51], [52]. It is important to notice that each teaching strategy should support or develop at least one of these factors in learners.

1.1.4 Learning Theory

The pedagogy literature presents numerous ideas and models, among them behaviourism, cognitivism, and constructivism. These theories and models promote or encourage various teaching and learning techniques. Behaviourism, a theory of learning, relies on observable behaviours at the end of the learning process to specify the information learners need to acquire through these behaviours [53]. By focusing on inputs and outputs rather than the processes themselves, behaviourists feel that mental structures are similar to a black box to which we do not have access. As a result, they believe that it is more realistic and effective to concentrate on the processes themselves. Critics argued that behaviourism solely relied on lower-level mental processes. Nevertheless, similar procedures are necessary in the majority of areas. Coding, for example, requires a significant amount of experience in applying a variety of programming ideas regularly. Cognitivism is a theory of learning that places a greater emphasis on mental processes. As a result, it is significantly more concerned with how we learn, and how our minds take in, store, process, and retrieve information [54]. This perspective, superior to behaviourism, emphasizes the learner's mental process of information acquisition [53]: Learning is a cognitive process that entails the organization of information into conceptual models. We need to structure, order, and deliver instructions in a way that is comprehensible and significant to the learner to facilitate effective learning. Retention and recall are crucial for the creation of neural patterns; organizing learning materials enhances memory; and educators must provide tools that aid the brain in information processing. When compared to behaviourism, this perspective is superior because it places more emphasis on the mental process of information acquisition being carried out by the

learner [53]. The theory of learning, known as constructivism, asserts that students learn most effectively when they actively construct their knowledge while in school. Assuming that students are capable of improving their grasp of a topic and applying it to new contexts, it is founded on the concept that students are active learners. Therefore, it is the instructor's responsibility to foster learning by assisting students in learning how to learn.

1.1.5 Explicit Instruction Method

Explicit instruction (EI) is a teacher-centred method, unlike less structured models which are student-centred and based on the pedagogy of discovery. This method emanates from the cognitivist that focuses on the development of intelligence and cognitive processes such as perception, memory, and thinking, as well as the resulting behaviours [54]. Rosenshine defined EI as *"...a systematic method of teaching with an emphasis on proceeding in small steps, checking for student understanding, and achieving active and successful participation by all students"* [55]. It is called explicit due to its actions for designing and implementing instruction. With this method, teachers intentionally use a variety of strategies to support students' learning. They guide students throughout the learning process with detailed explanations, demonstrations, and supported practices with feedback [56]. Its success is due to students' deployed efforts and the teaching strategies used by teachers. On the other hand, a strategy can be defined as a complex cognitive or meta-cognitive operation that allows one to reach a given goal by employing a series of actions carried out consistently or inconsistently [57]. The strategy will serve to carry out cognitive processes such as comprehension and memorization. It is an effective instructional approach to promote the success of the greatest number and is recommended for struggling, slow-paced weak, and good students as well [58], [59]. EI focuses on teaching critical content, and we assume that the implementation of EI in IP courses can help overcome most of the difficulties encountered by novices as non-computing students to enhance their motivation, mental model construct, commitment, learning strategies, outcomes, etc. That is, offer clear and direct teaching techniques, providing explicit and straightforward guidance to novice learners. However, implementing EI in a teaching and learning situation can be difficult. Therefore, for quality explicit teaching, we recommend the use of the ten guidelines suggested by Ben Newmark detailed here [60]. These principles do not constitute a definitive set of rules, but rather a contribution of advice to teachers on how to achieve high-quality explicit teaching. Its key features include clear learning objectives, teacher modelling, guide practice, gradual release of responsibility, structured feedback, repetition and review, monitoring progress and versatile application [54], [55].

Problem Statement

Various instructional tools and teaching methods have emerged in recent decades to help beginners learn computer programming and overcome the challenges of IP modules. However, the application or implementation of these tools has not yielded the expected results due to their ineffectiveness, time constraints, and lack of integration into existing teaching material [2]. Most of these tools emphasise more on learning to code rather than learning programming, which involves many steps where coding is the last phase; thus, targeting a specific experienced audience. These tools are limited, they statically present contents, and are silent about the learning and teaching theory implemented. They are also silent about their application to mitigate the problems of syntax faced by novices. It will be challenging for a student, especially non-computing students to use fundamental programming concepts that they are unfamiliar with to solve a particular problem. For those who are new to IP modules, syntax mistakes are typically a major source of annoyance, and discouragement, which often results in them dropping out [61]. Furthermore, the bulk of interventions based on constructivism were directed more at CS students than non-computing students. Thus, their application cannot be generalized to all students attending computing programming modules. Consequently, most beginners are not fully engaged in the learning process [6]. To address the above-mentioned drawbacks, further research is needed to develop new tools and teaching approaches, especially those devoted to non-computing students [6], [22], [62]. To provide such inclusive and effective tools and methods, this study seeks to develop a cognitive-based new teaching approach for introductory programming modules that promotes motivation as well as an interactive learning environment for IP courses, especially for non-computing students. The novel teaching approach prioritizes the development of students' programming memory, equipping them with a wealth of strategies, examples, and non-examples while also addressing common misconceptions. The learning environment aims to enhance learning and mastery of programming language syntax. This environment intends to include, at the same place content, interactive daily-life-based exercises with automated assessment, and the visualization of dynamic basic programming processes. The content is dynamic and based on the cognitivist theory. As a result, the environment will be easily integrated into the teaching material for the target modules from which the target population is taken. We assume that these developments will significantly boost students' motivation and performance, potentially attracting students outside of the current targeted population to programming courses. Furthermore, these developments will provide lecturers with an additional teaching technique, and developers will have the chance to create more effective and efficient educational software that can seamlessly integrate into teaching materials.

Research Motivation

Computer programming is a combination of cognitive skills required to solve a specific problem [8]. It allows students to build software that expands their personal and professional effectiveness [63]. However, given its abstract character, computer programming is difficult to absorb and grasp [8]. These challenges have led to a significant number of students failing to complete their degree, exacerbated by a remarkable increase in the percentage of dropouts, a general concern in computing education [18], [63]–[66]. Nikula *et al.* [65] reported that up to 30% of students worldwide in 1999 dropped or failed the introductory computer programming course; another study reported a failure rate of 28% of students was reported in [18] while Alturki [64] and Margulieux *et al.* [63] also reported that more than 50% of students withdraw or fail computer programming courses in Saudi Arabia and the United States. Various factors contribute to the dropout and failure rate, including the demotivation of students, their language skills, the absence of efficient resources and infrastructure, syntax errors, the size of the classes, the effectiveness of the teaching methodology, the immaturity of students, their prior programming knowledge, and the predominance of mathematics examples in the teaching materials, among others [22], [62], [67], [68]. Recognizing that, compared to their peers in computer science, non-computing computing students in overcoming some of these challenges and to equip instructors with a new teaching methodology for teaching IP modules to them. This study will assist in reducing the failure and dropout rates reported above among non-computing students.

Research Questions

To address the above-stated problem statement, the following research questions (RQs) shall be answered in this study:

RQ1: What are the state-of-the-art methods and tools for teaching-learning programming?

(Consult Chapter 2 for the response to this research question)

RQ2: How can we develop a teaching approach to motivate non-CS students to study computer introductory programming?

To answer RQ2, the following sub-questions will be answered:

RQ2.1: What are the features and challenges faced by the targeted population when learning computer programming? (Consult Chapter 5 for further details)

RQ2.2: What teaching method can be developed to suitably satisfy the concerned population when learning computer programming? (Consult Chapter 4 for further details)

RQ2.3: How can we design an effective interactive environment that can help students in their learning process? (Consult Chapter 4 for further details)

RQ3: How can we implement this environment?

To answer RQ3, the following sub-questions will be answered:

RQ3.1: How can we implement and evaluate the performance of the proposed teaching method and the interactive environment? (Refer to Chapter 5 for the answer to this RQ)

RQ3.2: How can we appreciate the dynamic of student progression in learning computer programming? (Refer to Chapter 5 for the answer to this RQ)

Research Aim

This research is aimed at designing and implementing a cognitive-based method and an interactive environment for introductory computer programming to assist non-computing students.

Research Objectives

To meet the main aim and address the aforementioned research questions, the following research objectives (ROs) will be performed:

RO1: Conduct a comprehensive literature review on the methods and tools for teaching and learning computer programming.

RO2: Conduct a survey to identify challenges and main features of the target population, to develop a cognitive-based teaching approach, and elaborate a unified learning system process building that couples a software engineering technique and a learning system design method to ease the conception of the proposed interactive learning environment devoted to the target population.

RO3: Implement and experiment with the interactive environment and the proposed teaching method, elaborating a questionnaire and conducting a survey to assess their effectiveness on students' motivation and performance.

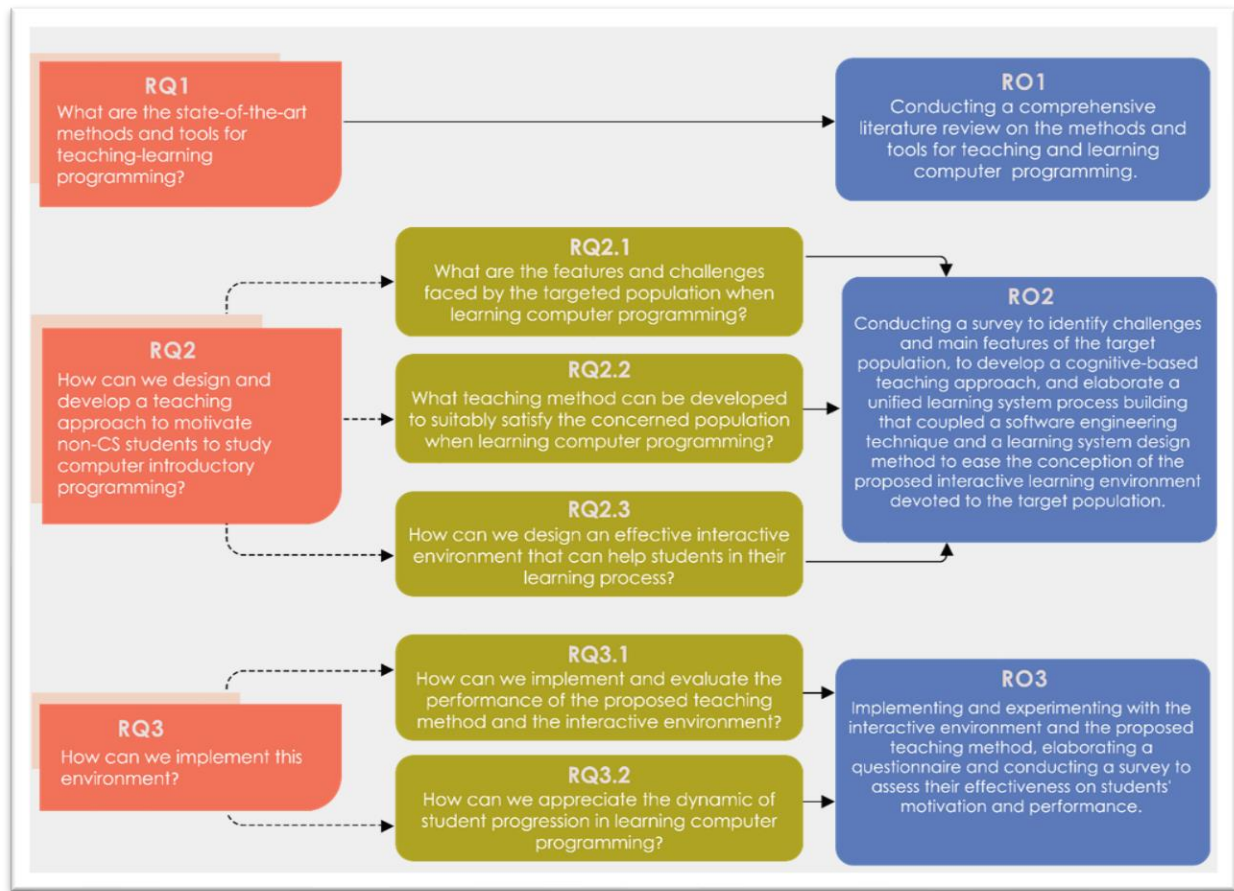


Figure 1.2: Mapping Research Questions to Research Objectives

Research Significance

Computer programming is a thorny issue in computing education research. Practical knowledge of computer programming is a requirement for higher-level computing courses [69] or for acquiring excellent computing thinking skills. A significant degree of mental abstraction and analytical thinking is required to come up with an efficient result [7], [8]. When it comes to learning programming, many inexperienced programmers feel demotivated, disappointed, and overwhelmed [70]. Conventional teaching methods are currently ineffective in teaching programming because they rely on static didactic resources such as books, lecture notes, and slides [71]. Additionally, despite the positive outcomes of existing methods and tools [23], [72]–[78], there remains ambivalence among students, resulting in not all enrolled students completing their programming courses. This study aims to address the shortcomings of existing methods and tools by developing a novel teaching method and an interactive platform that will enhance the

effectiveness of computer programming instructions for non-computing students. As a result, the study contributes to promoting programming success, encouraging more students to follow IP modules, and promoting computational thinking.

Research Contributions

In the realm of knowledge, academia, and the research community, this study has made the following significant contributions:

- We provided an analysis of the methods and tools used for teaching and learning introductory programming.
- We have created a novel teaching approach that effectively meets the needs of the targeted audience.
- We conducted experiments to evaluate how the proposed teaching method affects students' motivation.
- We conducted a conversation about the efficacy of the suggested teaching techniques and compared them to other current teaching approaches.
- We have developed and discussed an effective methodology for designing educational software.
- We specifically designed a comprehensive online introductory programming course for non-CS students. It includes activities to assess students' ability to learn and master syntax, visualizations to enhance understanding of basic programming concepts, and resources focused on everyday problems with minimal mathematical challenges.
- We conducted experiments to evaluate how the proposed learning environment affects the teaching and learning of programming, as well as its usability.
- We discussed the effectiveness of the proposed learning environment and compared it with other web-based learning environments.

Proposed Approach and Solution

To address the aforementioned research questions, this study followed the Design Science Research (DSR) methodology, which is a reference research methodology used in the Information Systems (IS) field to design new artefacts and/or design theory to enhance the present state of practice and existing research knowledge [79]. A research methodology

describes how a researcher plans a study to obtain valid and reliable outcomes that meet the research goals and objectives. This research especially followed the DSRM phases' proposed by Peffers *et al.* [80]: problem identification and motivation, the definition of the objectives for a solution, design and development, demonstration, evaluation, and communication. On the other hand, research methods refer to suitable approaches used in research to address each research question. Therefore, this study employs suitable research methods, such as comprehensive literature reviews or surveys, framework design, implementation, experiments, and evaluations. Consequently, we began with a comprehensive literature review of the methods and tools for learning and instructing on IP courses, which provided us with existing gaps [25]. We then conducted surveys of the characteristics and difficulties encountered by non-computing students in IP modules, which enabled us to develop specifications for an innovative teaching method and learning environment [20]. Based on this information, we created the following framework: We developed a cognitive-based teaching methodology known as the EI framework, a methodology for designing educational software called IIAPDIE [81], and a learning environment called PyLe to teach and learn programming syntax for non-computing students. To assess the effectiveness of the EI framework and PyLe, we further implemented them at two universities [21]. This study conducted four published papers and one paper was accepted for publication. We will cover this section in more detail in Chapter Three.

Research Scope and Limitation

The research designed and implemented a novel teaching method and learning environment for IP modules. We designed these frameworks for non-computing students, as they face more challenges than their CS peers. As a result, the experiments conducted in this study only included non-computing students. PyLe focuses on teaching and learning programming syntax for students, whereas the EI framework offers a teaching context that can improve students' motivation. However, time constraints restricted the scope of this research. However, this did not hinder the completion of the study.

Structure of the Thesis

This thesis is organized as follows:

- Chapter One: Introduction and Background. This chapter covers, among other things, the context of the research, the research area, the problem statement, the research questions and the research objectives.

- Chapter Two: The different teaching and learning strategies and tools that have been created to improve students' programming skills are reviewed and categorized in this chapter. Additionally, the Agile methods and instructional design methods are also reviewed in this chapter.
- Chapter Three: Research Methodology and Design. The design science research methodology and the various methods that are applied to this study are covered in this chapter.
- Chapter Four: This chapter describes an explicit instruction-based methodology for teaching introductory computer programming to non-computing students. This chapter also presents an integrated framework for efficient educational software development. Additionally, this chapter introduces a new interactive programming learning environment for Python named PyLe.
- Chapter Five: This chapter presents and analyses the results of a study into the computer programming challenges faced by first-year computer science students and other students. Additionally, the results of post-experiments of the new method on students' motivation are analysed and discussed. The context of PyLe implementation in introductory Python courses is also discussed, as well as the post-experimental outcomes.
- Chapter Six: Summary, Conclusion and Recommendations. This chapter summarizes the study's findings as well as concludes our research objectives and goals. Contributions to the research are emphasized after a discussion of open research areas and future projects.

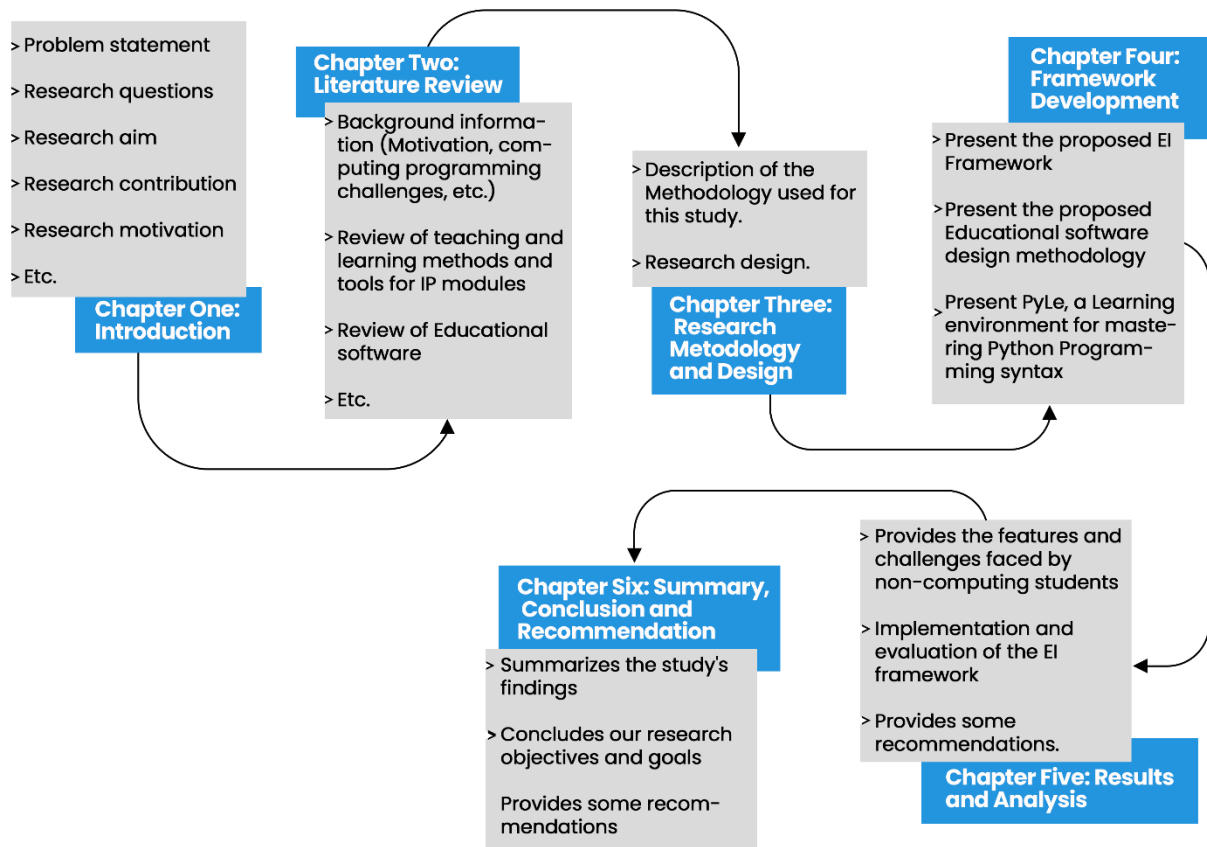


Figure 1.3: Structure of the Thesis

Publications From this Thesis

The findings of this study are presented in the following articles:

1. Mbiada, A., Isong, B., & Lugayizi, F. (2023). A Comparative Study of Computer Programming Challenges of Computing and Non-Computing First-Year Students. *Indonesian Journal of Computer Science*, 12(4), doi:10.33022/ijcs.v12i4.3330.
2. K. Mbiada, B. Isong, F. Lugayizi and A. Abu-Mahfouz, "Towards Integrated Framework for Efficient Educational Software Development" *2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA)*, Orlando, FL, USA, 2023, pp. 53-60, doi:10.1109/SERA57763.2023.10197734.
3. K. Mbiada, B. Isong, F. Lugayizi and A. Abu-Mahfouz, "Introductory Computer Programming Teaching and Learning Approaches: Review," *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, Prague, Czech Republic, 2022, pp. 1-8, doi:10.1109/ICECET55527.2022.9873427.

4. Mbiada, A. K., Isong, B., & Lugayizi, F. (2024). PyLe: An Interactive Tool for Improving Python Syntax Mastery in Non-Computing Students. *Journal of Information Systems and Informatics*, 6(2), 1008-1034, <https://doi.org/10.51519/journalisi.v6i2.751>
5. "Explicit Instruction-based Methodology for Teaching Introductory Computer Programming". Accepted for publication in the *International Journal of Modern Education and Computer Science*: A. K. Mbiada, B. Isong.

Chapter Summary

In this chapter, we provided an introduction to the research and outlined the problem statement, research objectives, research goal, motivation, research methodology, and so on. We will discuss the literature review in the next chapter.

CHAPTER 2 LITERATURE REVIEW

Chapter Outline

This chapter provides background information and examines a variety of learning and instructional techniques and tools in IP courses. This chapter further reviews some agile methodologies and instructional design models (IDM).

Introduction

Currently, educators teach introductory programming (IP) courses in a variety of environments, and students from various backgrounds and features participate in these classes. Therefore, learners from non-CS programs are unavoidably required to take IP modules as part of their academic curricula, as digital fundamental skills are becoming increasingly essential and critical across all domains [69]. In addition, some learners opt to enrol in the IP modules as an optional course, often designed for learners with diverse profiles in overcrowded classes [82]. Nevertheless, the majority of students, who are inexperienced programmers, have an inaccurate understanding of programming tasks and encounter several obstacles. As a result, there have been reports of a high rate of failure and dropout in IP courses, along with a noticeable decrease in motivation [7], [9], [13], [69]. Several educators have expressed concern about the creation and implementation of creative pedagogical approaches to augment students' educational experiences and mitigate elevated rates of academic failure and attrition. This chapter examines several teaching approaches that have developed over time to enhance students' competency in IP modules. Furthermore, it also investigates their classification and the experiences related to them. This chapter, on the other hand, provides a more comprehensive examination of agile methodologies and instructional design models that were used to create a framework for the development of effective educational software discussed in Chapter 4.

State-of-the-Practice in IP Learning and Instructional Techniques and Resources

Firstly, we should acknowledge the publication of this section [25]. Using a descriptive research approach, this section conducted a literature review to examine the various methods, tools, and techniques developed for delivering and acquiring IP courses. The descriptive research

methodology allowed the researcher to consolidate and scrutinize the gathered data. We compiled several relevant research articles published between 2012 and 2023, picked the best ones, and then used a battery of specific inclusion and exclusion criteria to evaluate them. Consequently, we tackled research question 1 (RQ1), as expounded in Chapter 1. We divided the existing approaches into four detailed techniques, based on the findings of each study. Refer to Figure 2.3 for further information. Subsections 2.1.2, 2.1.3, 2.1.4, and 2.1.5 each discuss these techniques in detail. The following subsection first discusses some analogue studies.

2.1.1 Related Studies

This subsection discusses some related surveys or reviews studies on tools and strategies for teaching and learning IP modules.

Luxton-Reilly et al. [83] conducted an exhaustive study of the most current developments in intellectual property and classified them into separate categories: instruction, assessment, educational programs, and student-centred learning. Theories, orientation, tools, delivery, and infrastructure are some of the five broader categories of teaching and learning approaches that the authors defined. Moreover, Kanika et al. [84] also identified the predominant techniques employed in teaching programming, whether at an introductory or advanced level. The research examined module contents that encompassed algorithms, data structures, and fundamental programming ideas. Additionally, there are five generally employed strategies in teaching programming, including robot programming, pair programming, collaborative and visual programming, game-based learning, and evaluation systems. Hooshyar et al. [85] also performed a survey on programming environments that use flowcharts. Their special focus was on the use of iconic notation to enhance the understanding of programming and problem-solving skills among novice programmers. Vihavainen et al. [13] analyzed relevant teaching approaches and determined how they affected students' achievement. The research sorted instructional strategies into five groups, described as follows:

- *Collaboration and Peer Support*: This method incorporates peer-led team learning activities, pair programming exercises, and cooperative and collaborative practices.
- *Bootstrapping practices*: This strategy either arranges a course before the commencement of the basic programming course (CS0) or begins the introduction programming section using a visual programming environment such as Scratch or Alice.

- *Relatable content and Contextualization*: This strategy uses relevant information to make programming more accessible to students. These techniques include Media Computation, real-world initiatives, and courses based on games.
- *Course Setup, Assessment, and Resourcing*: This approach modifies course setup, assessment, and resources by incorporating a wide range of practices such as adjusting course content based on data from an assessment system, introducing new content, a programming tool that provides additional support, and changing the grading schema assessment.
- *Hybrid methods*: These are methods that, after debate, were not classified into any of the basic categories. These involve mixtures of several practices.

Upon implementing these five instructional approaches with the students, the study indicated that there were no noteworthy variations in success rates after the intervention. This implies that while significant enhancements in success rates may be attained by implementing various treatments, the overall success rates remain substantially unchanged regardless of the kind of intervention used.

Based on the above-discussed studies, the perspective methods, tools, and innovations in IP are the subject of only a few surveys and evaluations. Consequently, we noted that the Luxton-Reilly et al. [83] study was more comprehensive and inclusive than other studies. For instance, the categories outlined in [83] readily include the instructional approaches described by Kanika et al. [84] and Vihavainen et al. [13]. Furthermore, [86] defines "tools" as visual programming, game-based learning, and robotics. Conversely, [86] categorizes paired and collaborative programming, along with bootstrapping methods, under the category of instructional delivery. Vihavainen et al. [13] did not include programming tools in their study, unlike [84], those that primarily focused on them. This study identified areas for improvement by critically analysing and synthesizing the methods and tools used in IP instruction.

2.1.2 Deliver-based Approaches

This category of teaching methodologies includes those used by lecturers to communicate knowledge to students, as well as those that involve organizing and structuring participation in the course. There are a variety of sub-approaches for delivery that we have identified, including collaborative programming, active learning, the inductive teaching method, flipped or inverted classrooms, blended learning, agile methodologies, the ADRI model, hybrid approaches, action research, mixed-context teaching approaches, and integrated teaching approaches. All of these methods were student-centred approaches derived from the constructivism paradigm. [87].

A. *Collaborative programming*: Group-based learning enhances collaboration and leads to enhanced learning [88]. Matching inexperienced individuals with high-achieving students improves the analytical reasoning and overall self-assurance of non-computing students [41], while also boosting the academic achievement of students who progress at a slower pace [89]. Furthermore, pairing or grouping beginners with more than two students facilitates the instant comprehension of their work performance by providing specific details and comments regarding any faults committed [72], [82], [89], [90].

Many studies have extensively implemented the concepts of pair programming and agile software development [62], [82]. In this scenario, two programmers engage in a collaborative process where they work together on a single computer simultaneously to address a shared problem, write code, or conduct testing [82], [91]. The aim is to disseminate knowledge and improve the design and code quality [82]. Williams and Kessler [92] outlined seven key ideas that form the basis of pair programming. The principles that contribute to the effectiveness of this approach include peer pressure, peer negotiation, peer daring, peer learning, peer trust, peer review, and peer debugging. In addition, the Scrum methodology is a widely used agile technique that promotes self-directed learning when implemented on an appropriate web-based platform [93].

The research conducted by Carbonaro [94] showed that peer review positively impacted student engagement, time management skills, and competencies in a standard IP course. When students participate in peer evaluation, they have the opportunity to evaluate code, write and read comments, and observe how other students are addressing the same issue. Teachers can only adopt this method once they are confident in the trustworthiness of students and the reliability of the results.

B. *Active learning*: This is an alternative to the conventional method of delivering programming classes, which often involves lectures. The curriculum comprises pedagogical exercises that enable students to engage in practical tasks and subsequently contemplate upon them [71]. Additionally, it is an educational method that prioritizes the students' involvement and encourages them to explore and learn independently, either individually or collaboratively, in a setting with minimal guidance. Furthermore, educators assist and foster students' inventiveness and originality [95]. Souza and Bittencourt [96] examined the effects of problem-based learning (PBL) on enhancing students' motivation and engagement in an IP course. Constructivist theory informs the PBL approach, an active teaching method that enhances students' problem-solving and critical-thinking skills. The authors recommended a smart design for tests and activities to enhance students' involvement and motivation. Furthermore, Șerban and Vesca [95] presented an active technique that is based on four different approaches to student learning. These methods

include project-based learning, workshop-based learning, inquiry-based learning, and collaborative-based learning. Project-based learning is an efficient method for helping novice programmers acquire both knowledge and self-confidence. Workshop-based learning allows students to enhance their projects at the end of the course by incorporating additional functions that surpass the required difficulty level or cover topics not included in the course syllabus. The implementation of inquiry-based learning enabled the students to construct their own assessments for the written examination. The assessment consists of a series of written examinations with multiple-choice questions, which account for 10% of the overall written examination. Collaborative learning promotes the development of learners' critical thinking skills. This activity occurs during seminar courses to evaluate the quizzes submitted by students as an assignment to prepare for their written test.

C. Inductive teaching method (ITM): Khan et al. define ITM as an approach that focuses on the learner and aids in the development of creativity and problem-solving abilities, which are crucial for engineering disciplines [74]. Khan et al. characterized ITM as a pedagogical method that is driven by examples and follows a bottom-up approach. In this technique, the instructor presents students with a collection of facts and enables them to derive their conclusions. It enhances efficient learning in technology education in comparison to the Deductive Learning Method (DTM). It serves as a source of motivation for students to actively engage in class activities, making it a potentially superior method for teaching programming. Khan et al. explained the introduction of variables to students using the ITM technique. By the conclusion of this exercise, students can effortlessly articulate the principles for identifying variables.

D. Blended learning: This method involves combining in-person and online instruction or web-based technologies to achieve a specific educational objective [71]. This method may include the use of streaming video and audio, as well as live virtual classrooms, collaborative learning, self-paced training, and text. The five types of blended learning that Alammary [97] discovered and articulated were the flipped, mixed, flex, supplemental, and online practising approaches. The blended learning strategy that is most well-known is the flipped model, sometimes known as the inverted classroom [98].

E. Hybrid approach: Cakiroglu [73] described the hybrid approach as a technique that integrates comments-first coding, analogy, and template techniques. Firstly, students analyse the provided source code, then apply previously learned concepts to build new ideas, and finally, they utilise preexisting functions called templates to design their solutions. Therefore, the hybrid approach combines multiple teaching approaches to achieve the educational objective [13]. In the same vein, Sharma et al. [99] presented Teaching Tips as a fusion of two blended learning

models, namely flipped learning and the online-practicing model. Similarly, Borovina Josko [100] included creative programming tasks, formative feedback, pair programming, and other techniques in the teaching of IP.

F. Approach, development, result, and improvement method (ADRI): ADRI is a review quality assurance approach that may be used for instruction [101]. When it is applied to IP instruction, the ADRI approach phases align with Malik and Coldwell-Neilson's programming jobs and sub-tasks [22]. This approach combines problem-solving methodologies, using a programming language for implementation, examining problem-solving procedures, and providing updated instructional materials based on suggestions.

G. Action research method: The ADRI paradigm serves as the foundation for this approach, which emphasizes problem-solving strategies and fundamental programming to enhance the abilities of beginner programmers [102]. Currently, instructors extensively utilize action research to assess their teaching methods and engage in thoughtful analysis, since it consistently improves the *process* of instruction and learning.

H. Problem-solving method: One can also view this approach as a pedagogical technique[22], [72], [103] and an achievable goal in IP instruction [85], [97], [98], [102]. As an illustration, Malik and Coldwell-Neilson [22] perceived this technique as a systematic plan that often involves the following sequential actions: conduct a thorough analysis of the problem and develop a solution strategy. Identify the data types and enumerate the necessary procedures to resolve the problem. Ensure that the correct solution is achieved by diligently following the previously outlined processes.

I. Mixed-context teaching approach: This pedagogical method combines Scratch's use in game development with Python, incorporating turtle graphics and image editing. Santana et al. [19] evaluated the impact of the mixed-context teaching approach for IP modules on CS non-major motivation. The results showed an increase in students' motivation.

J. Integrated teaching approach: One can view this approach as a specific type of hybrid approach, as previously discussed. It includes the following elements: Python as a first programming language, project-oriented and problem-based learning, multimedia resources, and assessment rubrics [104]. This approach has been implemented by students at a university, in Colombia, and data was collected from 2011 to 2017, and the analysis of the results showed that it has a positive impact on students' engagement and performance.

2.1.3 Content-based Approaches

This subsection explores several topics aimed at enhancing the understanding of the IP module. It includes papers that provide additional and elective exercises to enhance students' understanding, prepare them for assessments, and assist them in achieving their educational objectives. The recognized categories are as follows: logical thinking, selection of programming paradigm, low-level methodology, supplementary exercises, comprehensive approach, case-based instructional approach, code labelling, and live coding methodology.

A. Logical reasoning: Logical reasoning is one of the most crucial components of understanding computer programming [105]. Cheah and Leong [103] emphasized the importance of logic in the process of mapping various logical parts and explaining programming concepts within the framework of computer programming's program control structure.

B. Low-level approach: This approach proposes starting IP courses by acquiring knowledge of low-level concepts such as machine code and assembly language, thoroughly comprehending them, and subsequently advancing to higher-level programming languages [26].

C. Programming paradigm selection: In an IP course, the selection of a programming paradigm, followed by the programming language, is of the utmost importance. In their study, Koulouri et al. [106] found that teaching students programming principles through the use of a syntactically simple language, such as Python, rather than Java, which is a more complicated language, was significantly more effective. In a similar vein, Kasurinen and Nikula [107] reported that switching from a language such as C to Python resulted in students producing greater grades, improving their level of happiness, and reducing the rates of students dropping out of school and failing their classes. Instructors are required to manage the variances in the use of the Object-Oriented Programming (OOP) paradigm when they choose to use it rather than the procedural paradigm. There are three distinct methods of OOP that Kunkle and Allen [86] presented the object-first, object-early, and imperative-first approaches.

D. Optional exercises: Students have the opportunity to use this content, which is grade-free and optional, for further practice to enhance their confidence. Concerning an IP curriculum, O'Malley and Aggarwal [51] evaluated the applicability and effectiveness of ungraded practice tasks. To assist students in the development of fundamental programming techniques and confidence, Edwards et al. [108] proposed a strategy consisting of short programming exercises, which is a voluntary practice on tiny programming questions.

E. *Holistic approach*: This encompasses a range of activities, including problem-based learning, experiential learning, knowledge development, example-based learning, fostering effective student-teacher communication, promoting collaborative work, and offering feedback. For example, Hartanto et al. [109] demonstrated that the holistic approach improves students' engagement in an IP course.

F. *Case-based teaching approach*: This is an efficient technique for teaching beginners how to grasp the programs. During lectures, the teacher provides students with examples or best practice models. Previous studies pointed out that case-based teaching can improve students' engagement in class activities, critical skills, problem-solving skills, and creativity, which is important for novices to understand programs [24].

G. *Code labelling exercise*: This approach contributes to the teaching activity in the IP module that targets non-STEM students. The goal is to apply grammatically sound techniques to code snippets to address the challenges of teaching and learning IP modules, promote deeper learning, and improve students' programming syntax understanding [110].

H. *Live-coding*: This method is crucial for teaching the programming process, as it involves the instructor verbally explaining and articulating their thoughts as they consecutively write the code, while the students observe. Raj et al. [111] assessed the influence of this instructional method on students' IP learning. The findings demonstrated that the practice of live-coding facilitates novice programming comprehension, enhances students' ability to identify and correct code errors, and equips them with effective programming methodologies.

2.1.4 Tool-based Approaches

This section examines the studies that provide a comprehensive analysis of the tools utilized by instructors to transmit IP content to students. These tools include game-based learning environments, web-based learning environments, visual programming environments, visualization programming environments, and robot programming tools.

A. *Game-based learning environments*: The rapid progress in technology has led to a surge in the popularity of digital games and their use in several fields. Within the realm of education, serious games serve to accomplish significant goals. Anastasiadis et al. [112] found that these games stimulate and involve students, offering interactive and collaborative learning environments and activities. Soares et al. [113] also mentioned that game design can attract and keep people interested in the computer science field. Piteira et al. [114] described methods for implementing gamification in an online programming course using a specified theoretical

framework. According to their assertion, gamification has the potential to result in shorter cycles of receiving benefits, improve students' capacity to assess their abilities and establish a setting that incentivizes learning efforts. To enhance the understanding of fundamental OOP concepts, Tsiotras and Xinogalos [115] developed a text-based serious game named "Rise of the Java Emperor". In addition, Hooshyar et al. [116] created a sophisticated online game-based Bayesian Intelligent Tutoring System (OGITS) to improve students' proficiency in web-based problem-solving through the use of board games. OGITS creates a favourable environment for students by providing both a standalone course and a typical classroom supplement.

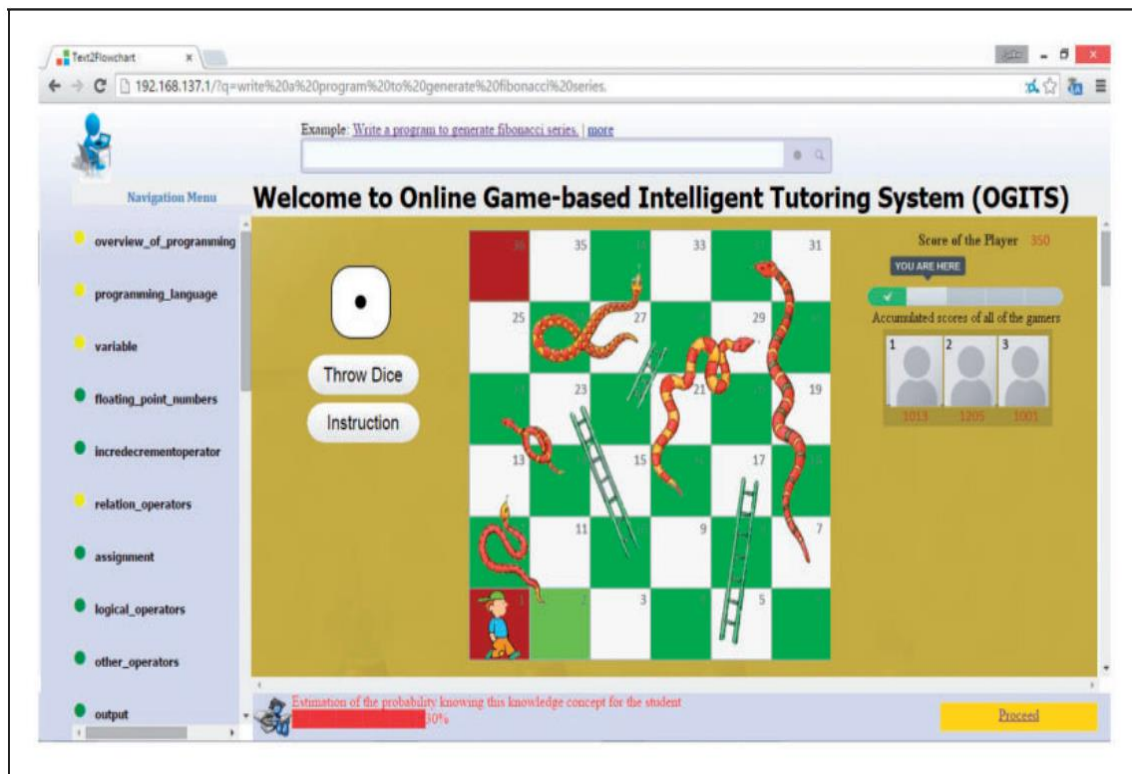


Figure 2.1: OGITS Main Page [116]

On the other hand, Facey-Shaw et al. [117] proposed a simulated programming learning environment that supports game-based simulations and measurements and boosts student motivation. Quantitative findings indicated that digital badges do not improve intrinsic motivation, but qualitative results showed that they do. Moreover, Jiau et al. [118] reported on a learning environment for simulated programming that aims to support the implementation of game-based simulations and metrics and improve students' self-motivation. This environment teaches fundamental programming abilities by building a computer-winning gaming strategy. Other research studies [119], [120] have also pointed out gamification as a good strategy to enhance students' motivation and engagement.

B. Web-based environment for programming: Educators are increasingly utilizing web-based educational platforms, whether in an online or blended learning setting, to mitigate the elevated rates of student attrition and academic underachievement. Additionally, certain tools have incorporated virtual programming laboratories to streamline and enhance online programming skills. We analyse the discovered programming environments and examine their current use in IP courses. Ramos et al. [121] introduced the Virtual Programming Lab module for Moodle (VPL-Moodle), which provides comprehensive online help for learning programming. VPL-Moodle is a Moodle module that is open-source and can assess over 40 programming languages. It also provides automatic evaluation. In addition, Gonzalez and Biørn-Hansen [87] suggested using a collaborative programming method using a web-based tool called Scrimba. This approach aims to enhance students' motivation, code quality, retention rates, and confidence. Scrimba is a web-based platform that enables users to collaborate on learning activities by sharing code and viewing each other's screens in real-time. It also allows users to create copies of projects for development purposes and share their changes and comments with others. In addition, Salleh et al. [23] investigated the efficacy of cloud-based learning systems (CBLS) in improving students' programming skills by assessing their problem-solving abilities in programming. Students utilize PHP programming on Codecademy as a surrogate measure for computer-based learning systems. Sharp [78] has also used Codecademy as a supplementary instructional resource for learning Python programming. Codecademy provides interactive classes on a variety of programming topics and teaching materials, such as class resources, classroom tracking, and teacher training [78]. Edgcomb et al. [75] created online learning material that allows students in an IP class to read and record their homework activities. Hunter et al. [6] created and put into action a tool named Nooblab that serves the purpose of providing education, testing, and assessment for a range of programming languages. Nooblab provides educational material in the form of a user-friendly Javascript-based program editor, a virtual console, and feedback on students' progress. Silva et al. [122] also investigated the impact of Colango, a multi-platform educational tool for programming classes. Colango facilitates algorithmic learning and provides a programming interface for tracing and debugging programs. In this scenario, students utilize their native language such as Portuguese to write code, focusing mostly on comprehending the logic of the programming language rather than its syntax and semantics. Saltan [123] examined the influence of online algorithm visualization (OAV) on students' performance in IP courses. The objective was to assist students in effectively managing the challenges encountered during the process of learning programming. Anyone, even without any prior programming knowledge, can access OAV programs through the internet on any device and receive visual programming instruction. Students can choose from a variety of courses that align with the instruction's specific

goals. Additionally, this system enables real time monitoring of their progress. Brusilovsky et al. [124] devised an architectural framework to integrate many intelligent content systems into a single system, resulting in the development of the Python Grids training system. The system works on servers located on two different continents, offering an optional learning environment for Python and emphasizing fundamental programming principles. It provides for a wide range of student requirements, providing a strong and adaptable learning experience. Similarly, Mutiawani and Juwita [14] explored the development of a customized e-learning application for IP courses that encompasses diverse activities, code practice, graphics, sounds, animations, and text. The code practice part uses EditArea, a freely available JavaScript editor, to improve code legibility by implementing syntax highlighting. In a related vein, Samat et al. [15] examined the development and application of a constructivist multimedia learning environment to enhance the programming abilities of computer science students. The primary emphasis is on acquiring knowledge, honing programming abilities, and utilizing technology. It encompasses both fundamental and advanced programming principles. Assessments demonstrated the efficacy of the program for kids. In a similar vein, Virvou and Sidiropoulos [125] presented a novel e-learning platform with collaborative capabilities designed specifically for Python programming teaching. It facilitates group management and encourages student engagement. Additionally, it performs the role of an intelligent tutor, creating a personalized student model through interactions and recommending courses tailored to each student's profile. Furthermore, Stupina and Paniotova [126] created Chatbox, an interactive tool for programming instruction in a blended learning environment. The Chatbox platform enhances student engagement, facilitates the achievement of learning objectives, and fosters mobile and interactive learning. The system adjusts to the individual student's learning requirements and preferences, allowing one to study at their own pace and providing feedback. Similarly, Kakeshita and Murata [127] employed pgtracer, a tool that allows users to fill in missing information, as an aid for homework in programming courses. Pgtracer, a Moodle plug-in, aids novices in comprehending programming fundamentals like loops, functions, and pointers. Students complete the missing parts, and pgtracer illustrates the program's sequential execution, compares the output to the correct solution, and automatically assesses the student's performance. Pgtracer also possesses data-collection functionality. Ferreira et al. [128] also examined and assessed SICAS2, a Moodle plug-in that is grounded in constructivist philosophy. SICAS2 enables students to generate programs using flowcharts and visually represent the program's execution. It applies to all programming languages and facilitates students' comprehension of fundamental programming concepts such as input/output operations, iterative loops (for and while), and conditional statements (if and if/else).

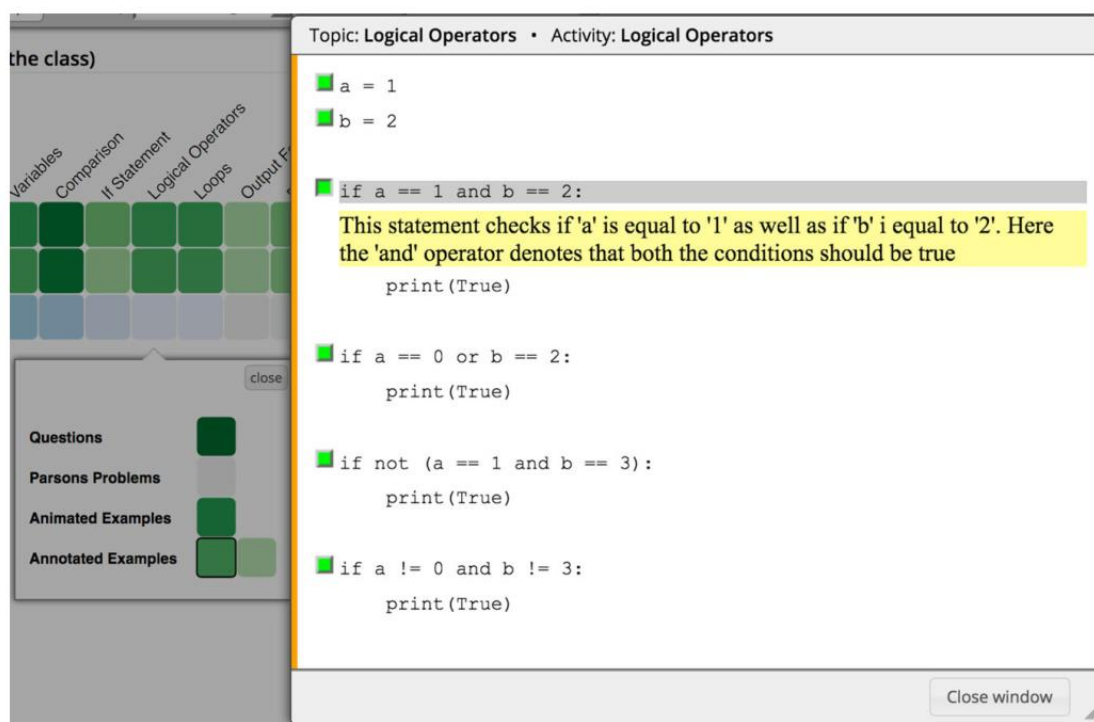


Figure 2.2: Sample View of Python Grid System [124]

C. *Visual environment for programming:* These tools provide a user interface consisting of icons that represent fundamental programming concepts like conditional and iterative statements, along with function calls [84]. Students can create a flowchart-style program by simply dragging and dropping the required icons into the appropriate order for a particular problem. In most cases, students with backgrounds in fields other than CS and novice programmers benefit from using these visual tools, which come equipped with an interpreter for program execution [129]. Hooshyar et al. [85] identified several flowchart-based environments that are available for improving the understanding and problem-solving abilities of novice programmers. A few of the available tools are BACCII and BACCII++, FLINT, and others. Furthermore, Cárdenas-Cobo et al. [129] introduced CARAMBA, a Scratch-developed web application that includes an editor and a recommender system for executing exercises. Scratch is a visual programming tool that teaches fundamental programming concepts to novices with little to no prior programming experience. Kyfonidis et al. [130] introduced a block-based programming teaching tool designed to simplify learning C IP courses for beginners. The tool progressively translates visual models into programming concepts, allowing students to build programs by dragging and dropping blocks. However, it emphasizes logic over syntax and provides the option to convert block-based code to C text code, as well as vice versa. In the same vein, Jung et al. [3] built an interactive block-based environment to support program creation using the visual programming language and code

generation in the C programming language. Concurrently, Moussa et al. [4] designed OOPVisual, an interactive 3D visualization tool, to enhance novice females' understanding of Object-Oriented Programming (OOP) concepts, particularly polymorphism. OOPVisual employs the drag-and-drop technique, which ultimately helps students disregard syntax errors. It incorporates tutorials detailing the concept of polymorphism, quizzes, and exercises to aid students in practice.

D. Visualization programming environments: The visualization environments are likewise web-based platforms that provide learners with the ability to graphically see the various stages of program execution. The goals of this endeavour are to improve code comprehension, code construction, and understanding of the majority of fundamental programming ideas. In addition to using virtual compilers, the majority of these environments provide students with some degree of interactivity. Consequently, Rowe and Thorburn [131] presented Visual Instruction for Novices in a C Environment (VINCE), a web application that is based on Java and graphically illustrates the execution of a valid C program. This tool allows students to view or develop a C program and evaluate its detailed execution. This tool presents a fundamental understanding of programming principles, including pointers, structs, arrays, function calls, and dynamic memory allocation. Similarly, Hijón-Neira et al. [132] introduced the Visual Execution Environment (VEE), a tool that instructs CS1 students on fundamental Java programming concepts. Java operations are guided by visual metaphors, and integrated applications with preloaded scripts are provided by this programming language. During their studies, students can choose from a wide range of scripting exercise alternatives provided by these scripts. In addition, Yan et al. [133] developed the Programming Visualization Tool (PROVIT) for Web, which is an e-learning platform for C programming that is based on Java. PROVIT facilitates the writing, execution, verification, and visualization of C programs. In contrast to many other self-study programs such as VILLE, VINCE, WADEIn, Jeliot, and VIP, ProVit is also suitable for use in formal classroom settings. Each of these environments significantly enhances the programming learning experience, each with its unique characteristics.

E. Robotic programming: This entails the utilization of a robotic device for instructing programming courses, which has the potential to captivate and inspire students since they will no longer be preoccupied with the intricacies of language syntax [84]. Majherová and Králík [134] examined students' programming experiences with Lego Mindstorms, Bricx programming environments, and the virtual educational environment ROBOTC. These tools are the primary reasons why teaching programming languages in the classroom is attractive. Moreover, the use of a graphical robot technique in a Java programming course effectively motivates students both within and outside of the class. They said that this teaching method motivates students by

showing them their progress instantly. Visualizing results drives student motivation. Most of these students were future high school informatics instructors without programming experience. The qualitative survey (interviews) found that lack of time, the teacher's explanations not matching the students' comprehension, and motivation, and the difficulty of forming a group were the main reasons students struggled in IP courses.

2.1.5 Assessment-based Approaches

IP modules specifically aim to assist students in understanding essential computer notions and engage them in small programming tasks to foster their passion for computing [135]. As a result, it is essential to have reliable and appropriate measurements to effectively track their development and understanding of significant concepts. Scientists have developed a variety of evaluation procedures and equipment to assess students' comprehension: Dawar Murphy [77] introduced a pedagogical method called assignment a day and shared their personal experiences with it. The technique necessitates that students engage in consistent practice to effectively manage and overcome stress related to meeting deadlines and acquiring necessary supplies. This method entails the completion of one assignment per day and a maximum of four assignments per week, instead of having to write tests and conduct experiments for each module. In CS1, two studies [136], [137] investigated the application of an auto-grader called many small programs to students' activities. They assigned several small tasks to students every week instead of one huge program, utilizing the online platform zyBooks. Maguire et al. [76] conducted similar research, which showed that the HackerRank system's automatic machine assessment allows students to generate high-quality code on their own and promotes independent development. HackerRank is a platform that provides solutions for competitive programming assignments in Java, C++, PHP, Python, and SQL, including a wide range of CS fields. Al-Khalifa and Devlin [89] introduced an internet-based peer-assessment method to enhance students' programming expertise, skills, analytical thinking, and learning mindset.

2.1.6 Discussion

The field of teaching and learning IP courses is experiencing a surge in research, leading to the creation of various teaching methods and technologies. The overall goal is to reduce or completely eradicate the high rates of student failure and dropout that are common in these courses around the world. This subsection thoroughly examined and presented comprehensive research on the learning and teaching tools and methods of IP units. Therefore, various methodologies and resources are available to enhance students' comprehension and engagement, problem-solving abilities, self-assurance, active learning, and programming

proficiency. This study effectively categorized the findings into four distinct categories: delivery-based, content-based, tools-based, and assignment-based approaches, as summarized in Figure 2.3. Furthermore, the majority of these techniques advocate for a student-centred approach, and the learning materials offered include collaborative programming and auto-graded evaluation, among others. Overall, the results indicate that current methods and tools have a substantial beneficial effect on students.

However, despite great efforts to create and execute these tools and approaches, their use encounters some obstacles and constraints, as detailed in [25]. Furthermore, while the instructional approaches outlined in the "content-based approach" and "delivery-based approach" categories are important, only a few of them focus on student engagement and motivation in IP courses. This analysis failed to include the prevailing lack of motivation among the majority of novices, which therefore results in higher rates of both failure and drop-out in IP courses [107], [138]. Moreover, constructivist and socio-constructivist theories, including active learning, problem-based learning, and peer review, have primarily influenced the majority of these approaches. Regrettably, the constructivism approach necessitates that students allocate additional time to resolve difficulties and acquire problem-solving skills. Consequently, they are prone to making more errors and experiencing heightened levels of frustration. Ultimately, they will acquire a diminished amount of knowledge. As a result, the constructivism theory suggests that minimal or unguided monitoring during training is not very successful for beginners [139], and for non-computing students in particular. Furthermore, these approaches primarily targeted computing students rather than non-computing students.

Conversely, this literature review provided some of the digital tools to improve beginner IP skills, highlighting the importance of this in CSE. Again, these tools often do not explicitly address their application to non-computing students. Even though these tools usually avoid syntactic problems, previous research [20] shows that syntax is still a big problem for these students [61], and teachers also struggle to modify these tools to meet their requirements. Furthermore, dynamic content and adaptability are lacking in large-scale learning management systems [140].

Consequently, a previous study by [20] suggested that implementing a cognitive teaching approach might benefit non-computing students. This strategy has the potential to enhance their motivation, engagement, and performance. Moreover, this study also proposed an interactive learning environment to improve non-computing students' understanding of basic programming concepts and syntax, aiming to fill existing gaps and offer a more customized and effective

learning experience. Chapter four will introduce a novel instructional approach and learning material devoted to non-CS students.

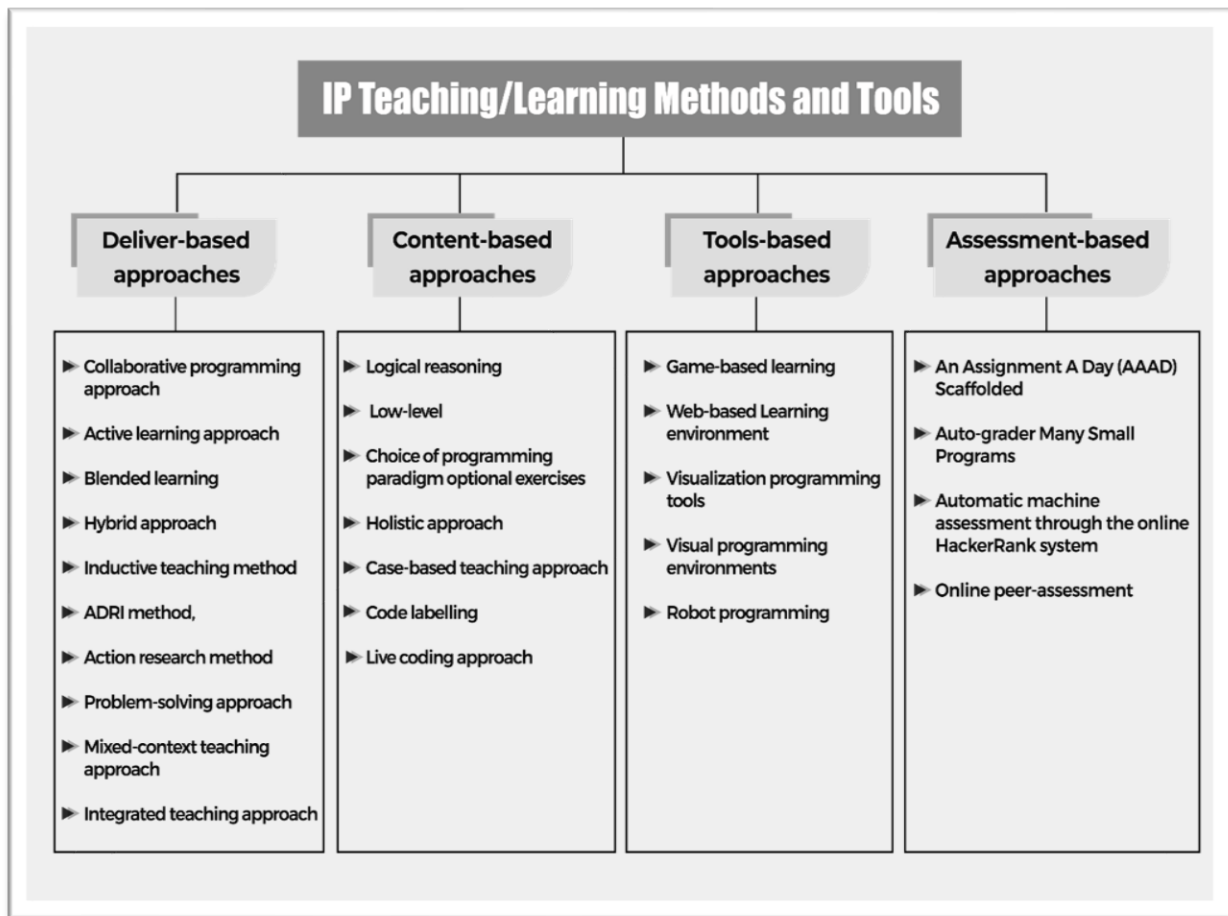


Figure 2.3: Summary of the Learning and Instructional Approaches and Materials for IP Modules [42].

Agile Methods

We should acknowledge that this section was published in [81]. Agile methods are contemporary software development strategies that incorporate people and processes, customer interaction, iterative development, workgroups, and change [141]. Agile methods are also known as agile software development. In comparison to traditional software development methods, which involve stringent protocols, tools, and documentation, these approaches are more intuitive and straightforward. Traditional methods were not only inefficient and expensive, but they also occasionally failed to meet the requirements of the client [141]. Agile methodologies help software management teams deal with ambiguous or rapidly changing requirements by utilizing four ideals or manifestoes and twelve principles, as illustrated in Figure 2.4 [141]. Software development, on

the other hand, is characterized by many obvious constraints, such as its reliance on customer participation and inability to provide maintenance due to inadequate system documentation [142]. Agile approaches such as Scrum, Extreme Programming (XP), Dynamic Systems Creation Method (DSDM), Features-Driven Development (FDD), Kanban, Requirements-Understanding (RUP), and Design Thinking, amongst others, are beneficial to the creation of software [9], [52]. These methodologies are consistent with agile manifestos and concepts, but each has its own practices and procedures. As a result, this investigation will focus on Scrum and DSDM because of the significance of their respective steps, accompanying descriptions, and objectives.

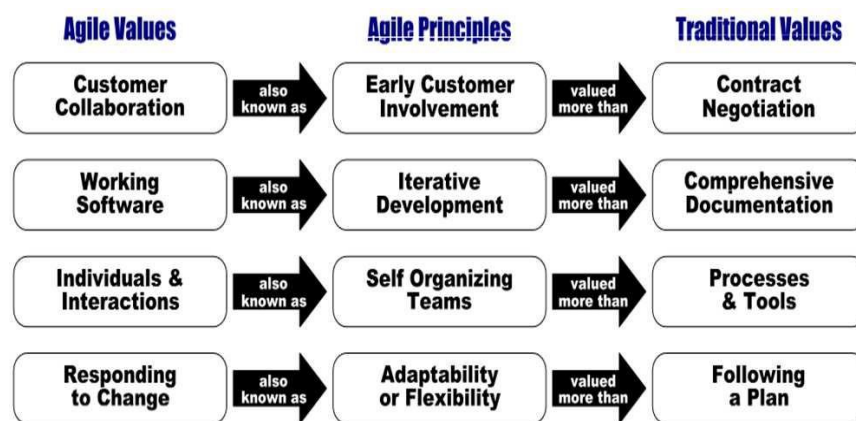


Figure 2.4: Agile Method's Values and Principles [141]

A. *Scrum*: This flexible framework efficiently manages tasks related to software development. The approach is based on the principles of industrial flow control theory and emphasizes adaptation, flexibility, and productivity [143]. Scrum utilizes brief iterations, known as sprints, along with extensive cooperation among all parties involved, a sustainable rate of development, and a team that organizes itself [[9],[10]. Furthermore, this method is characterized by its agility and has a well-defined project management framework that involves three key roles: the Scrum product owner, the Scrum Master, and the development team [9],[10]. The product backlog and sprint backlog express tasks, ensuring transparency and enabling inspection and modification possibilities [143]. The Scrum approach comprises three primary roles: the product owner, the scrum master, and the development team [144], [145]. The product owner is responsible for selecting team members, with the assistance of the scrum master. In addition, he or she establishes the project's limits and consolidates the consumer requirements into user stories. The Scrum Master's role entails validating the implementation of the Scrum methodology and fostering effective collaboration between the development team and the product owner. Five to nine individuals, each possessing expertise in multiple fields and capable of self-management,

comprise the development team. Their primary responsibilities include interacting with end-users, customers, and the product owner to comprehend the business needs, collaborating to define specifications, writing code, validating the product, documenting the process, and ultimately delivering a high-quality result. As a result of its widespread application and its suitability for the organization of team members and the management of projects, this research chose Scrum as an efficient framework for the design of ES. We made this decision in light of the rapidly changing nature of our world. Table 2.1 illustrates the difference between Scrum and other methodologies.

Table 2.1: Comparing Scrum with other Agile Methods [145]

Scrum	Other Methods
Implementing more directive and official meeting structures, clearly defined roles and iterative processes.	Kanban is characterized by its lack of prescription, as it does not involve formal meetings, fixed roles, or certain iterations.
Priority is given to productivity to achieve better flexibility and fulfil the needs of customers.	XP's communication is sporadic, and it entails more intricate protocols.
Efficient team member collaboration, less complexity.	FDD communication is sporadic and entails more complex procedures.
Better collaboration among team members.	The communication among team members in the DSDM approach is inconsistent at best.

B. DSDM: Built on agile methodology, this framework for rapid application development emphasizes collaboration between stakeholders [146]. Additionally, it establishes a framework of guidelines for the efficient and expedient development and delivery of quality software. The development process significantly stresses stakeholder interaction and utilizes an incremental approach [144]. Therefore, it is a proficient method to generate software that guarantees that the result aligns with the progressing business requirements of the organization. According to [144]–[146], the DSDM life cycle comprises five components: feasibility study, business study, functional model iteration, system design and build iteration, and implementation. The feasibility study aims to assess the suitability of DSDM as a method for the project. This step also involves the analysis of risk. The feasibility report documents the results. The development plan is also elaborated. The client and the development team organize meetings during the business study phase to define relevant data, including the users' requirements, a prioritized list of functionalities, the technology to use, a detailed process description, ER diagrams, object models, system architecture, and a prototyping plan. Each iteration in the functional model iteration phase is incremental and iterative, involving analysis, coding, and prototype development. The prototype does not have to be perfect all at once. It will undergo regular improvements to be able to integrate the final product. We prioritize the functionalities here. At the end of this phase, we produce a

functional model that includes the prototype's code and the analysis models. At the stage of system design and build iteration, the software reaches a mature enough stage for delivery to the customer, who tests it to verify its compliance with user requirements. The user feedback will then guide future development. The implementation stage involves moving the product to real-world production for effective use. The user manual is also provided with the review project report, and training is organized for the users. The project concludes at this stage if the product satisfies all user requirements; if not, the process undergoes a complete redo.

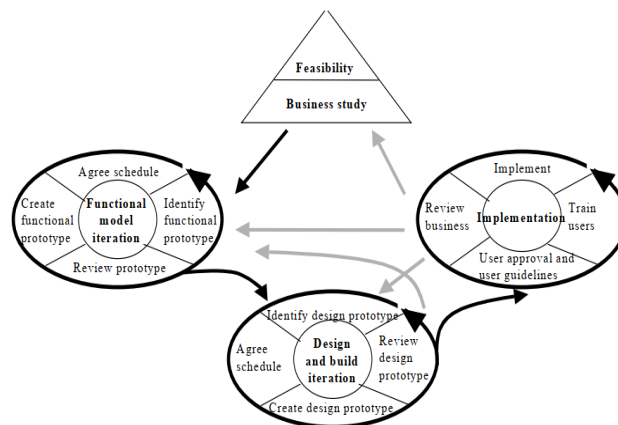


Figure 2.5: DSDM Components and Process Diagram [146]

Instructional Design Models

We should acknowledge that this section has been published [81]. Within the context of educational software design (ESD), instructional design (ID) refers to a framework and field of study that focuses on the methods and principles used to develop dependable and effective learning materials [147]. The primary goal is to facilitate the dissemination of educational resources that adequately meet students' needs and align with the training program's objectives. Instructional design models (IDM) provide instructions for organizing and structuring the process of designing and developing learning materials. Over time, various models for improving the design of learning materials have been developed and can be implemented in a variety of settings. ADDIE, ASSURE, SAM, Kemp Design Model, Dick, and Carey are among the frequently used instructional design models [147]–[150]. This research chose to utilize the ADDIE and ASSURE techniques to create an efficient framework for building instructional software. Unlike other

methods, the ADDIE technique provides the opportunity to use both sequential and interactive methodologies for projects of different scales. Moreover, ADDIE adheres strictly to the software engineering life cycle and stands out for its simplicity. Moreover, the ASSURE technique is not only easily comprehensible and applicable, but it also offers a systematic framework for arranging Scrum sprints.

A. *ADDIE*: Due to its linear and interactive nature, this generic IDM model applies to all project sizes. The acronym ADDIE defines five phases that represent different stages of the elaboration of learning material or system: analysis, design, development, implementation, and evaluation [148], [149]. The ADDIE's success is because it is simple to learn and use, flexible, versatile, and cyclic. It also allows for the correction of mistakes made in previous phases to improve the material's quality. Here is a description of the various phases of ADDIE's model: The analysis phase aims to identify the target audience, its features, prerequisites, content, the goals and objectives of the training, the context of learning, the teaching and learning constraints, and so on; The design phase aims to specify the learning objectives, develop the instructional strategy, select the learning medium, and elaborate on the model and prototype of the learning system; The design involves the personalization of the content, the definition of the strategies and learning methods, tools and assessment modes, communication and collaboration tools, and tools to track students' progress; The development phase assembles all the elements created during the previous phase. Thus, the content and activities are created, and the learning system is formalized; The implementation phase aims to make the learning material accessible to both the target population and teachers. It necessitates the establishment of an organization and a technological infrastructure that can include the drafting of procedures, as well as training, communication, and a support plan for future users; and the evaluation phase involves evaluating the quality and effectiveness of the learning material, confirming the achievement of the initial objectives, and making any necessary adjustments. We do formative evaluations of the various components of the learning material at different phases of the ADDIE model, not just at the end of the process (Figure 2.6).

Albeit commonly used for designing learning material, the ADDIE model presents some drawbacks, as detailed in [149], [151]. For instance, the model is not iterative enough for complex problems; it supposes that all the requirements must be known at the beginning of the project before developing the content.

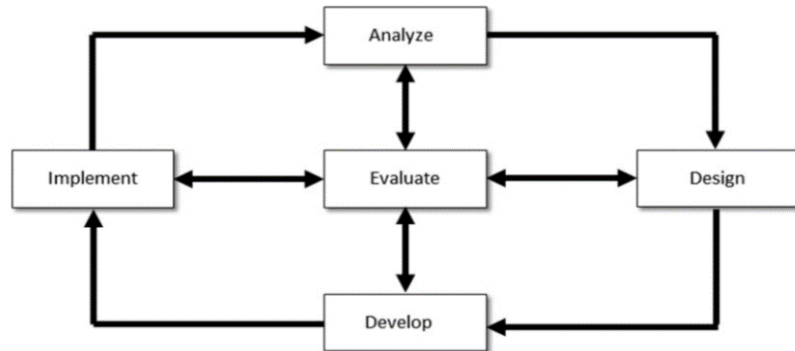


Figure 2.6: Component of ADDIE’s Model [149]

B. ASSURE: Heinich et al. [150] (1989) developed this model, which serves as a guide for planning and delivering learning activities that integrate technology. The ASSURE model is an ideal model for writing lesson plans. ASSURE is an acronym that means: Analysing learners: In this section, we examine learners' features, prerequisites, and learning styles; Stating objectives: This should take into account the target audience features collected from the above analysis, the learning environment, and the time devoted to learning; Selecting methods or strategies, technology, and materials: At this stage, the teacher should outline the expectations for students at the lesson's conclusion. Thus, adequate strategies and media have to be defined to achieve favourable outcomes; Using the available resources: prepare the learning materials, the environment, and the learners, then deliver the lesson; Requiring learner participation: Choosing activities that involve the learners as much as possible; Evaluating and revising: The assessment focuses on achieving the objectives, selecting the appropriate media, and implementing effective methods. We also make revisions for improvement.

Chapter Summary

This chapter reviewed IP course teaching approaches and resources to address student issues, such as lack of enthusiasm, high failure and dropout rates, and so on. This study classified tools and instructional techniques as delivery, content, tools-based, and assessment-based. The goal was to compile and summarize IP course resources and teaching approaches. On the other hand, this chapter discussed the Scrum method and the Dynamic Systems Development Method

(DSDM), two famous agile software development methods. This chapter also analysed the ADDIE and ASSURE methods, two instructional design models. These methods and models were used in designing the suggested efficient educational software design methodology presented in Chapter 4. The next chapter introduces the research methodology and design used to achieve the research objectives.

CHAPTER 3 RESEARCH METHODOLOGY AND DESIGN

Chapter Outline

This Chapter describes the research methodology and design this study applied to answer the research questions defined in Chapter 1.

Introduction

Research, whether basic or applied, is a structured, organised attempt to study a specific problem that has to be solved using a well-thought-out strategy [152]. A research methodology explains how a researcher plans a study to obtain valid and reliable outcomes that meet the goals and objectives of the research. Thus, this research aims to design, implement, and evaluate an explicit instruction-based teaching methodology and an interactive environment for mastering and understanding programming syntax for non-computing students. This research was inspired by the gaps identified in the relevant literature review chapter. Therefore, to accomplish our research objectives, we have chosen to employ the DSR approach as our research technique for this study. This methodology requires problem identification and motivation, development, execution, assessment of artefacts, and communication of the findings. As a result, it provides a clear direction for achieving the research objectives of the study, as outlined in Chapter 1. Figure 3.1 summarises our research process. This study started with a thorough literature analysis of IP modules teaching and learning methods and tools to identify deficiencies. Chapter 2 presents the findings from this phase. The assessment of non-computing students' features and challenges in IP modules helped us outline some specifications for an innovative instructional method and learning tool (refer to Chapter 5 for more details). Based on this information, we developed the EI framework, the framework for educational software design, and PyLe to assist non-computing students in mastering programming syntax, as detailed in Chapter 4. Chapter 5 reports the results of our application of the EI framework and PyLe at two universities to test their efficacy.

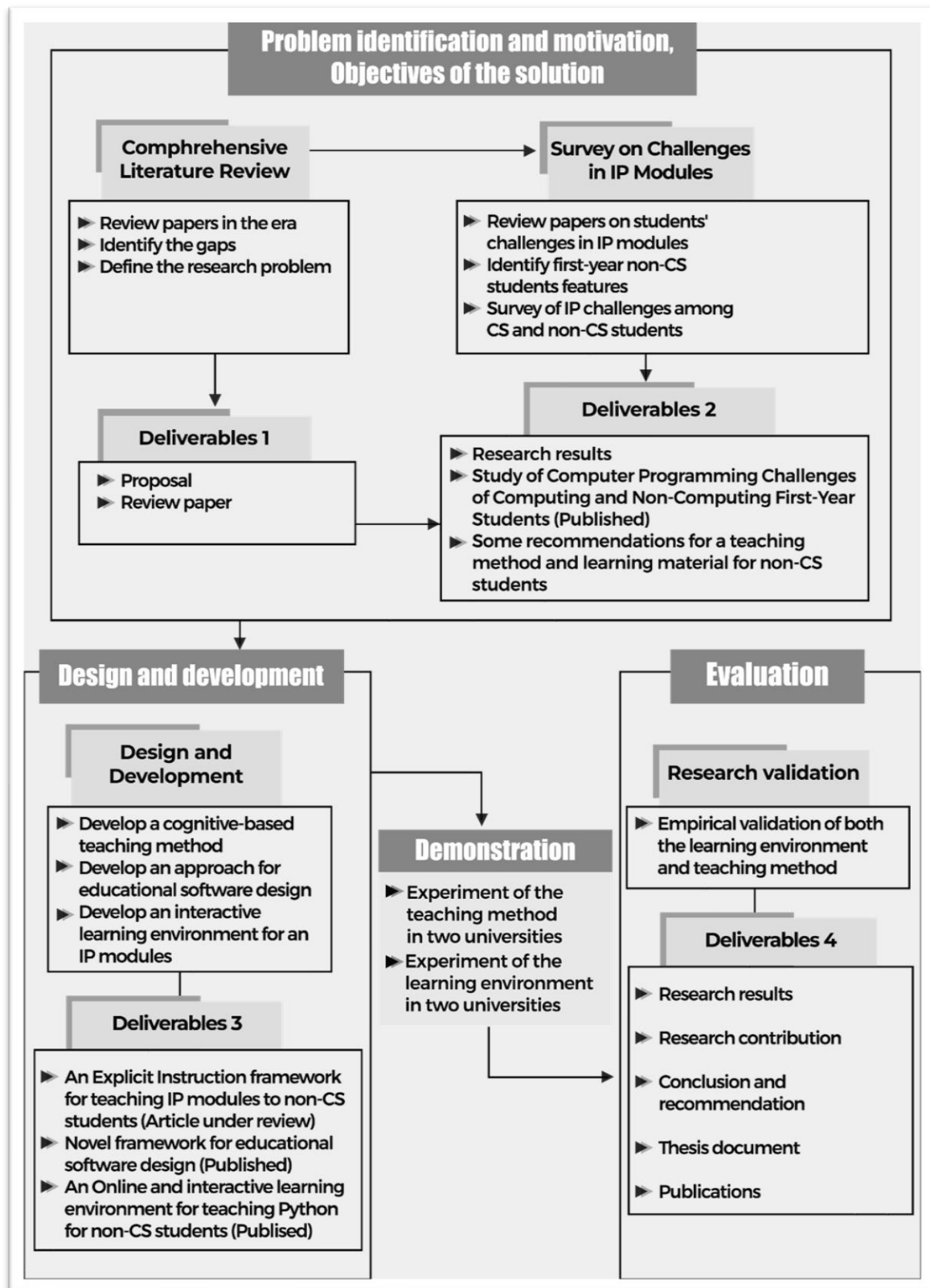


Figure 3.1: Overview of the Research Process

Research Methodology

A. Qualitative vs Quantitative

Methodology is a systematic approach to conducting research that includes a set of approaches and processes [153]. It embodies the basis for scientific investigation. There are two sorts of methodologies: quantitative and qualitative [154].

Qualitative research: This method provides a complete and detailed description of the research subject. It is generally more exploratory and focuses on interpretations, experiences, and their meaning. Qualitative research helps the researcher to understand the "why" underlying the findings. It focuses less on statistics and organized data and instead emphasizes emotions, motives, and human behaviour via textual information. Individual interviews or focus groups are common forms of qualitative research that rely on in-person contact. Researchers often do it in a natural setting, observing individuals as they are, without comparing them to control groups. While the qualitative technique allows you to obtain a deeper understanding of your study topics, the findings are typically more challenging to interpret. Qualitative research collects data to describe a topic rather than quantify it. It focuses on feelings and opinions rather than basic data.

Quantitative research: This method aims to prove or demonstrate facts and focuses more on counting and classifying characteristics, as well as creating statistical models and figures to explain observations. It provides the researcher with a clearer picture of what to expect from his or her research than qualitative research. A quantitative study allows you to draw broad conclusions about your research and forecast specific actions. It employs an empirical approach to collecting numerical data to measure variables. The organization of the collected quantitative data is based on statistical principles.

Consequently, the early phases of research projects are ideal for qualitative research, while the latter part of the project strongly recommends quantitative research. In this vein, this study collected data using both quantitative and qualitative methods. We used the quantitative technique to gather data from the EI and Pyle framework experiments proposed in Chapter 4 and employed the qualitative method to collect articles during the literature review stage. We also conducted interviews with instructors to evaluate the use and acceptance of the PyLe environment.

B. Design Science Research

A research methodology explains how a researcher plans a study to obtain valid and reliable outcomes that meet the goals and objectives of the research. We chose the DSR strategy as

discussed in Section 0 because it explicitly emphasizes the production of both practical solutions and theoretical design models, unlike other approaches like constructive research, which primarily focus on enhancing existing knowledge or models [80], [155]. The DSR methodology requires research to generate a considerable artefact to solve a particular problem. A framework, models, design theories, architectures, constructs, procedures, and/or instantiations can all be considered examples of that artefact [156]. It is widely acknowledged that "*...the artefact must be relevant to the solution, and its usability, quality, and efficacy must be carefully assessed.*" [80]. These DSR requirements align with the aim of our research, which was to create, implement, and assess the following artefacts: an EI-based IP teaching methodology, a framework for efficient educational software design, and a learning environment called PyLe for mastering and understanding Python programming syntax for non-computing students. Moreover, DSR also enabled us to validate the EI-based teaching framework and PyLe by combining qualitative and quantitative techniques presented above. By using DSR, we built theoretical solutions and developed practical, useful artefacts for teaching and learning IP modules. The following DSR steps summarized in Figure 3.1 are involved in achieving our study's objective:

Problem identification and motivation: We conducted a literature review to identify various teaching and learning methods and tools to improve undergraduate students' skills in IP modules. The process of searching for articles involved using logical operators to combine specific words to identify keywords that caught our attention on Google Scholar, ScienceDirect, and IEEE Xplore. These words included "introductory programming," "teaching-learning methods and approaches," "teaching-learning tools," "paired programming," "blended learning and programming," "programming in a web-based environment," "review," and others. We conducted a qualitative analysis of the chosen papers and subsequently highlighted their deficiencies. The study concluded that although the approaches utilized were effective and efficient, there is still a serious concern over the recurring issues of students suffering, failing, or giving up, particularly among non-computing students.

Objectives of the solution: This research aimed to address the shortcomings of existing methodologies by creating a new teaching method and a learning programming environment designed for non-computer students. Furthermore, to tailor the solution to its target demographic, this research conducted a survey that highlighted the features and challenges non-computing students encountered with IP modules. We also aimed to facilitate the seamless integration of the learning programming environment into instruction activities.

Design and development: Based on the results of the previous phases, this research developed three artefacts: firstly, a novel teaching programming methodology named IPEI; secondly, an efficient method for educational software design called IIADPIE; and third, an interactive learning programming environment called PyLe.

Demonstration: We carried out two case studies, one in South Africa and the other in Cameroon, involving first-year non-computing students, and implemented the PyLe environment and IPEI framework to assess their efficiency. The IPEI methodology experiments lasted two weeks, during which teachers followed the steps of the IPEI framework to teach some programming concepts to students. On the other hand, students from the experimental groups received instruction using the PyLe environment during the two- to four-week PyLe experiments. The experiments concerned most of the programming concepts covered by PyLe.

Evaluation: Using the quantitative technique utilizing structured questionnaires, we proceeded in this phase to post-experiment surveys to assess PyLe's usability and the impact of IPEI on students' motivation. We evaluated PyLe's efficacy by quantitatively comparing the outcomes of paper tests (grades and time to complete the tests) conducted in the control and experimental groups. Additionally, through interviews, we further assessed instructors' acceptance of PyLe.

Communications: We communicated the study's results by publishing four articles in journals and conference proceedings. Consequently, two conference proceedings published the completed literature review on teaching and learning methods in IP modules along with the effective methodology for educational software design called IIADPIE. Moreover, two journals published papers related to the PyLe environment, non-computing features, and challenges. The paper on the EI-based teaching methodology is under review. We can find more details in the publication section of Chapter 1.

Research Design

The research design aims to offer a suitable framework for the study. It is a gateway created from strategic orientations that leads to action to meet the research aim or answer research questions. Another way to think about research design is as a guiding principle that helps organize the process of data collection and analysis. In this section, we discussed research methods, the research hypothesis, data collection techniques, data analysis techniques, instruments, ethical considerations, validity and reliability, and research limitations, respectively. Figure 3.2 below presents an overview of the research design for this research.

A. Research methods

While research methodology coordinates the whole research process, research methods are approaches utilised to address specific research questions. Accordingly, suitable research methods used in this study are described as follows:

- i. Comprehensive literature reviews:* Surveys, observational studies, and case studies are examples of comprehensive research methodologies, with data collected in either qualitative or quantitative form. Although descriptive studies do not demonstrate correlations between cause and effect, their findings can be used to direct future studies. Thus, a comprehensive research method was used to identify and analyse the existing teaching methods and tools used by other researchers to address the problem. Using the same methodology, this study reviewed existing techniques for designing educational software. The results are reported in Chapter 2.
- ii. Framework design:* With the insights gathered from existing tools and methods, this study proposed an effective framework for designing educational software. The foundation of this framework is a mixed-model strategy. We then built PyLe, a Python learning environment, on top of the suggested framework. In addition, drawing on the cognitivism movement, the research presented an explicit instruction-based paradigm for teaching IP courses to non-CS students.
- iii. Implementation:* This study implemented the PyLe environment. We achieved this by adhering to the seven stages of the effective framework for educational design, known as IIADPIE, suggested before.
- iv. Experiments:* To address RO2: student features and challenges in IP courses, a quantitative research approach was employed through the use of a questionnaire for data collection. Participants were first-year non-CS students at the NWU, South Africa, who agreed to participate in the survey. Using the same methodology, we conducted another survey of first-year computer science students at the University of Dschang, Cameroon, to identify the challenges they too encountered in IP modules. Additionally, to compare the challenges in IP modules encountered by both groups, we used the Mann-Whitney U-test. On the other hand, we tested the framework based on explicit teaching in two face-to-face IP classes, strictly adhering to the phases of the framework. We conducted each experiment over two weeks with randomly selected participants. Similarly, we used PyLe to teach two IP courses in both online and blended contexts. Each experiment took place over two to four weeks, during which we randomly selected participants and divided them into control and experimental groups. Students in the control groups received face-to-face instruction using the traditional method, whereas those in the experimental groups received training using PyLe. We conducted these two last experiments with non-computing

students, including first-year trainee teachers from the HTTC in Cameroon and first-year students from the FNAS in South Africa.

- *Evaluations:* To meet RO3 on the impact of the new teaching framework on student motivation, we conducted a post-experiment survey on students' motivation. The survey used a quantitative research design and questionnaire data collection. Students who attended all of the experiment's classes and consented to fill out the questionnaire were the survey participants. Following students' experience with two groups, the experimental and control groups, we administered a handwritten post-test to also meet RO3, which asked about the effect of PyLe on student performance. We also performed surveys with the experimental groups to assess the usability of the novel learning programming environment. The experiments and survey included students from two universities and followed a quantitative research design method. Teachers were interviewed about the potential effects of the learning environment on the instructional process.

B. Research Hypotheses

The following research hypotheses were stated in the study:

- The following are the null hypotheses that were utilized in the process of determining the significance of the equality of the difficulty levels of the responses in the two groups (i.e., the difficulty degree of any concepts is the same for both non-CS students and CS students):

$$H01: \mu_{D1} = 0, H02: \mu_{D2} = 0, H03: \mu_{D3} = 0, H04: \mu_{D4} = 0, H05: \mu_{D5} = 0, H06: \mu_{D6} = 0, \\ H07: \mu_{D7} = 0, H08: \mu_{D8} = 0, \text{ and } H09: \mu_{D9} = 0.$$

D1, D2, D3, D4, D5, D6, D7, D8, and D9 denote the following programming concepts: variables, comparison operators, sequential and conditional structures, functions and procedures or methods, recursion, arrays, and error handling.

- H_{10} : The ability to learn and master program syntax is the same for both the experimental group and control group in terms of GRADE.

$$H_{10}: \mu_{\text{GRADEEG}} = \mu_{\text{GRADECG}}$$

- H_{11} : The time spent (TT) performing the test is the same for the experimental group and control group.

$$H_{11}: \mu_{\text{TTEG}} = \mu_{\text{TTTCG}}$$

The null hypotheses H_{10} and H_{11} tested in these experiments aimed to gauge the significance of adopting the PyLe environment as a crucial learning material for improving students' ability to learn and master program syntax in the introductory Python programming courses. Additionally,

based on the test, these null hypotheses will be rejected if their respective p-values $p < 0.05$. We completed the Kolmogorov-Smirnov test on the GRADE and TT variables to test the normality of the data. The results indicated that both GRADE and TT were not normally distributed, as their p-values were less than 0.05. Consequently, we used the independent-sample Mann-Whitney U-test to test hypotheses H_{10} and H_{11} . Moreover, we carried out Pearson's correlation test to check whether there was a relationship between TT and GRADE.

C. Data Collection Techniques

This study mainly dealt with primary data collected mostly after experiments or research and was both quantitative and qualitative. Note that we only collected non-experimental data on non-computing features and challenges in IP modules. Primary data refers to the raw and unprocessed data obtained directly from the research efforts. We collected data using the following techniques: surveys, interviews, and case studies. Surveys and case studies provided quantitative data, while interviews generated qualitative data.

D. Data Analysis Techniques

This study manipulated both quantitative and qualitative data. We analysed quantitative data using descriptive statistical methods like means, standard deviation, mode, median, frequencies, etc. We used SPSS to analyse all the gathered data. On the other hand, the different purposes for which this study collected the qualitative data led to their analysis and interpretation.

E. Instrumentation

Instrumentation refers to the process of creating instruments that are suitable for collecting data in research [154]. This study used questionnaires, interviews, and post-tests to collect data. We designed three questionnaires to collect data, respectively, on student characteristics and difficulties in IP modules, student motivation, and PyLe's usability. We conducted interviews to gather instructors' opinions on PyLe's acceptability. We conducted post-tests to collect data on PyLe's effectiveness. The designed questionnaires are described as follows:

Questionnaire on the challenges faced by students in IP modules: The questionnaire was divided into six sections, each including a set of concise questions that were meticulously crafted to minimize the chances of misunderstandings. Part A provides an analysis of students' backgrounds; Part B evaluates students' preferences for learning environments in programming modules; Part C investigates students' preferences for learning materials; Part D assesses students' understanding of fundamental programming concepts; Part E explores students' overall perceptions of programming; and Part F examines the initial programming concepts taught at the

beginning of the learning process. The study employed a thorough methodology in formulating responses to the questions. We requested the participants to provide their responses using the following ordinal scale: Poor, Insufficient, Average, Good, and Very Good.

Questionnaire on students' motivation: We designed the 21 statements of the questionnaire based on the Instructional Material Motivation Survey (IMMS) [157]. As the IMMS statements have been developed based on the four components of the ARCS model, our questionnaire therefore comprises four sections corresponding to each of the ARCS model categories: attention, relevance, confidence, and satisfaction. Responses to the questions in each section were based on a 5-point Likert scale: Strongly disagree (1), Disagree (2), Fairly Disagree (3), Agree (4), and Strongly disagree (5).

Questionnaire on PyLe's efficiency: We designed the questionnaire about the usability of the PyLe environment based on several usability assessment criteria. The questionnaire, adapted from the one referenced [158], consisted of six sections with a total of 23 questions. The participants were asked to respond based on an ordinal scale with values ranging from Strongly Disagree (1) to Strongly Agree (5).

Post-test: the tasks given to the students included quizzes and problems identifying and correcting syntax errors in given programs. The quizzes aimed to enable students to predict the result of executing a given program, containing or not syntax errors, by choosing the correct answer from several propositions.

Interviews: We structured the interviews with teachers based on the following elements: observing students' growth, which is based on tangible (observable and communicative) results; determining whether the tools are beneficial to the teacher's professional development in terms of personal growth, recognition or valuation, and the significance of their work; and assessing the software's acceptability and potential integration into teachers' professional practices.

F. Ethical Consideration

Observance of ethical rules is required in research. When conducting research that involves people and has the potential to harm them, special caution must be exercised. In academic institutions, this is a common and obligatory policy and procedure. Accordingly, research ethics aims to safeguard the well-being of human subjects participating in investigations, enhance the validity of the research, and maintain the study's integrity. This study involved first-year students with low risk. However, we noted the following ethical considerations:

- *Ethical approval for the study*: The FNAS ethics committee approved the ethics application and issued a certificate of ethics before data collection from NWU students. This marks the start of the study and highlights the absence of any prerequisites before gathering data from NWU students.
- *Participants' selection*: We informed students of the study's purpose, and they were free to opt in or out at any time. We collected data from participants anonymously, without including their identities or personal information.
- *Intellectual property*: There has never been a study like this one, and all the publications that came out of it had authors who actively participated in the research.
- *Results communication*: We have made sure that this work is devoid of plagiarism and research malpractice, and we have faithfully presented the results. This thesis contains no falsified data or misrepresented results.

G. Validity and Reliability

When referring to a measurement, reliability refers to its consistency, whereas validity refers to its precision. When developing a research strategy, preparing statistical techniques, and writing up the findings, it is essential to consider the notions of reliability and validity, especially in the context of quantitative research. Therefore, to satisfy these requirements, all of the questionnaires used in this study included a limited number of questions with 'Likert-scale' responses to avoid any bias in the participants' answers. Additionally, we compute Cronbach's alpha values to ensure the reliability and validity of the findings presented in Chapter 5. The Cronbach's alpha value for PyLe's usability surveys was 0.80, and for surveys on students' motivation, it was 0.78, indicating a high degree of internal consistency in the questionnaire responses. Furthermore, we coded and analysed the data gathered using SPSS software, and descriptive statistical techniques were used to represent the results.

H. Limitations

Study limitations are features of the design or methodology that may influence the interpretation of research results. For instance, due to time constraints in this research, we were unable to conduct the experiments over an entire semester. Moreover, Furthermore, we conducted the experiments with a limited number of participants. Future research will therefore focus on these constraints to validate and improve the results of this research.

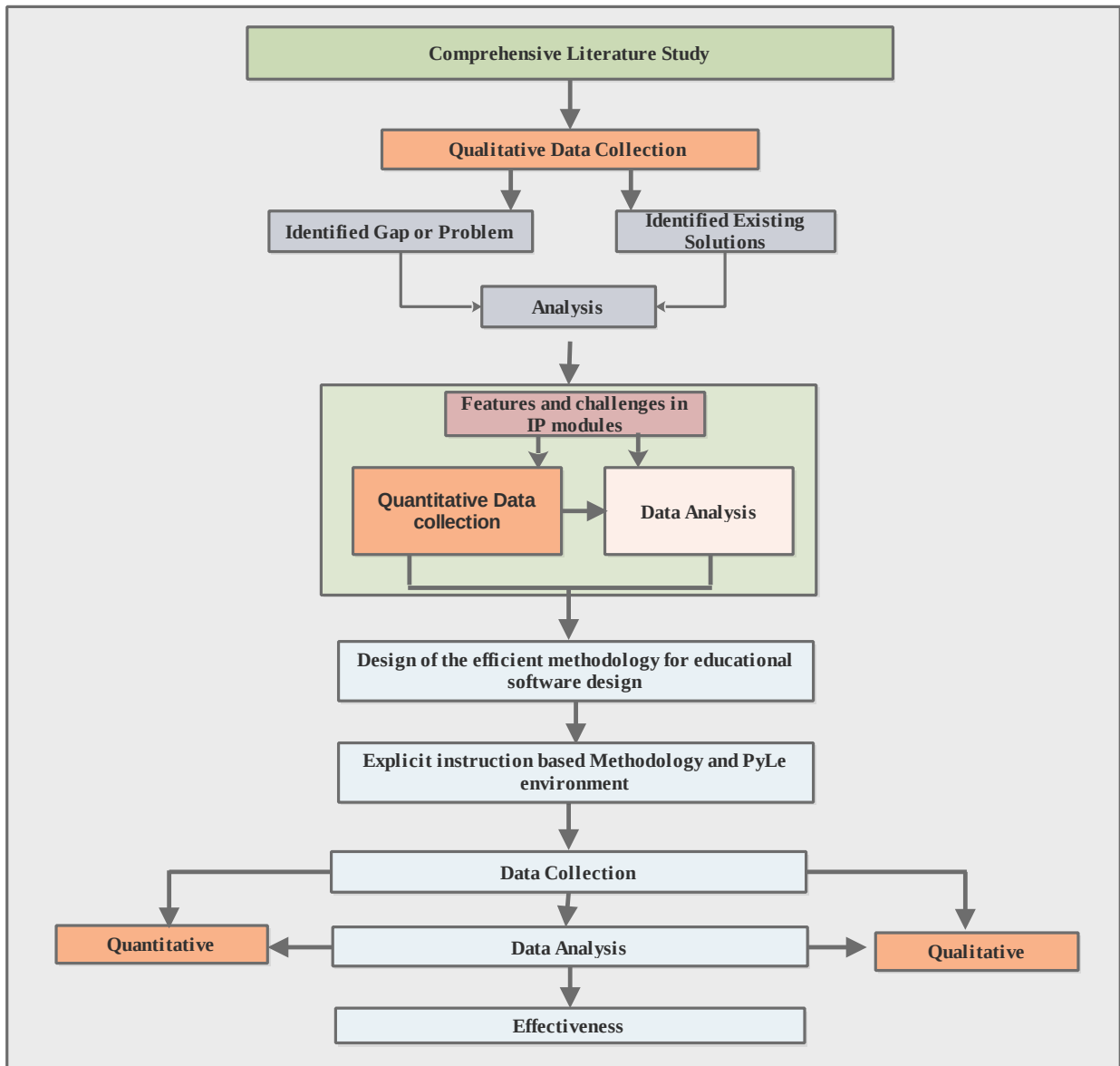


Figure 3.2: Research Design of the Study

Chapter Summary

In this chapter, we outlined the research methodology and design used in this study. We adopted the Design Science Research (DSR) approach to guide the entire research process, incorporating a mixed research methodology for data collection. We examined the research design in more detail under the following headings: research methods, research hypothesis, data collection techniques, instruments, ethical considerations, validity and reliability, and research limitations. The next chapter will discuss the artefacts or frameworks created during this study.

CHAPTER 4 FRAMEWORK DEVELOPMENT

Chapter Outline

This chapter presents the three frameworks developed in this study. The chapter is organized as follows: Section 4.2 introduces this chapter; Section 4.3 introduces and presents the cognitive-based teaching methodology for teaching computing programming, known as the IPEI or EI framework; Section 4.4 introduces a new educational software design methodology called IIADPIE; Section 4.5 introduces and presents PyLe, an interactive learning environment designed to improve students' mastery and understanding of Python programming syntax; Section 4.6 analyses the different frameworks distinctly; and Section 4.7 summarizes the chapter.

Introduction

This chapter provides answers to RQ2.2 and RQ2.3, and describes the design and development stage of the study methodology, as shown in Figure 3.1. The goal of this research is to address the issues identified in Chapters 1 and 2 regarding non-computing students' lack of motivation for computing programming modules and their difficulty mastering and understanding programming syntax. Our approach to addressing this concern is based on cognitivist theory. Indeed, as shown in Chapter 2, constructivism served as the foundation for the majority of existing interventions, which primarily targeted CS students over non-CS students [25]. However, the fact that most non-computing students lack prior programming experience, insufficient mathematics, ineffective programming strategies, and the deeply nicely organized knowledge needed to apply effective problem-solving steps may also justify the use of a cognitive-based technique [20]. Moreover, most of them, regardless of their young age, need a lot of guidance during instruction, whereas experts or CS students can work independently [20]. Therefore, we should provide direct pedagogical guidance on core concepts and strategies to non-computing students, rather than relying on their research [139]. As a result, this research suggests an explicit instruction-based method for instructing introductory programming modules (Section 4.3), thereby enhancing non-computing students' motivation. The explicit instruction (EI) approach is based on cognitivist theory, and its usefulness has been proven in reading native and foreign languages, mathematics, sciences, and history. This teaching framework begins with the instructor modelling learning strategies and providing extensive explanations, demonstrations, examples, and non-examples. Students then practice strategies in groups, on sustained practices with feedback, and finally individually. This chapter further presents PyLe (Section 4.5), an interactive educational platform

created to enhance students' proficiency and comprehension of Python programming syntax. PyLe is based on the cognitivism learning theory and includes interactive, visual, and organized content and activities that will gradually prepare students to build and understand code. PyLe offers a complete collection of learning materials and a powerful software infrastructure for Python programming basics. It lets instructors choose from relevant, accessible resources to create dynamic, module-based courses. It is noted that before PyLe, this chapter discussed IIADPIE (Section 4.4), a framework for the design of educational software (ES). IIADPIE is an acronym that stands for initial phase, instruction phase, orientation phase, analysis phase, design phase, production phase, integration and implementation phase, and evaluation phase. This framework organizes ES development into the seven stages of its acronym. This framework guided the design, development, and implementation of PyLe.

Of these three developed frameworks, two have been published: IIADPIE [81] and PyLe [21], respectively, and the IPEI framework paper has received publication acceptance in the *International Journal of Modern Education and Computer Science*.

The Proposed Explicit Instruction-based Framework for Teaching IP

This section answers the RQ2.2 defined in Chapter 1. To achieve this objective, a comprehensive review of existing literature was carried out, as detailed in Chapter 2. This process resulted in the discovery of shortcomings in the majority of current instructional programming teaching approaches. In addition, surveys were carried out on non-computing students in two separate universities to identify the challenges they encounter in IP courses which can be found in [20], and in Chapter 5. Based on the outputs of these two first phases, we outlined IPEI requirements. These requirements aided in the design of the IPEI framework. This section therefore presents and discusses the proposed teaching methodology, which is a cognitive approach based on the EI of the cognitivist theory [54].

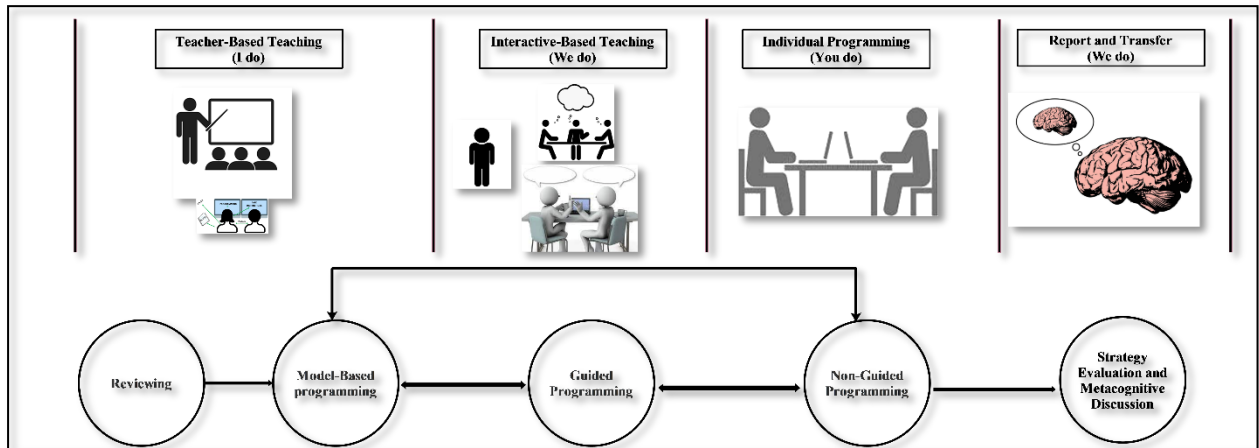


Figure 4.1: IPEI Framework

The proposed EI method for IP modules is known as IPEI, which has iterative phases.

Figure 4.1 shows the phases including reviewing, model-based programming teaching, guided programming, non-guided programming, and report and transfer in our context. These are discussed as follows:

1. *Reviewing*: At this stage, the teacher will address Why... are students learning this. When... will it be relevant to them? And where... will students use these skills? This stage also includes a review of homework and relevant previous learning, as well as a review of prerequisite skills and knowledge.
2. *Model-based programming teaching*: Modelling is a method of training self-management strategies for understanding, where the teacher acts as a guide, explaining his or her behaviour when faced with a comprehension problem. The teacher explains the source of his or her difficulty and the strategy used to overcome the problem. The teacher also models effective attention and performance strategies for students. Thus, this phase is more based on strategies deployed and steps followed to solve a programming task, which is one of the main focuses when preparing the class lesson. The teacher offers through demonstration an appropriate set of programming examples and non-examples to draw the line as to when to apply or not a skill, strategy, concept, or rule [55]. The use of non-examples, in this case, might help students reduce inappropriate uses of skills, and frustration faced when starting programming. During this phase, the teacher should introduce, whenever possible, misconceptions, how he or she or programming experts proceed to understand programming, and how they overcome the difficulties they have faced as well. Examples will progressively move from daily life-based to mathematics-based problems to ease

students' integration, increase students' motivation, and promote knowledge acquisition. During this phase, students must listen, not interrupt, and save questions until an appropriate time. On the other hand, the teacher is advised to involve students as much as possible through questioning.

3. *Guided programming*: In this phase, the teacher supports the acquisition of the strategy through explanation, teaching, and modelling. In addition, following the principle of scaffolding, the teacher encourages independent and autonomous use of the strategies introduced and developed during the modelling phase. Students better learn strategies when they experience the role of the teacher and have to justify the importance of using the strategies. These practical classes are assisted by the teacher and allow students to correct errors as they proceed through immediate feedback. This phase aims to verify students' comprehension and immediately correct their misconceptions. This in turn will promote student self-confidence, satisfaction, and collaborative programming. The activities are organized around the use of collaborative programming techniques described as follows:

- (i) The teacher gives a small programming problem to the students.
- (ii) The students individually take a few minutes to think about the solutions.
- (iii) Students are organized into groups of two or more, as they sit in the classroom, discuss and share their thoughts, and come up with a group solution. To improve students' logical programming thinking, the paper-and-pencil strategy (PPS) is required. The PPS is a handwriting technique that helps students solve problems as computers do and guides them to illustrate their solutions in a diagram, table, symbols, etc. [159]. On the other hand, during practical sessions, students sit in pairs in front of one computer and implement their elaborated solutions. While one types the code, the second observes, corrects, and improves the code quality. This situation has to be interchanged for the next activity or exercise.
- (iv) The teacher approves or redirects the groups' proposals. This interaction will help teachers identify students' misconceptions which will later ease the preparation of concept inventories. Again, these misconceptions will also direct the elaboration of exercises that are closer to the student's zone of proximal development. These practices have to be continued until students master the strategies. But, if, after many attempts, the strategies prove futile, the teacher should go back to the modelling phase or move to the next phase.

4. *Non-guided programming*: This is an extension of the guiding practice phase, which is designed to promote memory retention and personal skill development by allowing students to develop their understanding to the highest possible level of learning mastery. These are programming activities that will allow students to work independently, i.e. without any support

from other students and the teacher. Like the guided programming phase, these activities must be continued until students individually master the strategies. But if not mastered after several attempts, the teacher should go back to the guided programming or the modelling phase depending on his observation, or otherwise simply move to the next step.

5. *Report and transfer*: This is an evaluation stage where the teacher invites students to discuss what they have learned. This discussion enables students to check their understanding of the strategy and its relevance in a given context. This last element should enable students to discuss possible transfers of the strategy to other contexts or disciplines.

The Proposed Framework for Efficient Educational Software Development

This section addresses RQ2.3 of the study. To achieve this, this section first justifies why the need for a new framework remains a persistent concern, and then presents the details and process of the new framework.

4.1.1 Justification

Although producing computer software has always been a difficult endeavour, developing educational software (ES) systems is equally difficult; yet, the two are not synonymous. Unlike normal software, which develops to address specific business problems, ES primarily aims to support instructional activities and meet the needs of its diverse users across various educational levels and types [160]. Furthermore, the analysis phase of ES design takes into account pedagogical factors such as learning objectives, instructional design, remediation, and feedback systems when developing the product [160], [161], whereas functional requirements remain the emphasis of traditional software design. Furthermore, ES designers must have a strong understanding of teaching, learners, and classrooms [161], whereas software engineers' skills are more closely tied to engineering as a broad field of knowledge than CS [162]. Furthermore, ES frequently contains tools for monitoring learning progress, tracking student achievement, and providing feedback and assessment [6], as well as capabilities for assessing and evaluating student performance. In this situation, designers must consider how the software will gather and report data on student progress, as well as how educators will use this data to improve learning outcomes [163], [164].

This research subsequently uncovered several publications that utilized a variety of proposal methodologies for ESD. As an illustration, Hinostraza et al. [163] emphasized the importance of

incorporating the various aspects of the software's context of use and the implications for teachers throughout the development process. They proposed three primary strategies for ESD: designing cognitive tools, creating professional tools for teaching, and providing teaching materials and resources. Costa et al. [165] introduced a hybrid user-centred development methodology (HUCDM) for ESD. HUCDM is a synthesis of agile concepts and the user-centred design technique. Lage et al. [166] further developed a prototype by gradually improving the software's life cycle using a cognitive-constructivist approach. They also proposed a method for designing instructional software. The authors put out a sequence of 23 activities for instructional software. In addition, Johnson and Schleyer [167] suggested a method for creating dental instructional software that combines essential pedagogical approaches with life-cycle software engineering. In addition, they delineated some crucial characteristics of high-quality ES, as well as the vital qualifications of the ESD team. In a previous study, Beale & Sharples [168] laid out significant tasks for the development of ES. In a similar vein, Filho [169] devised and implemented the Praxis process, which encompasses a set of prerequisites for an ESD procedure. Sriram [160] identified common requirements among educational users during the analysis stage of the software development life cycle. Teachers and students independently conduct the requirements analysis for ES in this scenario, given their diverse perspectives and viewpoints on daily operations.

Some ESD-related studies are highlighted above. While there has been little research on this topic, the majority of it focuses on the implications for teachers and students in the design process. However, these approaches did not provide users with enough information or guidance. Teachers and students aimed at reducing time waste as well as improving and generalizing the strategy and software quality. Furthermore, the literature overlooked some of its most important material and neglected newly created learning and teaching strategies. Again, current and widely used methodologies and practices for designing and producing software, such as agile methods and IDMs, received little attention. As a result, for this study, we advocated integrating agile methodologies and IDMs into ESD design to ensure that educational software serves the needs of both students and teachers.

4.1.2 IIADPIE Process

This subsection outlines the suggested framework for designing efficient ES. This framework is a blended approach that draws on methods and principles from agile methodologies such as DSDM and Scrum, as well as instructional design models such as ASSURE and ADDIE. The proposed ES development framework, called IIADPIE, consists of seven main phases: initial, instructional, orientation, analysis, design, production, integration and implementation, and

evaluation phases, as shown in Figure 4.2. These phases' design ensures the final product's effectiveness. The execution of these phases, except for the initial phase, involves an iterative process of refining and improving the idea based on feedback from previous versions.

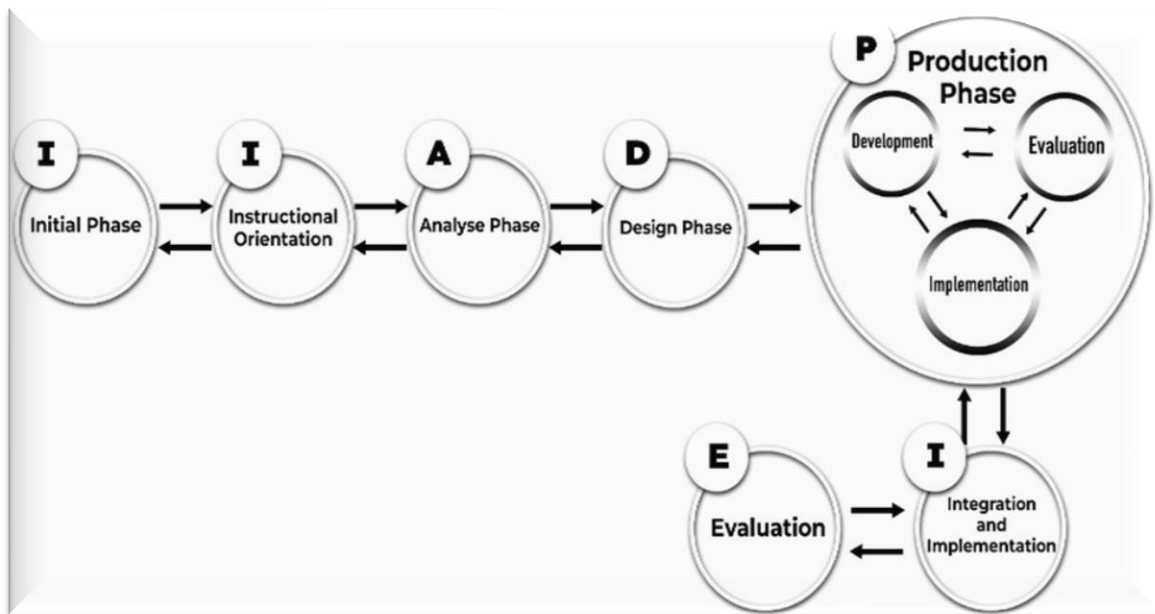


Figure 4.2: IIADPIE Process [81]

a Initial Phase

This stage launches the operation by organizing the initial meeting, establishing the project scope, and identifying the product owner, the Scrum master, and the development team. Furthermore, we establish the assigned financial resources, the project's timeframe, and the specific technical specifications. We undertake a feasibility study to evaluate the project's technical and economic feasibility, identify potential hazards, and determine the duration of each phase. At the end of this phase, we release the feasibility report.

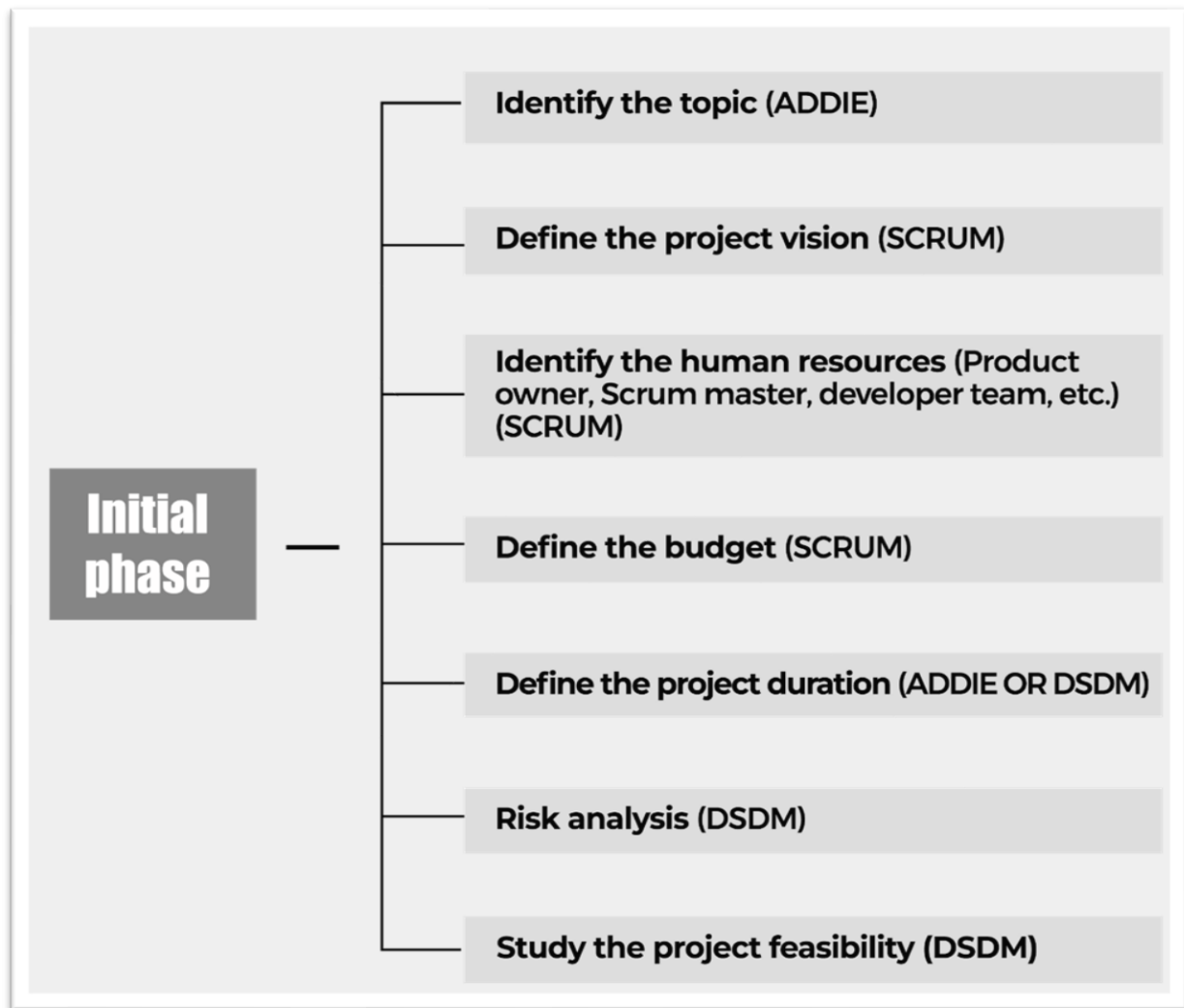


Figure 4.3: Initial Phase of IIADPIE's Process.

b Instructional and Orientation Phase

After obtaining a satisfactory feasibility assessment, the educational orientation stage begins. This phase involves achieving six (06) primary objectives:

1. *Specify the target population*: At this stage, the study specifies the population involved without further precision.
2. *Specify the educational situation*: Make a description of the surroundings where the ES is going to be utilized. This includes designing cognitive materials, professional materials for instruction, and instructional tool suppliers [163].
3. *Specify the overall objective*: Make a note of the primary purpose of the training.

4. *Specify the main learning and instructional techniques*: The methods chosen at this stage will guide the overall concept of the ES, encompassing its activities, functions, and content. Instructional, learner-directed, collaborative, individual-directed, discovery-based, or teacher-directed are some of the several directions that they might take[13], [168].

- *Teacher-directed*: In a classroom setting, teachers should guide students' use of the ES.
- *Learner-directed*: Students should use the ES without direct teacher guidance.
- *Collaborative*: The ES aims to assist pairs or groups of students.
- *Individual*: A single student can work independently with the ES.
- *Discovery*: The ES enables learners to discover knowledge and delve into concepts.
- *Instructional*: The ES provides direct instruction on a knowledge or skill element.

5. *Identify the crucial aspect of the topic*: The ES does not necessarily need to cover all aspects of the given topic. Here, it is important to clarify which crucial aspect of the topic the ES will concentrate on.

6. *Enumerate the teaching resources*: This includes items that can help or guide the productive creation of content. This encompasses books, course outlines, web links, and other related materials.

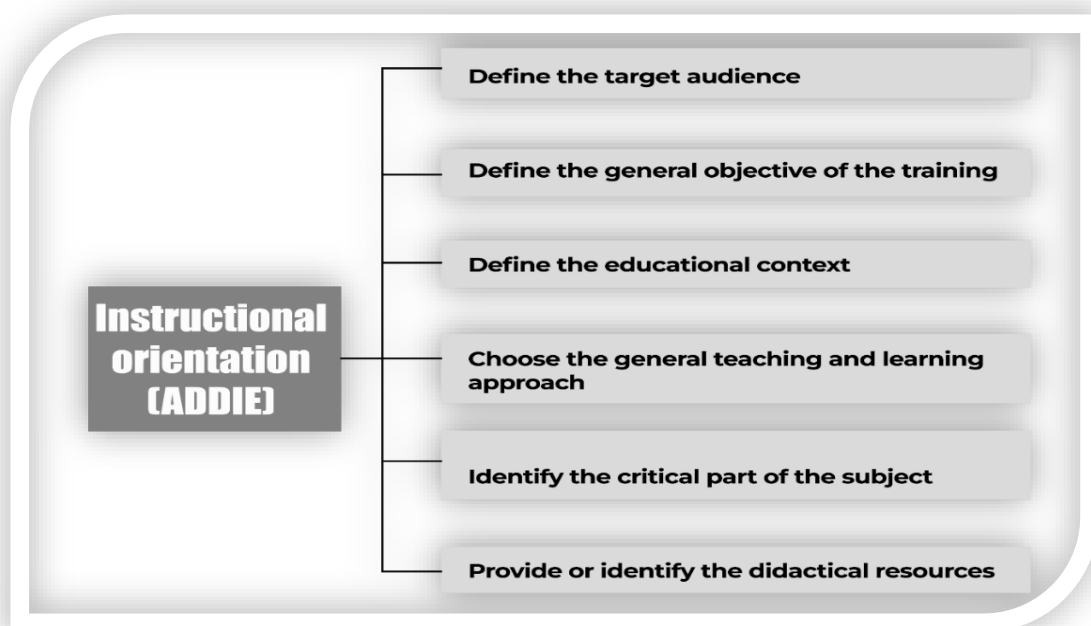


Figure 4.4: Instructional and Orientation Phase of IIADPIE's Process.

c Analysis Phase

The scrum master must organize this stage's many components into a unified project backlog. After that, depending on how complicated the stories are, the scrum master organizes them into an iterative sprint cycle that usually lasts two to four weeks.

1. *Define the features of the target audience:* Students and teachers are the end users of educational software. A sample of the target audience should therefore be surveyed to find out what difficulties students and teachers encounter in using the modules. The use of questionnaires, interviews, and student grade records to survey and analyse the features and challenges of the target audience.

2. *Define learner profiles:* Learner profiles, whether computerized or not, contain information about learners' abilities, knowledge, preferences, learning methods, ES interactions, and instructional activities [170], [171]. Determining the learner's profile is a crucial first step in gathering and storing pertinent data regarding the learners' activities, tailoring the activity and content to the individual learner, and monitoring the learner's advancement via the system. This is especially important given the variety of student backgrounds attending various training modules today. Personalization, whether automatic or manual, seems to improve the system's evolution, efficiency, and usefulness throughout development. A student has many profiles in the system at any one moment, depending on how they vary and grow in their comprehension of a subject or instructional scenario. The designer must build a learner profile around certain standards and norms if they believe the system's data can interact with other systems. These consist of IMS LIP (IMS Learner Information Package), IMS RDECEO (IMS Reusable Definition of Competency or Educational Objective), and PAPI (Public and Private Information for Learners). For a thorough explanation of these criteria, see Reference [172]. It is up to the designers to mix and match those requirements to create a learner profile that works for them.

3. *Define user requirements:* At this stage, the goals are user identification and formal system requirements. Concerning the formalization of user needs, this blended framework recommends the use case diagram be used.

4. *Investigation of learning and instruction strategies in the area:* This phase entails creating a comprehensive inventory of the capabilities and limitations of the system. Furthermore, these limits encompass factors such as efficiency, protection, selection of technologies, specific client and server platforms, implementation and operational limitations, operating systems, and access modalities.

5. *Content storyboarding*: The objective of this step is to dissect the content into smaller, distinct, and autonomous components. We divide the training's overall purpose into multiple specific objectives to enhance the organization of the material. We divide the training content into smaller training units known as pedagogical grains, which align with each unique aim. On the other hand, the goal should meet the SMART criteria, which stands for Specific, Measurable, Achievable, Relevant, and Time-Bound, and should be based on Bloom's taxonomy [173].

6. *Study of the teaching and learning methods in the field*: Over the years, many fields have conducted in-depth studies to improve their instruction. This point suggests using these key findings for all types of pedagogy as they apply, rather than focusing solely on conventional education and instruction approaches. For example, the inductive pedagogical approach proved effective for teaching the notion of variables in IP courses [74].

7. *Study of types of content and activities*: This stage aims to delineate all the tasks and information necessary to achieve the primary objective of the training. The categories of material can include, among other things, text, documents (such as Microsoft Word, Adobe PDF, and Slide), games, visualizations, animations, simulations, pictures, and videos. Multiple-choice questions, binary questions, concept inventories, games, drill-and-drag questions, and other forms of activities might be among the types of activities that are available.

8. *Generate the product backlog*: In this stage, we generate user stories to describe the lessons associated with the previously established pedagogical grains.

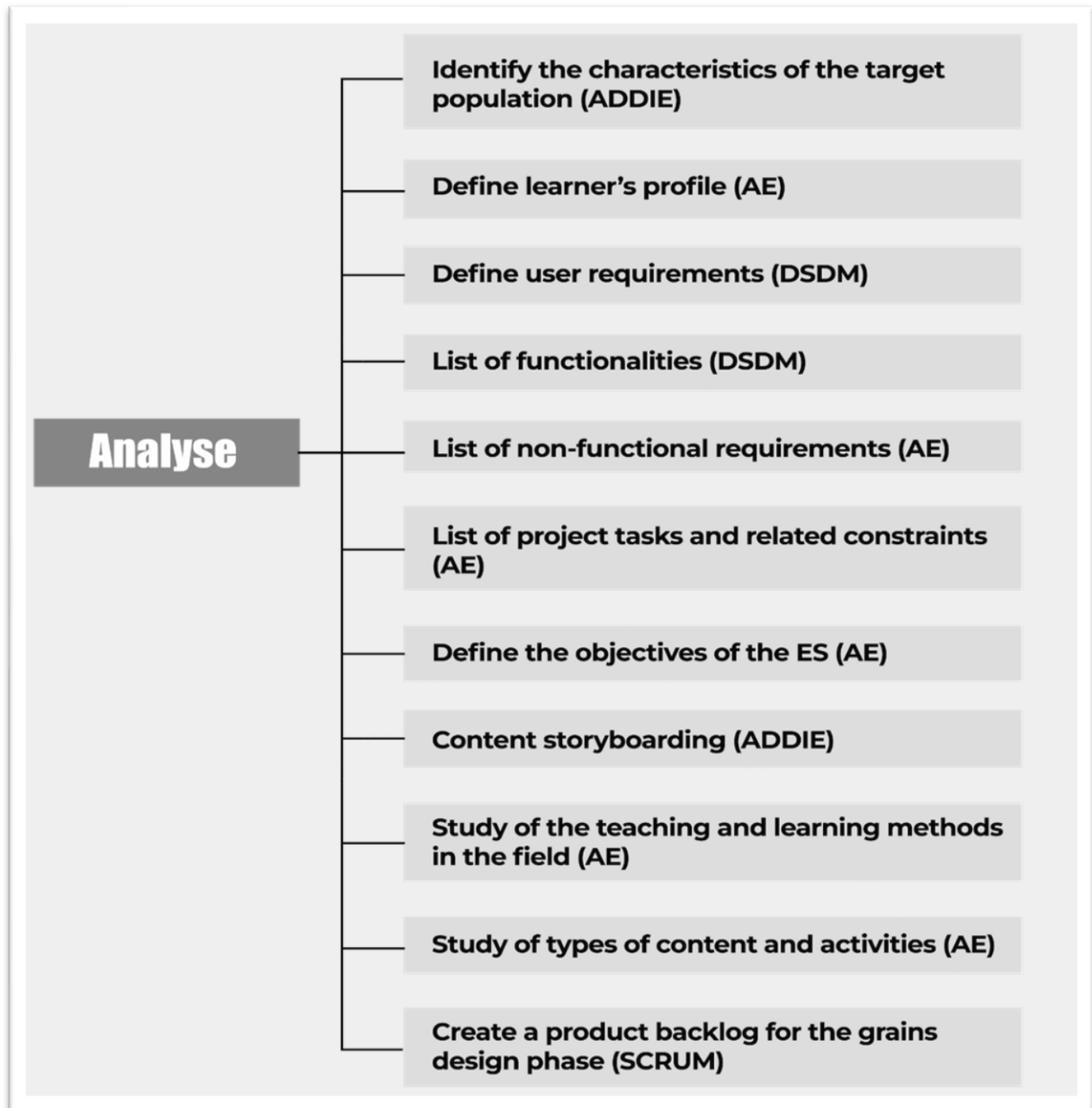


Figure 4.5: Analysis Phase of IIADPIE's Process.

d Design Phase

This stage involves the comprehensive conceptual development of the product before its actualization. It contains several different key components, as described in Figure 4.6. The first point provides ergonomic information; the second point operates within the sprint cycle process; and the remaining points proceed iteratively with feedback at the conclusion.

1. *Ergonomic design*: The body of knowledge known as ergonomics guides the design of systems, machines, or tools to ensure maximum comfort, safety, and efficiency for all users.

Ergonomics makes systems simple to use and manipulate for as many people as possible by increasing their usability and utility [164], [174]. In this context, system usability refers to the degree to which a specific user can utilize a product to achieve set goals with effectiveness, efficiency, and satisfaction in a given context [164]. Therefore, this phase serves as a reminder to developers of the ergonomic principles they must take into account when designing and creating interfaces. This study [164] identified 8 key points for assessing human-machine applications:

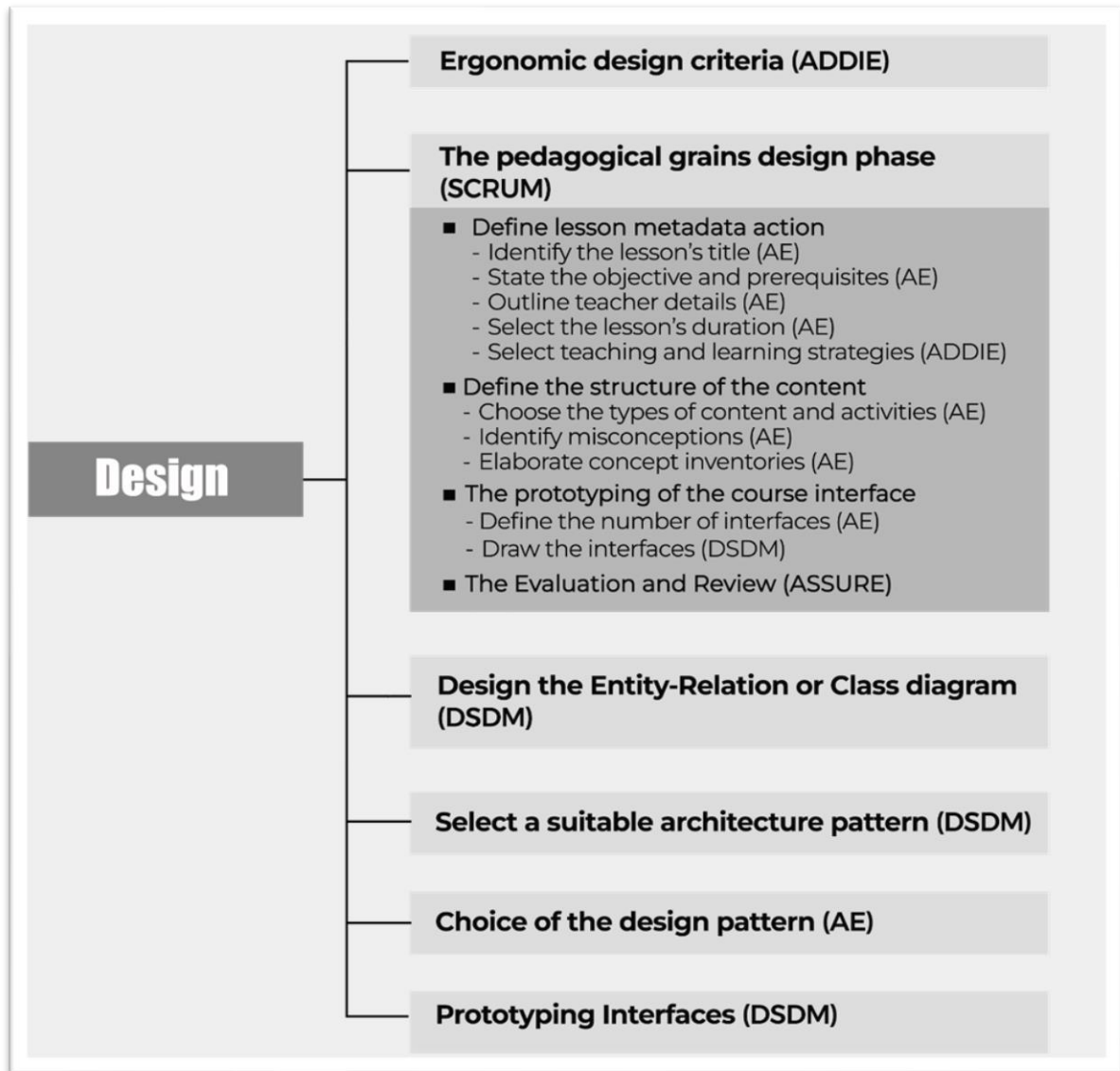
- *Guidance*: This is the set of means implemented to counsel, direct, inform, and guide the user during his interactions with the computer (alarms, messages, labels, etc.).
- *Workload*: This refers to all of the interface's elements that contribute to reducing the user's perspective load and improving the dialogue's efficiency.
- *Explicit control*: It refers to both the user's control over the interface or software and the explicit nature of his actions.
- *Adaptability*: The system can react according to the context and the user's needs and preferences.
- *Error management*: It concerns all means to avoid or reduce errors and correct them when they occur.
- *Homogeneity/consistency*: This refers to the process of maintaining the same interface objects (code, procedures, names, etc.) for identical contexts and different objects for different contexts.
- *The meaning of codes*: This is meant to indicate the level of adequacy between the objects, displayed or requested information, and their referent.
- *Compatibility*: It refers to the agreement that can exist between the characteristics of the user (memory, perception, habit, etc.) and the organization of the outputs, inputs, and dialogue.

2. *Pedagogical grain design stage*: The smallest element that breaks down a course or training program is known as a pedagogical grain. It encompasses all the pedagogical information and activities necessary to achieve a specific educational goal. At this stage, we use sprints to identify and organize the key components of each pedagogical approach. Each sprint involves the following actions over 1–2 days: Define lesson metadata, structure content, lesson interface prototyping, evaluation, and review. Figure 4.6 provides additional information on all the actions taken at this point. As the primary users of the system, the teacher and students must complete these actions.

3. *Create the entity-relationship (E/R) or class diagram*: This action, when applicable, determines the numerous objects or entities, the connections between them, and constraints, as well as creating the E/R diagram or class diagram for the entire system.

4. *Choose the appropriate architectural scheme*: It outlines the various components of the application system about organization, interdependence, and interconnections. It focuses on the key decisions that will ensure the software's excellent quality. This study [175] provides additional information on reusable architectural models.

5. *Prototyping interfaces*: Each key interface, including the connection and main pages, the student resource's view, the learner activities and assignment's view, the instructor and learners tracking views, and so forth, requires the creation of a prototype. Upon the completion of the design phase, we update the order book to set the stage for development. This phase concludes with the preparation of all user stories required for system development.



* AE: Added Element

Figure 4.6: Design Phase of IIADPIE's Process.

e Production Phase

The product backlog plan will guide the execution of this stage through a series of sprints lasting 1 and 2 weeks. We will proceed with the sprint in three stages: 1) Pedagogical Grain Development, where we fully develop content and activities; 2) Execution: Based on the user's narrative, we bring the design to life through the implementation process, which involves the use of a programming language; and 3) Assessment: A unit test is run here to identify whether or not the end product has accomplished its primary goal. In addition, we will continue to hold daily meetings to bring the backlog up to date after each sprint.

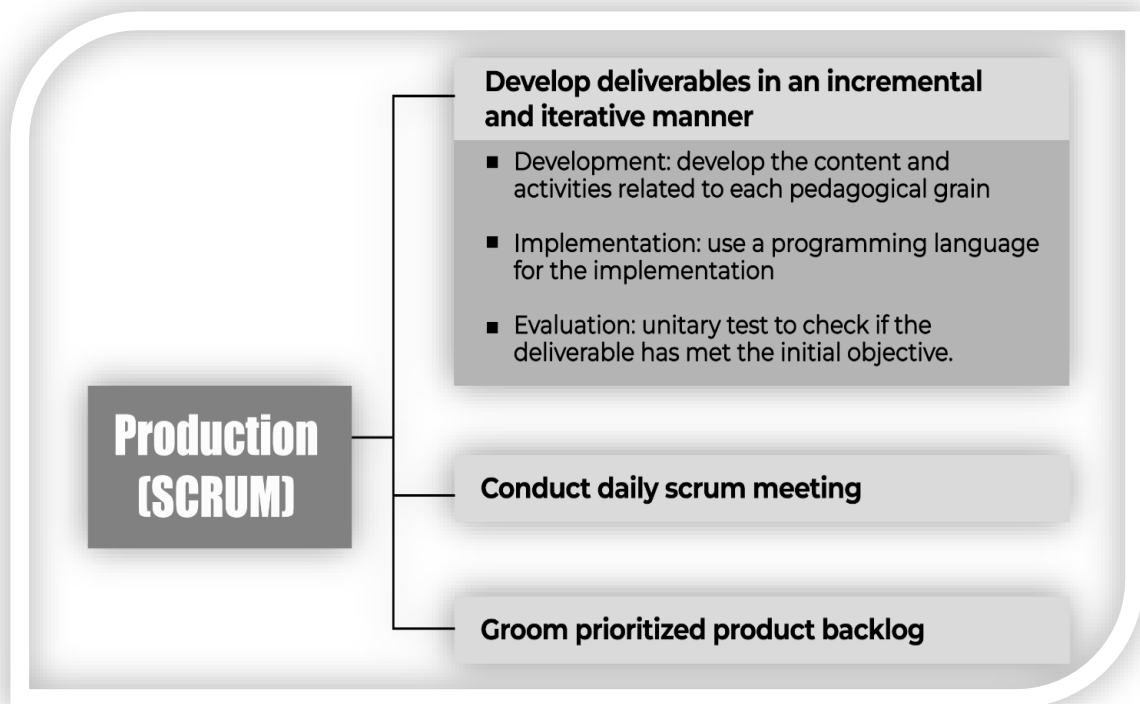


Figure 4.7: Production Phase of IIADPIE's Process.

f Integration and Implementation Phase

This stage consolidates numerous outputs and implements the final product in an actual educational environment for learning and teaching purposes. Furthermore, the user guide is written, instructor coaching is arranged, and students are equipped to use the product.

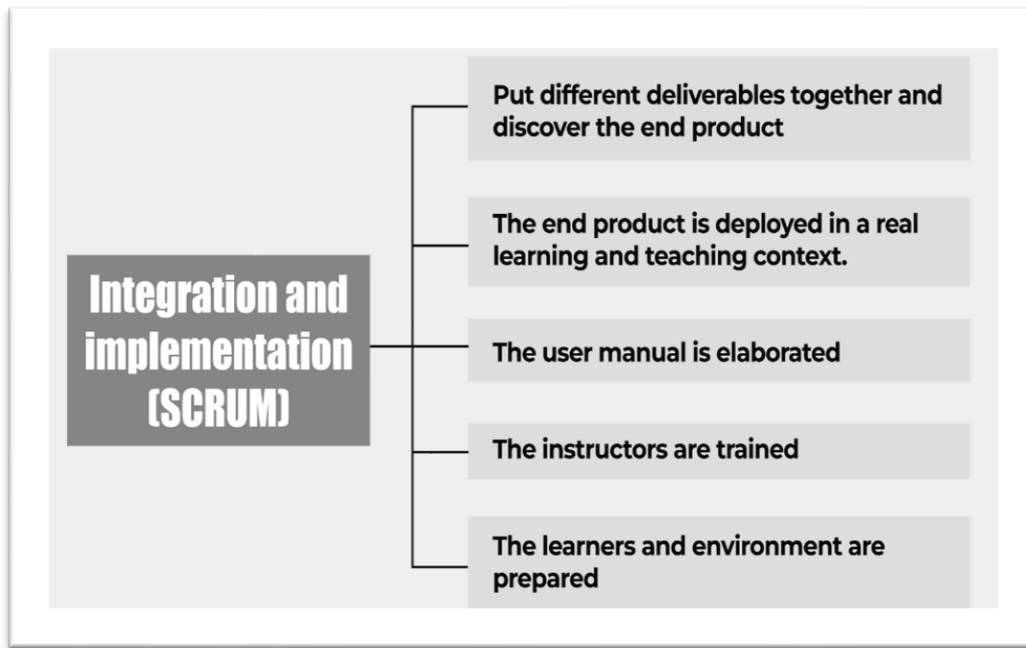


Figure 4.8: Integration and Implementation Phase of IIADPIE's Process

g Evaluation Phase

This enables the verification of whether the final product aligns with the overall software goal established in the second stage of this procedure. This study [163] suggested three methodologies for evaluating educational software: experiments, check-listing techniques, and mixed methodologies (quantitative and qualitative).

By combining the two most important parts of agile scrum—DSDM and IDM—it is possible to greatly improve the development of inclusive ES that meets everyone's needs and helps students learn effectively.

PyLe: The Proposed Interactive Learning Environment

This section also serves as an answer to RQ2.3 of this study. The design and development of the proposed interactive learning followed the same process as the EI framework in Section 4.3: a literature review to identify the majority of current instructional programming teaching approaches and surveys from non-computing students in two separate universities to identify the challenges they encounter in IP. The findings from this phase guided the formulation of PyLe's requirements. The proposed interactive learning environment is called PyLe. PyLe was designed and implemented following the various stages of the educational software development framework

known as IIADPIE previously discussed. PyLe is grounded in the principles of cognitivism, which emphasizes the development of the learner's memory. The primary objective of PyLe is to assist students in acquiring a thorough understanding of the syntax of basic programming concepts in Python. Therefore, we concentrated primarily on syntax comprehension because we wanted to facilitate the learning process and improve the students' programming abilities. The PyLe source code is available at <https://gitlab.com/kmbiada/pyle>. The processes involved are discussed as follows:

4.1.3 Design Goals

PyLe aimed to provide a comprehensive set of learning materials and a robust software infrastructure to support courses covering the fundamental concepts of Python programming. It empowers teachers to select from available and relevant materials to incorporate into their courses, making the content dynamic and organized into modules. These modules, managed by instructors, are further organized into a series of lessons. Each lesson comprises a set of sections or parts and includes resources and activities. These resources and activities are presented straightforwardly to enhance learning effectiveness. Moreover, activities are specifically designed to engage and motivate students, while teachers can assign homework to assess the quality of student learning. Thus, PyLe materials meet the following specifications and features:

- i) *Platform independence* - can be accessed from various platforms and devices.
- ii) *Non-proprietary format* - materials are not restricted to a specific proprietary format.
- iii) *Scalability and reusability* - modules can be scaled and reused as needed.
- iv) *Dynamic content* - content is not static and can adapt to different learning scenarios.
- v) *Interactive Simulation* - facilitates in-depth understanding.
- vi) *Interactive and automated feedback evaluation activities* - activities provide instant feedback to students.
- vii) *Storage of student exercise attempts and grades* - allows tracking of students' progress.
- viii) *Learner's profile building* - PyLe tracks students' progress and builds learner's profiles.
- ix) *Feedback mechanism* - provides the instructor with the ability to give feedback to students and vice versa.
- x) *Student progress management and display* - individual student reports and progress are managed and displayed.
- xi) *Time-bound lessons* - lessons must be completed within a given timeframe, otherwise, students will not be able to move on to the activities section of the lesson.

These features collectively make PyLe a comprehensive and effective learning environment for Python programming.

4.1.4 PyLe Infrastructure

PyLe is a web application that operates on a client-server architecture implemented using the PHP CodeIgniter Framework, with a MySQL database serving as the data storage medium. Additionally, PyLe employs the Model View Controller design pattern, which separates the application logic into three interconnected components, enhancing its manageability and scalability. The overall behaviour of the system, as depicted in Figure 4.9, is governed by the interactions of different users. Each client request is processed through the application programming interface (API). The system initially verifies the authenticity of the user, following which it directs the processing of the request to the system core. Depending on the nature of the user's request, the system kernel routes it to the appropriate module. This module then processes the request and generates the corresponding response. This response can sometimes necessitate access to the database or an external service. While the system predominantly uses the API for most requests, when a user is logged into their administration panel, they are granted a session. This session permits them to access certain modules directly for the duration of the session. Figure 4.10 summarizes the previously described sign-in process. This architecture ensures a secure, efficient, and user-friendly interaction with the PyLe environment. The components of the PyLe architecture as shown in Figure 4.9 are as follows:

1. *Administrative tools*: This provides teachers and learners with essential workspace management functions, allowing them to interact with the system. For instance:
 - **Module management**: Teachers can create courses from scratch or use pre-existing modules. The default course module is already available in the environment.
 - **Lesson updates**: Teachers can add and update lessons, incorporating various resources and activities.
 - **Assignments**: Teachers can add, update, grade, and export assignment grades. Learners can complete assignments and view their grades.
 - **Personalization**: Learners can update profile information, including learning style, colour preferences, and prior programming experience.
2. *Client layer*: This enables users (teachers and learners) to access PyLe via any mobile device or computer.

3. *PyLe server*: At the core of the system, this handles security, authentication, authorization, routing, and interaction with other system elements. It constructs different learner profiles based on their interactions with the system.
4. *Database*: This organizes data storage within the database.
5. *Third-party*: The content delivery or distribution network (CDN) facilitates rapid resource transfer for loading internet content. It includes HTML pages, JavaScript files, style sheets, images, and videos. In PyLe, we utilize Bootstrap, jQuery, and the TINYMCE editor.

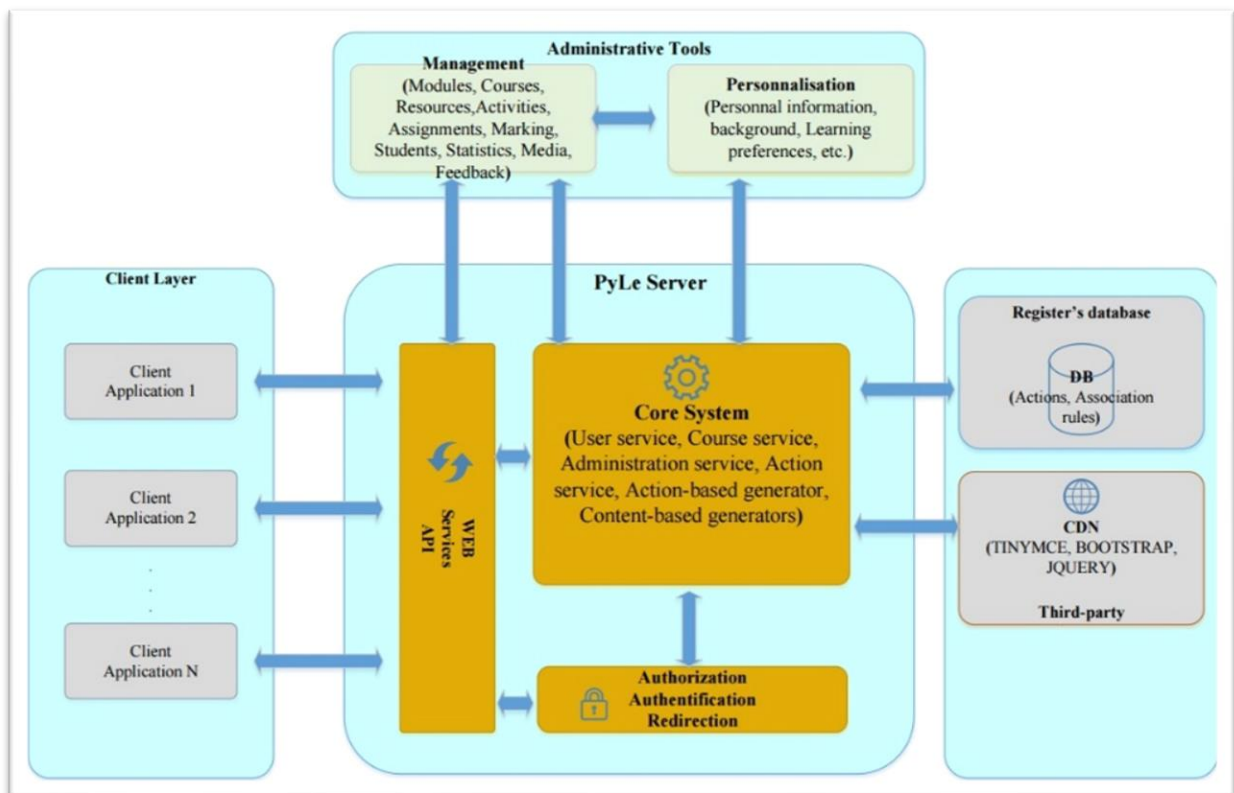


Figure 4.9: PyLe Architecture [21]

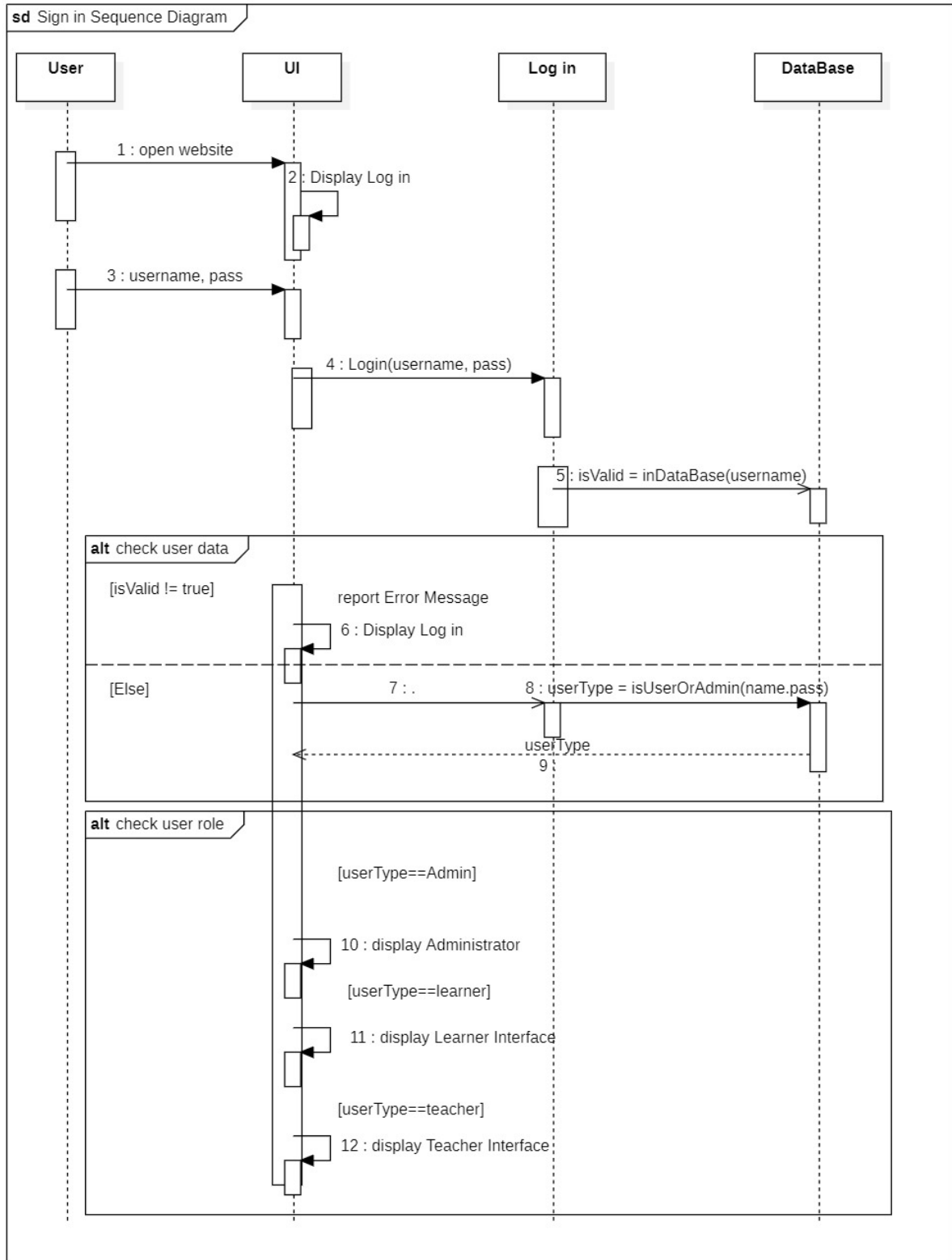


Figure 4.10: Sign-in Sequence Diagram

4.1.5 User Requirements

Figure 4.11 depicts all the actions that can be performed by a learner. In some cases, instructor intervention is needed. For instance, the instructor must create the learner's account before they can log in. An instructor must add the learner to a class before he can view the list of modules. Figure 4.11 also shows that some actions are extensions of others. For example, viewing module details is an extension of the viewing module action. Changing your password is an extension of the edit your profile action.

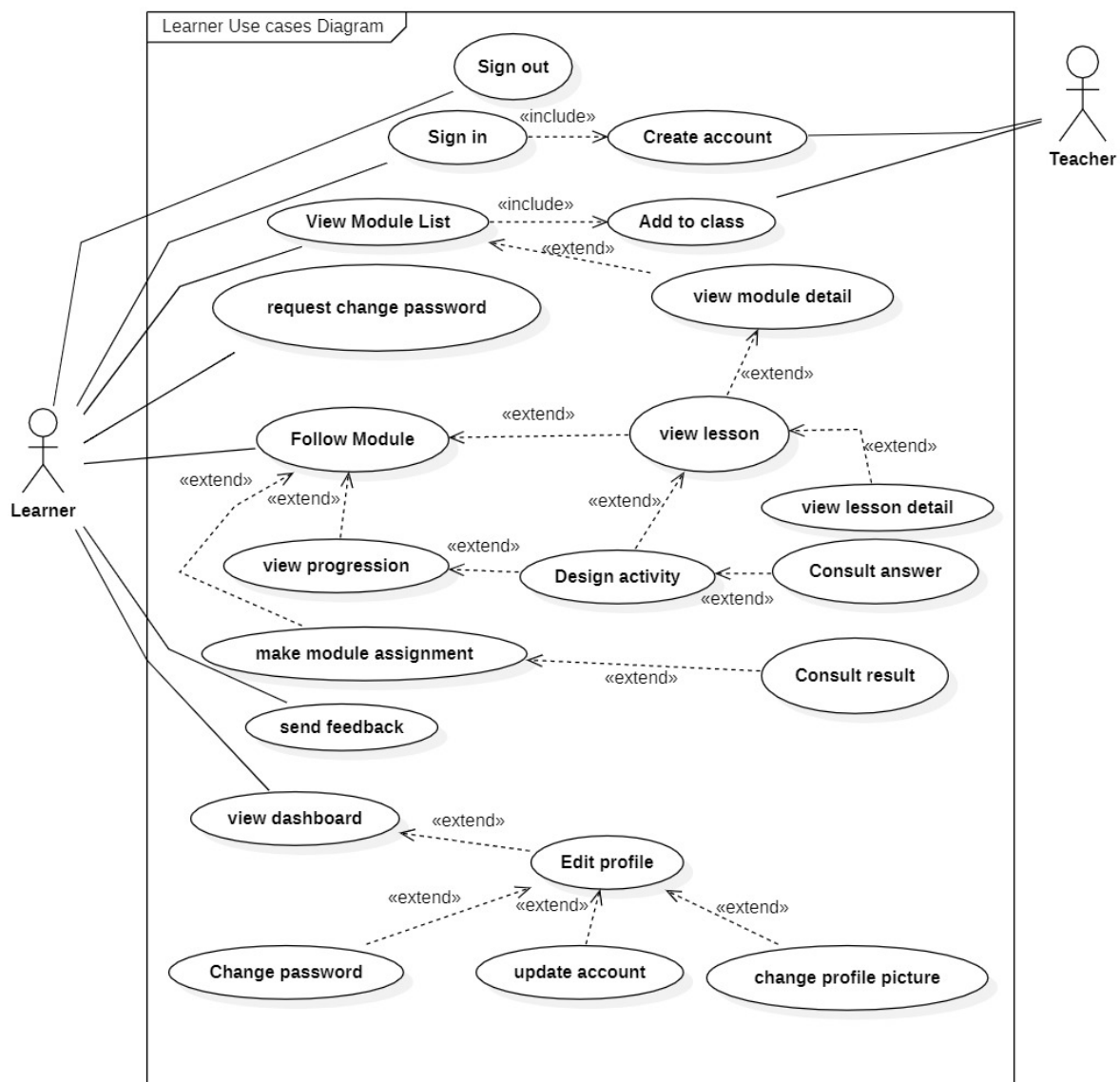


Figure 4.11: Learner Use Cases Diagram

4.1.6 Class Diagram

Figure 4.12 provides a comprehensive overview of the development system's overarching structure. The fundamental properties of each class in the system are visible. Additionally, we observe how the various classes interact.

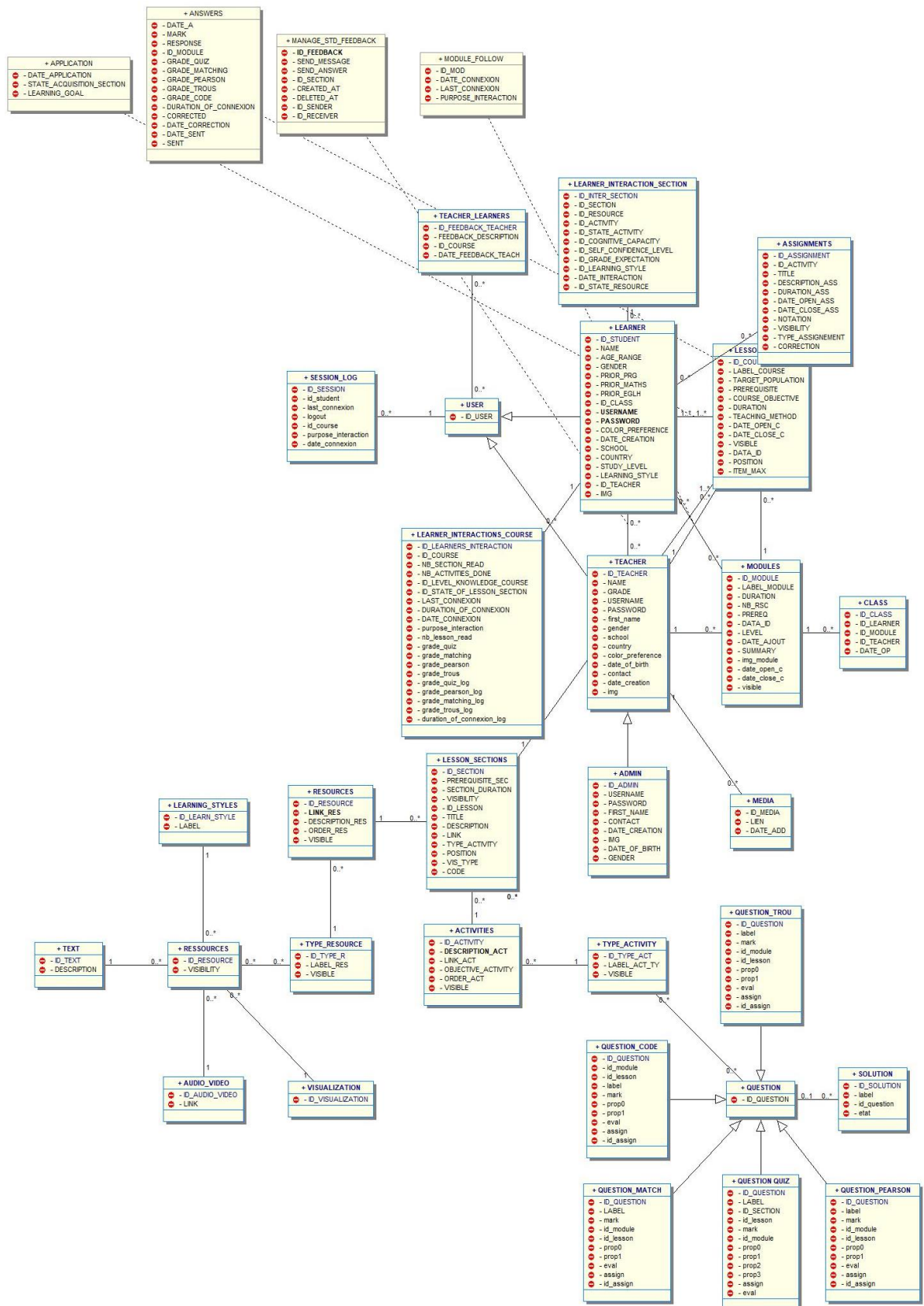


Figure 4.12: Class Diagram

4.1.7 Student Profile in PyLe

In PyLe, profiles were constructed by amalgamating the following norms: PAPI, IMS LIP, and IMS RDECEO [81]. Learners' profiles were created based on the interactions of the students with the system, as shown in Table 4.1. In essence, each student can have multiple profiles within the system. These diverse profiles were utilized to compile the student's progress reports. This approach ensures a comprehensive understanding of each student's learning journey, facilitating personalized instruction and feedback.

Table 4.1: Student Profile in PyLe [21]

Categories	Elements	Type/Values	Use
Personnal Information	Nom	String	Identification
	Gender	String	Identification
	Age	Scale	Identification
	Country	String	Identification
	School/University	String	Identification
	Study level	{Undergraduate, PostGraduate, Pupils, Other,}	Identification
	Prior programming experiences	{Yes, No}	Initialization
	Prior mathematics experiences	{Yes, No}	Initialization
	Prior English experiences	{Yes, No}	Initialization
	UserName/ Password/ Email	String	Identification
Learner's Features	Cognitive capacity	{Slow/Fast working memory, Poor/Good memory person}	Initialization (Time assigned to each lesson might depend)
	Learning styles	{Visual, Auditory and Reading/Writing}	Initialization; Choice of content type and activities
	Learning goals	{Acquire skills, Train for Exams, Obtain a good Grade, other,}	Initialization; Choice of concepts and examples
	Self-confidence level	{High, Medium, Low}	Inference of learner's understanding
	Grade expectation	{A, B, C, A+, etc.}	Check if the desire is in adequation of the time dedicated to learning
	Learner color preference	{Set of defined colors}	Personalization of the learner's environment
Learning states	State of Section/lesson	{Acquired, To be Acquired, To be revised, Not to be suggested}	Statistics
	State of the Activities	{Acquired, To be Acquired, To be revised, Not to be suggested}	Statistics
Interaction between learner and system	Number of lessons read	Number	Statistics
	Number of activities done	Number	Statistics
	State of the Lesson	{Completed, Not completed}	Statistics
	Last connexion	datetime	Statistics
	Understanding of the activity	{Not Understood, Not well understood, Well understood, All understood}	Statistics
	Duration of the connexion	In minute	Statistics

4.1.8 Resources in PyLe

The current version of PyLe encompasses nine (9) lessons, as detailed in Table 4.2. The resources provided by PyLe place a strong emphasis on the syntax of the fundamental concepts of the Python programming language. These resources utilize a variety of mediums, including text, code, images, sound, video, and visualization. This multi-modal approach caters to diverse learning styles, thereby enhancing the learning experience.

Table 4.2: A Suggestion for the order of PyLe's Lessons [21]

Lesson	Description	Number of activities
Lesson 1	Output function	09
Lesson 2	Some Basic Programming Notions	09
Lesson 3	Variables	14
Lesson 4	Simple Data types	04
Lesson 5	Input Function	09
Lesson 6	List	37
Lesson 7	Conditionals	09
Lesson 8	Loops	15
Lesson 9	Functions	10

The code type resource in PyLe provides the instructor with the flexibility to add or modify a piece of code in the section. To alleviate students' frustration and enhance their debugging skills, the lessons incorporate, wherever feasible, the most common mistakes or misconceptions made by novices. These misconceptions, sourced from the literature, are accompanied by the corresponding compiler's error message, providing students with practical insights into error handling. Furthermore, the lessons include visualizations designed to bolster students' understanding of programming concepts. A distinctive feature of PyLe visualizations, as shown in Figure 4.14 is the ability for students to provide the input data before execution. This feature contrasts with most existing visualizations, where the user cannot modify the input data, thereby offering a more interactive and engaging learning experience. The significant challenge in developing the content was to avoid problems based on mathematics, which are prevalent in traditional teaching methods and most learning environments. This approach ensures that the focus remains on understanding programming concepts rather than mathematical problem-solving.

LESSON 8: PHYTONS LOOPS

Prerequisites : print(); input() methods; variables; comparison; logical operators; list; if statements

Objectives : Define the for and while statements; Define some common errors

Teaching Method(s) : DTM, annotated examples, visualization, Low level approach

RESOURCES Time elapsed : 0:14 **N.B: It is only by clicking on the button NEXT that the section is considered read.**

FOR LOOP

[Read](#)

A for loop is used for iterating over a sequence a specified number of times, by using the range() function. The Python range() function returns a sequence of numbers, in a given range, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

range () function syntax:

```
range(start, stop, step)
```

Parameters:

- start: [optional] start value of the sequence
- stop: next value after the end value of the sequence
- step: [optional] integer value, denoting the difference between any two numbers in the sequence

Examples of use:

```
range(6) # The loop will be for values 0, 1, 2, 3, 4, 5 and when we will get to 6 it stops
```

[back](#) [+ Send feedback](#)

Progression

14%

Lesson Content

Section 1: Presentation ✓

Section 2 : For loop

Section 3 : Looping through a List and String

Section 4 : Imbrication of for

Section 5 : While loop

Section 6 : Continue and break into loops statement

Section 7 : Loop visualization

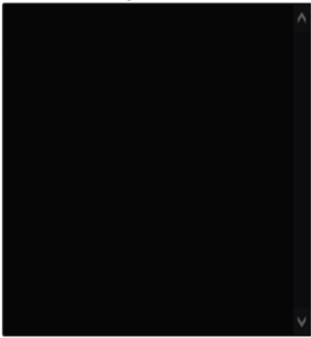
Figure 4.13: PyLe Lesson Overview [21]

For ease of understanding, enter the **children's** names in the field bellow, separating them with a ",". Do the same with the **Age** field. Then click on the next button to see the program run.

Code to execute

```
→ 1 children =
2 ages =
3 for child in children:
4     for age in ages:
5         print(child, age)
```

Output



Children = Ages =

[< Prev](#) [Next >](#) [reset](#)

Figure 4.14: Sample of Interactive Visualization [21]

4.1.9 Activities in PyLe

PyLe activities encompass a broad range of exercises designed to verify whether the student has comprehended the lessons and can consequently identify and correct syntax errors. Most of the activities are based on misunderstandings of programming students discovered in the literature and previously covered in the course content. These activities, as shown in Figure 4.15, are categorized into the following types:

- (i) Concepts inventories: Chapter 2 provided more details on concepts inventories. These are a series of quizzes based on students' misconceptions that can help the instructor determine whether or not students have mastered a concept [81].
- (ii) Pearson problems: Unlike most Pearson problems that use the drag-and-drop system, the Pearson problem in PyLe requires students to type their answers in the correct order. This ensures that students can enter the corresponding answer without making syntax errors.
- (iii) Matching questions: These use the drag-and-drop mechanism to test the students' understanding.
- (iv) Fill-in-the-Blank Questions.

Moreover, PyLe has resulted in a series of activities for all these categories, ready to be used for the aforementioned concepts in Python. These diverse and interactive activities aim to reinforce learning and enhance the student's understanding of Python programming syntax.

As discussed above, PyLe is an educational environment that incorporates students' misconceptions to enhance motivation and performance. It focuses on improving students' Python programming syntax and offers user-friendly content on Python basics. It is suitable for use in both blended and online learning settings for an IP course or self-directed learning. Its rationale stems from the fact that most lecturers struggle to integrate programming syntax into teaching, and current research often overlooks syntax issues and their relevance to non-computer science students. PyLe addresses these concerns by considering students' misunderstandings and introducing features for module reuse and content updates. Unlike other existing tools and given the target population and challenges discussed in [20], its content is based on real-life situations, not just mathematical skills, making it unique in its objective and development process.

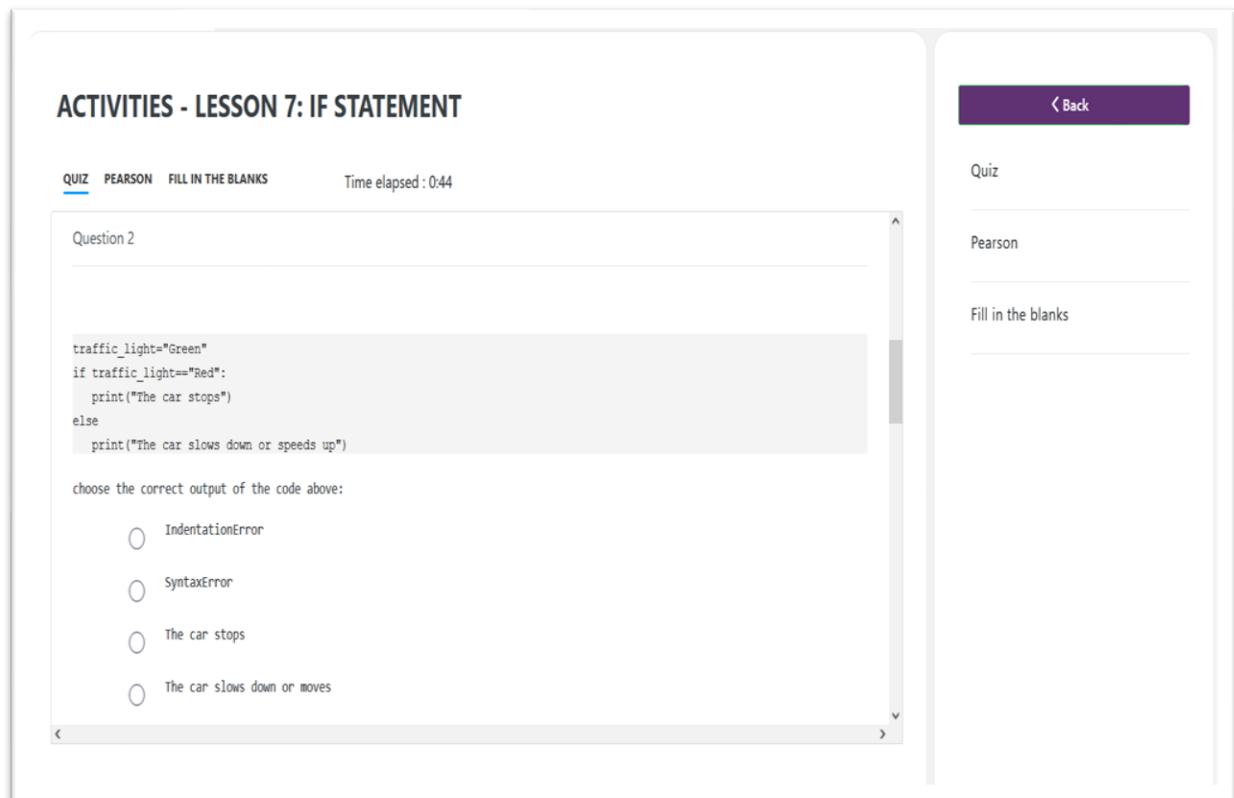


Figure 4.15: Overview of Activities [21]

Analysis

The previous sections introduced and presented, respectively, the EI framework, IIADPIE, and PyLe environments. This section provides an analysis of each of them respectively.

The IPEI framework deals with the teaching of core programming concepts to novices based on strategy and the construction of a knowledge scheme. IPEI starts each lesson with a short review and introduces new features step by step. It then proceeds with guided activities followed by students working independently, and thereafter tests students' acquisitions frequently to know what they have retained, not just to assess them. Therefore, IPEI places the teacher at the centre, and its implementation requires the teacher to have quality preparation in terms of choice of concept, quality of explanation, research on students' misconceptions, the development of CIs, the choice of examples and non-examples, etc. Incorporating misconceptions and CIs in the process of teaching IP would reduce students' frustration and discouragement, allowing the teacher to form an accurate impression of the student's comprehension of core programming notions. It sequences skills logically, focuses instructions on essential content, and provides a

wide range of examples and non-examples, primarily based on misconceptions of any kind. IPEI also allows the teacher, based on the results observed in one step, to return to the previous step for further explanation. The collaborative programming activity included in the IPEI stage is highly effective in promoting high-level cognitive skills in students. Through this collaborative exercise, learners experience their limitations, inconsistencies, and beliefs. IPEI as an instructional method satisfies the nine teaching events described by GAGNE [176]: Increase retention and transfer; Assess performance; Provide feedback on performance; Highlight performance; Guide the learner; Present stimulus material; Stimulate recall of previous knowledge; Inform the learner of the objective; Gain attention. It bounds all the teacher's and learners' activities in the classes and can be applied in both lecture and practical classes, contrary to most other methods [51], [108], [155], [177], [178] which base their methods on the coding part. IPEI acknowledges that programming is about analysis, designing, coding, testing, and debugging. We assume that the implementation of this teaching method supported by Ben Newmark's principles [60] in any IP module would improve student motivation, engagement, and performance. On the other hand, explicit teaching is the foundation of this approach and has been very successful in the teaching of mathematics, foreign languages, and so on. IPEI has five steps, and contrary to other IP teaching methods [19], [94], [96], [104], [109], [117], [118], bound all the activities of learners and teachers during classes. The IPEI approach can be used for both lectures and practical classes, which is not the case with most available IP teaching methods that instead emphasize coding or problem-solving activities. Furthermore, compared with traditional and other IP teaching approaches, IPEI focuses more on teaching strategies to better grasp programming content and prevent shallow learning. For example, as stated by Malik et al. [54], the traditional approach favours surface rather than deep learning, as problem-solving or content comprehension strategies are not highlighted. On the other hand, the IPEI approach requires a lot from teachers before and during classes, starting with content preparation, research and elaboration of misconceptions, and strategy preparation. The preparation of misconceptions and strategies is not common to most existing IP teaching methods. Table 4.3 provides a comparison of the IPEI approach with some existing IP teaching methods (rows in the table) based on some selected criteria (columns in the table). According to Table 4.3, among other observations, all the pedagogical approaches use constructivism as a learning theory and are aimed primarily at CS students, unlike IPEI, which uses cognitivism. This study assumes that the IPEI approach can significantly enhance the traditional teaching method used extensively in the teaching of introductory programming modules.

Table 4.3: Comparison of IPEI with some IP Teaching Methods

Teaching methods	Learning theory	Target population	Features
IPEI approach	Cognitivism	Non-CS students with no prior programming or mathematics experiences, slow-paced students	Includes 5 steps and promotes motivation, and self-efficacy. The teacher is an active or a Facilitator. Learners are active and passive depending on the step we are in. The method can be used for both practical and lecture classes and promote students' motivation.
Traditional method	Behaviourism	All students	Promotes performance. The teacher is active and the learner is passive. The method can be used for both practical and lecture classes.
Low-level approach [178]	Constructivism	First-year students	This approach is before the beginning of the programming classes and promotes performance. The teacher is the facilitator and the learner is active. The method can be used for lecture classes.
Agile process [62]	Constructivism	First-year programming with no prior programming experience	Includes 4 steps with teacher feedback and promotes motivation and engagement. The teacher is the facilitator and the learner is active. The method can be used for practical classes.
ADRI model [57]	N/A	CS students	Includes 4 steps and promotes more practice of programming. The method can be used for both practical and lecture classes.
Problem-based learning (PBL) [96]	Constructivism	CS students	Promotes problem-solving skills, critical thinking, motivation and engagement. The method can be used for practical classes.
Mixed-methods survey design (Gamification with Digital Badges) [119], [120]	Constructivism	Mostly CS students	Promotes motivation. The method can be used for practical classes.
Mixed-contexts teaching approach [117]	N/A	Non-CS (Civil Engineering students)	Promotes motivation.
Holistic approach (Python as the first programming language, project-oriented and Problem-based learning, multimedia resources, and evaluation rubrics) [104]	Constructivism	CS	Promotes engagement. The teacher is the facilitator and the learner is active. The method can be used for practical and lecture classes.

Peer review [104]	Constructivism	CS	Promotes engagement and performance. The teacher's role is not mentioned and the learner is active. The method can be used for practical classes.
-------------------	----------------	----	---

This chapter also proposes and presents an effective ESD framework. When educational software is simple to integrate and improves outcomes, it is efficient. Applying the IIADPIE framework to ESD will assist the project team in evaluating what has to be done and where to intervene to achieve more consistency, resulting in a reduction in production overheads and time lost. The IIADPIE framework recognizes teacher and learner interface production during each Scrum sprint and defines the number of interfaces, learning and instructional approaches, resources, and activities for each pedagogical grain. Each pedagogical grain has a clear teaching strategy. The proposed method includes misconception studies and concept inventories to motivate and engage students. This framework suggests including student misconceptions in ES content to boost motivation and performance [166]. The study on misconceptions will help students avoid past or expert mistakes. Concept inventories evaluate fundamental pedagogical grains, allowing teachers to assess student progress. In this novel technique for content adaptability based on student learning styles, defining and developing a learner profile should help teachers match content to the learner's learning style and training goals. Table 4.4 shows the theoretical IIADPIE framework phases. This seven-phase framework organizes educational software design using two agile and two IDMs. These approaches were not incorporated into ESD methods.

Table 4.4: IIADPIE Analysis [81]

Phase	Analysis
Initial phase	At this stage, the aspects examined are novel and not common to other methods.
Instructional orientation	Existing techniques [41], [46], [47], [49], [50], [180] share most of the components of this stage, with the notable distinction of the “learning resources” component.
Analyse	Most approaches [41], [46], [47], [49], [50], [180] describe analysis but not its details. The DSDM and ADDIE methodologies allow this framework to specify the analysis step at pedagogical and software levels. This includes avoiding the high prices and time waste of most solutions. The originality comes in studying field-specific learning and instructional approaches, defining the learner profile, and listing implicit non-functional needs.
Design	This framework defines the specific components and outcomes of instructional design and software design. The majority of the techniques [41], [46], [47], [49], [50], [180] currently in use do not address ergonomic needs analysis or design mode, leaving these aspects to the discretion of the developer. In this technique, one of the most important aspects is the examination of misconceptions. Other important aspects include the definition of lesson metadata, as well as the selection of activities and material types for each instructional element.
Production	This framework, as opposed to current practices, outlines the process for the

(SCRUM)	production stage.
Integration and implementation (SCRUM)	SCRUM methodologies give a clear framework for defining the critical parts that are necessary for effective implementation and integration, in contrast to traditional procedures [41], [46], [47], [49], [50], [180], which frequently rely on developers to make decisions without explicit consideration.
Evaluation (SCRUM)	The majority of the currently available techniques include evaluation, but they don't outline the process. Furthermore, the clarity of the features in each step will facilitate the development of assessment standards.

In this chapter, we have designed a novel learning environment for programming in Python called PyLe. PyLe is an educational environment designed for introducing Python programming, grounded in the IIADPIE framework. This framework integrates pedagogical and agile approaches to structuring and managing educational software design [81]. PyLe is specifically designed for non-computing students, focusing on syntactic issues in Python. It offers easily digestible content on the fundamentals of Python programming. In blended or online learning settings, instructors can employ PyLe for Python courses, while students can use it for self-directed learning and assessment. Unlike most existing tools [14], [15], [124]–[128], [130]–[133], [181], [182], PyLe incorporates students' misconceptions into its content development to alleviate widespread frustration during coding. Moreover, the key features of PyLe include module reuse across different classes and the ability for teachers to update existing content, aspects that are often overlooked in other tools. In addition, PyLe stands out due to its specific objectives and development process, which prioritize real-life scenarios, contrasting with tools that are primarily tailored to mathematically skilled students. Table 4.5 provides important details on the differences between PyLe and some existing tools.

Table 4.5: Theoretical Comparison of PyLe with Some Existing Learning Programming Environments [21]

Learning Tools	Education theory	Target population	Aims and learning context	Nature of the Content and Other Features
PyLe	Cognitivism	Non-computer students	Learning and understanding of programming syntax. Use in blended and e-learning contexts.	Resources (text, images, video, dynamic visualization, annotated examples, etc.), and activities that can be updated by the teacher. Based on the IIADPIE framework [81] and Python programming language. Includes output, comparison operators, logic operators, simple data types, input, lists, variables if statements, loops, and functions.
Practice Grids training system [124]	N/A	N/A	Code comprehension, construction, and non-mandatory practice. Can be used in blended situations.	Focus on Python programming language. It includes variables, if statements, loops, and logical operators.
Multimedia Learning Environment [14]	Constructivist	Computer education students	Code comprehension	Defined its development design method in four main points. Includes basic and advanced programming concepts.
Constructivist multimedia learning environment [15]	Constructivist	N/A	Support student interaction and group formation	No content.
Chatbox [126]	N/A	N/A	Facilitate self-paced learning and can be used in a blended context.	A development design method based on 10 principles for PHP and others not mentioned. Includes variable and data types, conditionals, loops, arrays, strings, and functions.
Pgtracer [127]	N/A	N/A	Moodle plugin for Code comprehension. It is used as a homework aid in programming.	Focus on C programming language and includes loops, functions and pointers.
SICA2 [128]	Constructivist	N/A	Moodle plugin for the construction of program flowcharts.	Programming concepts included variables and basic control structures.

Chapter Summary

In this chapter, we introduced a new way of teaching computer programming called the IPEI framework, designed to boost the motivation of non-computing students. IPEI framework is based on cognitivist theory and consists of five phases: reviewing, model-based programming teaching, guided programming, non-guided programming, and report and transfer. What sets it apart is that it can be used for both lectures and practical sessions, unlike other methods that mainly focus on coding and are student-centric. Our approach, IPEI, is teacher-centric, offering significant guidance to students. This chapter also introduced a new ES design. The suggested framework combines Scrum and DSDM's best practices with ADDIE and ASSURE's educational approaches. The framework comprises the initial stage, instructional orientation, analysis, design, production, integration, implementation, and evaluation. Detailing each phase helps designers navigate the development process. The goal is to improve ES usability, product efficacy, and learner motivation and engagement. This chapter further presented PyLe, a novel tool designed to help non-computing students master Python syntax. PyLe's significance lies in its focus on improving syntax error detection and correction skills, an area often overlooked by existing programming learning tools. Our PyLe, designed using the IIAPDIE framework and cognitivism principles, offers dynamic resources and activities. It can be utilized for self-learning or in blended and online educational settings, providing various features beneficial for educators. The next chapter will present the implementation and evaluation of these two frameworks.

CHAPTER 5 RESULTS AND ANALYSIS

Chapter Outline

This chapter examines the programming difficulties encountered by non-CS first-year students at NWU, South Africa, and CS first-year students at UDS, Cameroon, as well as their features. This chapter further provides a thorough examination of the implementation and evaluation of the IE framework and PyLe learning environment. The findings are presented and analysed in detail in this chapter.

Introduction

This chapter is essential for answering RQ2.1, RQ3.1, and RQ3.2, and for verifying the hypotheses formulated in Chapter 3. It outlines the stages of demonstrating and evaluating the study methodology, as seen in Figure 3.1. Therefore, to address RQ2.1, this chapter conducted a comprehensive survey (first phase) to identify the ongoing challenges that students continue to face, despite the implementation of several interventions aimed at enhancing their programming learning in IP modules. We did this by completing a quantitative questionnaire survey of first-year students from the NWU, FNAS, South Africa, and the UDS, Cameroon Department of Computing. Instead of only computing students, we targeted NWU students who double major in Agricultural, Geography, Physics, Mathematics, Chemistry, Biology, and other fields. This study examined students' backgrounds, learning and instructional settings, learning material preferences, and IP module difficulties. It also sought to investigate whether the programming barrier is institution-specific or shared by computer and non-computer students. Participants completed questionnaires, and we analysed the results. Both of these groups' programming challenge analysis and comparisons yielded recommendations for the EI framework and PyLe learning environment described in the preceding chapter.

On the other hand, to answer RQ3.1, this chapter presents the implementation of the EI framework (second phase) using two case studies. Firstly, in a CMPG121 module with non-CS first-year students from the FNAS, NWU, Mafikeng, South Africa (Group 1). Secondly, in an introduction to Python course, the groundwork for module INFO2122, OOP with Python, was laid with first-year trainee teachers from the HTTC, UY1, Cameroon (Group 2). After this implementation, this chapter presents the results of the post-experimentation surveys conducted to find out how the proposed teaching framework impacts student motivation.

In the same vein, to answer RQ3.2, this chapter implements and evaluates the effectiveness of PyLe (third phase) on first-year students at NWU in South Africa and the UY1, Cameroon. Firstly, the study experimented to assess the effect of PyLe on the time taken to solve a problem and the response quality. Secondly, the students and instructors evaluated PyLe's usability and its instructional value, respectively. This chapter outlines and analyses all the results. It is noticed that phases 1 and 2 have been published, respectively, in [20] and [21].

Phase 1: IP Challenges for First-Year Students in CS and Non-CS Fields

The difficulties encountered by students in IP modules are a major concern in CSE. This research identified some of these studies [27]–[30] carried out to highlight programming difficulties among students. On the other hand, the majority of these studies have concentrated on undergraduate students majoring in CS. Regrettably, only a few of these studies have investigated students from different academic disciplines who have chosen to take IP as an elective module. Programming difficulties disproportionately impact these student groups [20]. Therefore, there is a pressing need to investigate the various challenges these students encounter when offering programming modules as electives.

Therefore, we designed our study to identify the challenges faced by both computing and non-computing students across different institutions. Moreover, the study highlighted certain requirements for a teaching method and learning tools aimed at non-computer science students.

Hence, we used the questionnaire from Chapter 3 to conduct a thorough survey based on a quantitative research design methodology. This was to answer RQ2.1 of our research. The target audience consisted of non-CS first-year FNAS students from NWU, South Africa (Group 1), spread across two campuses: Mafikeng and Potchefstroom, as well as first-year CS students from UDS, Cameroon (Group 2). The study followed a cross-sectional design, resulting in a sample of 260 respondents, 81% from NWU and 19% from UDS. We distributed questionnaires for data collection to participants at NWU's Mafikeng campus (face-to-face), NWU's Potchefstroom campus (online), and UDS (online). Subsection 5.1.1 presents Phase 1's results and analysis.

Phase 2: IP-EI Framework Evaluation

This subsection presents the experiments of applying the five stages of the EI framework to the teaching of computer programming presented in Chapter 4. The goal was to answer RQ3.1 of our

research regarding the implementation of the EI framework. As stated in the introduction, these experiments involved two groups. The experiment with Group 1 was conducted during lectures and lab sessions with the participation of almost 45 students. Each session began with students being told that the module would use an experimental new teaching method, its processes, and their responsibilities. Because students seldom attend class, attendance lists are made after each lecture. In two weeks of testing, we focused on binary files. Before the experiment, we researched students' binary file misconceptions and prepared non-examples. The experiment was simple to set up because the framework had clear procedures. We were unable to complete all five framework phases in the first class because of the students' inattention. This occurred during modelling when the instructor was asking questions to engage students. None of them could replicate the strategy. Since the instructor would challenge them later, the students' interest increased once the teacher repeated the method during other classes. As a result, the subsequent classes progressed smoothly into the next stages. The other experiments went nicely too. In contrast, guided programming randomly establishes groups of 3-5 students with their consent. On the other hand, the experiment with Group 2 included fifteen students randomly selected. The IF statement and For loop were taught in a week-long experiment. Compared to the previous intervention, all stages of the teaching method were run throughout each course session. Participants reacted smoothly and were engaged during the experiment, which recorded no missed sessions.

We used the structured questionnaire from Chapter 3 to collect data on students' motivation after the experience to address question 3 of our research on the impact of the EI framework. The sample size consisted of 49 randomly selected students, 24 from Group 1 and 15 from Group 2. The sample from Group 1 corresponded to the exact number of students who attended all classes and agreed to take part in the survey during the last experiment class. Therefore, the data collected correspond to the responses of 49 students who completed the questionnaire to assess the impact of the proposed EI framework on motivation.

Phase 3: PyLe's Evaluations

This section presents the experiments conducted with PyLe to assess its usability and potential impact on the instructional process, as well as its effectiveness in improving the programming skills of non-computing students. We utilized these experiments to address RQ3 of this study. We have set the following goals to address this research question: *GO1: Evaluate how PyLe influences students' performance in terms of learning and mastering program syntax; GO2:*

evaluate PyLe's usability and its potential to affect the instructional process. Figure 5.1 presents an overview of the implemented strategy in this evaluation.

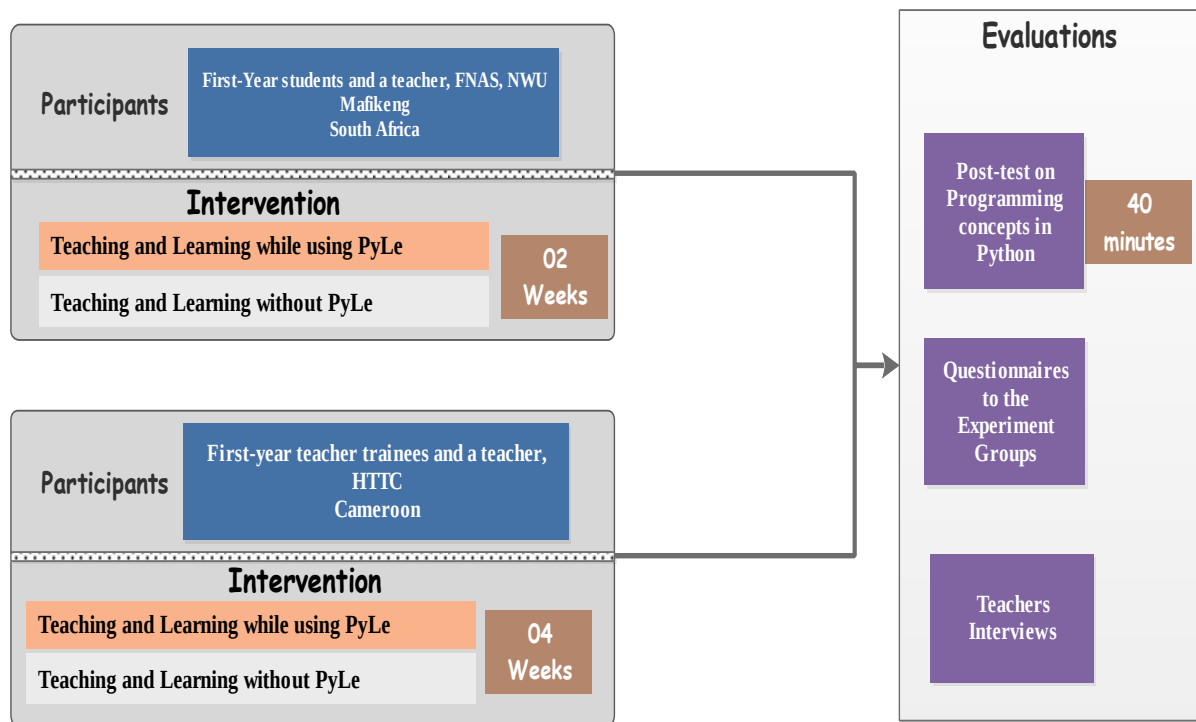


Figure 5.1: Experiments and Assessment Workflow

The first experiment tested PyLe's effectiveness by testing whether the time (TT) to solve a problem affects the grade or quality of the solution and whether PyLe can teach program syntax better than not using PyLe. In so doing, we addressed GO1, previously defined. The experiment included first-year trainee teachers from the HTTC in Cameroon and FNAS students in South Africa, who participated in the study. At both institutions, 15 HTTC and 20 FNAS students formed control and experimental groups. All 70 students took introductory Python programming courses, with Cameroon students in the second semester of 2023-24 and South Africa students in the first semester of 2024. The FNAS course is part of the CMPG111 module, while at HTTC, it prepares for INFO2122 on OOP with Python. Students in HTTC's experimental group (EG) used PyLe in a blended learning environment for nine lessons. We used in-person sessions so the instructor could address student concerns and explain. A WhatsApp group provided ongoing support and engagement. The FNAS EG used PyLe online with a WhatsApp support group. The experiment examined conditional statements and loops. The control groups (CG) at both universities took traditional courses on the same topics. Experimental programming included conditionals and

loops. Both universities' CGs took traditional programming courses simultaneously. The study examined how TT to finish a task affected GRADE. After TT, the GRADE is based on the quality of the students' final product. There were no group pre-tests. The experiments assessed students' Python program syntax knowledge using Chapter 3 post-tests. GRADE (0-100%) and TT for a 40-minute test were data. The independent variable is TT and the dependent variable is GRADE. Additionally, EG participants were coded E01 to E015 for HTTC and E101 to E120 for FNAS, while CG participants were coded C01 to C015 and C101 to C120.

We further employed post-experiment procedures that adhered to the quantitative design technique to address GO2. In this experiment, the participants were all CG students discussed above, totalling 35 students. In this case, we used the questionnaires designed in Chapter 3 following the experiment to assess the usability of the PyLe environment based on several usability assessment criteria, including visual design, navigation, accessibility, interactivity, self-assessment and learnability, and motivation to learn.

Moreover, still, in response to GO2, we also conducted post-experimental interviews with the two teachers responsible for the modules. This was aimed at providing feedback on PyLe's content, its utility, and their willingness to incorporate it into their teaching materials.

Results and Analysis

The section presents and analyses the results of phases 1, 2 and 3 discussed in the previous sections.

5.1.1 Phase 1's Results and Analysis

This section presents and analyses the results of Phase 1 above. This includes the participants' characteristics, the difficulties encountered in learning the IP modules, and comparative analyses of computer science students' and non-computer science students' results.

A. Participant features

This part organizes the results on the characteristics of non-computing students into the following categories: background profile, learning styles, and learning preferences. These categories are discussed as follows:

Background profile: We examined the overall characteristics of participants in both groups, including gender, prior programming experience, proficiency in mathematics and English,

utilization of the lab, and the duration of the course. These parameters are crucial for success in IP courses [9], [32], [44], [51], [52], [69], [71], [138], [183], [184]. The survey sample consisted of 72.3% men and 27.3% women, with the majority falling between the age range of 18 and 20 (57.7%). A majority of participants, specifically 52%, expressed dissatisfaction with the amount of time dedicated to IP modules, while 67% reported having no prior programming knowledge. Figure 5.2 displays the distribution of Group 1 across several fields of study. Analysis indicates that a majority of respondents in Group 1 (52.2%) possessed a level of mathematics that was at least average. Conversely, over 50% of respondents in Group 2 had a level of mathematics that was at most insufficient, as depicted in Figure 5.3 The data indicated that the students suggesting these IP courses were youthful and exhibited a wide range of experience levels, providing a challenge in designing instruction that would be engaging and beneficial for all learners.

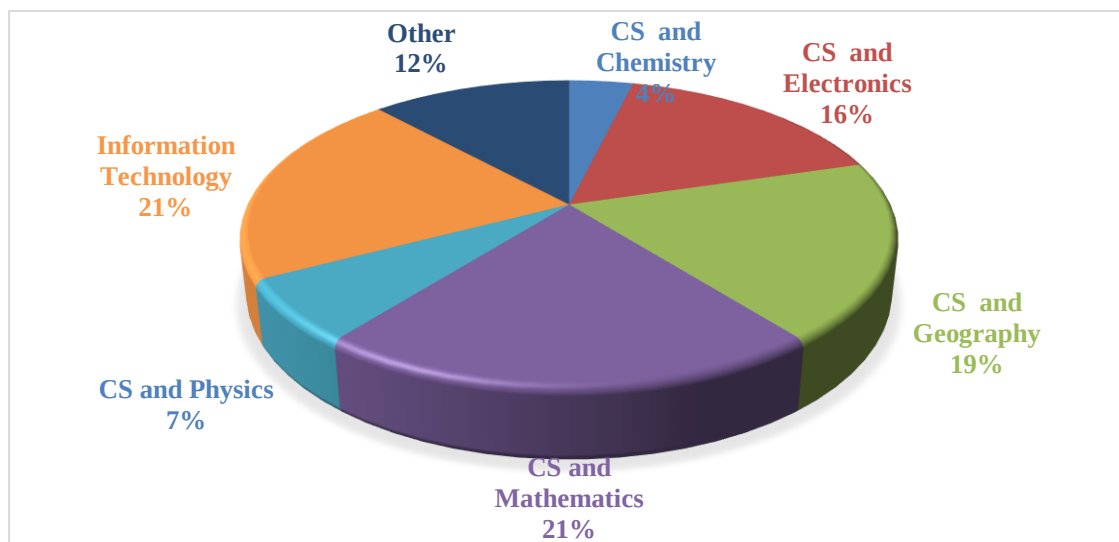


Figure 5.2: Repartition of NWU Respondents by Field of Study [20]

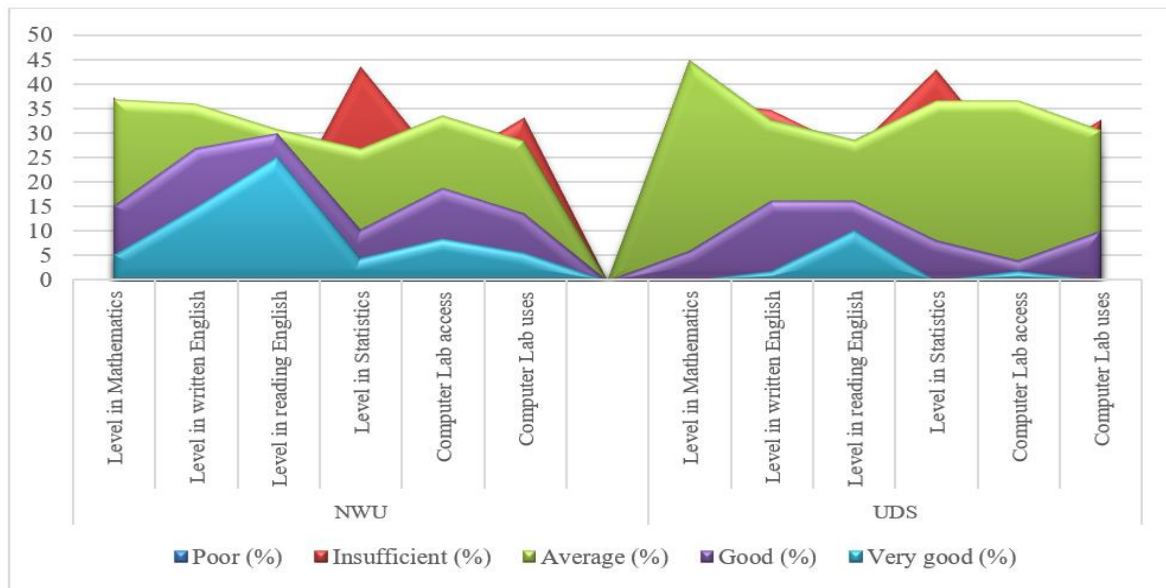


Figure 5.3: Students' Background Features [20]

Learning situations preferences: The findings presented in Figure 5.4 reveal that the majority of participants in Group 1 demonstrated lower levels of knowledge acquisition during lectures and lacked confidence when it came to studying and working independently on programming tasks. They derived pleasure from the alternative learning scenarios. Similarly, respondents in Group 2 demonstrated lower levels of learning during lectures but showed sufficient self-assurance to engage in independent learning and expressed enjoyment in alternative learning environments. Several studies [27], [29] have also found that lecture sessions decrease student participation. This emphasizes the need to employ instructional approaches that captivate and include Group 1 participants in their educational endeavours while fostering their self-assurance.

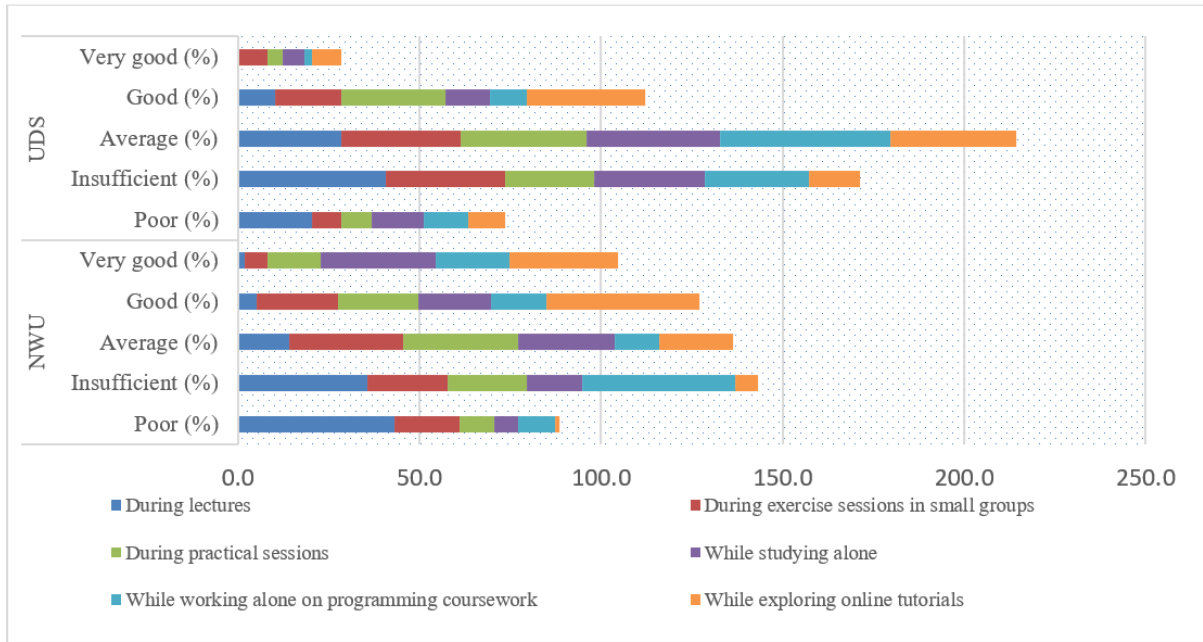


Figure 5.4: Students' Preferred Situational Learning [20]

Learning materials preferences: Upon analysing the results depicted in Figure 5.5, it becomes evident that most participants consider programming manuals, lecture notes, and PowerPoint slides to be less beneficial compared to other educational resources. Additional research, such as the studies referenced in [24] and [190], has similarly indicated that programming manuals are of limited utility to students. Hence, it is imperative to employ more cutting-edge teaching techniques and educational resources that emphasize visualization and interactivity. This will facilitate students' comprehension of programming texts and manuals, both within and outside the confines of the classroom.

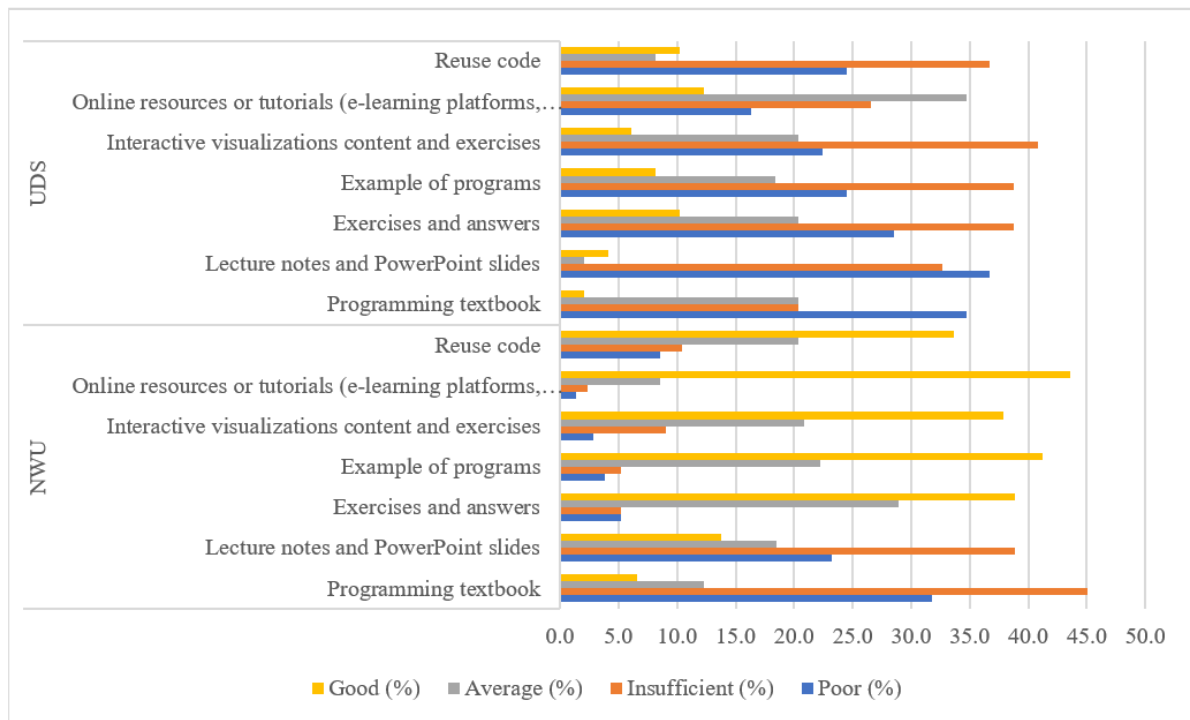


Figure 5.5: Students' Preferred Learning Materials [20]

B. Difficulties faced in learning IP modules

According to the IP modules examined in this research, the C and Python programming languages, the examination of the results in Figure 5.6 revealed that the majority of the questions in this area were scored as "insufficient" by the respondents. Consequently, they encountered the identical challenges that most novice programmers encounter. Other investigations have yielded comparable findings [29], [185]. To facilitate students' comprehension of these programming challenges, it is critical to employ a meticulously crafted pedagogical approach and effective educational resources.

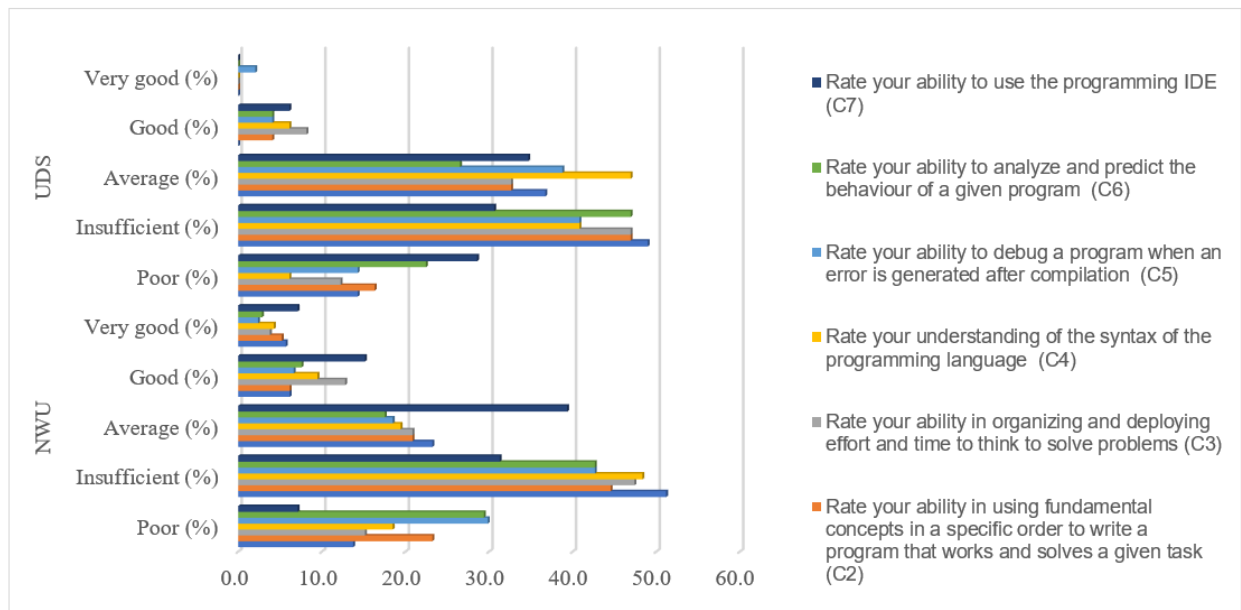
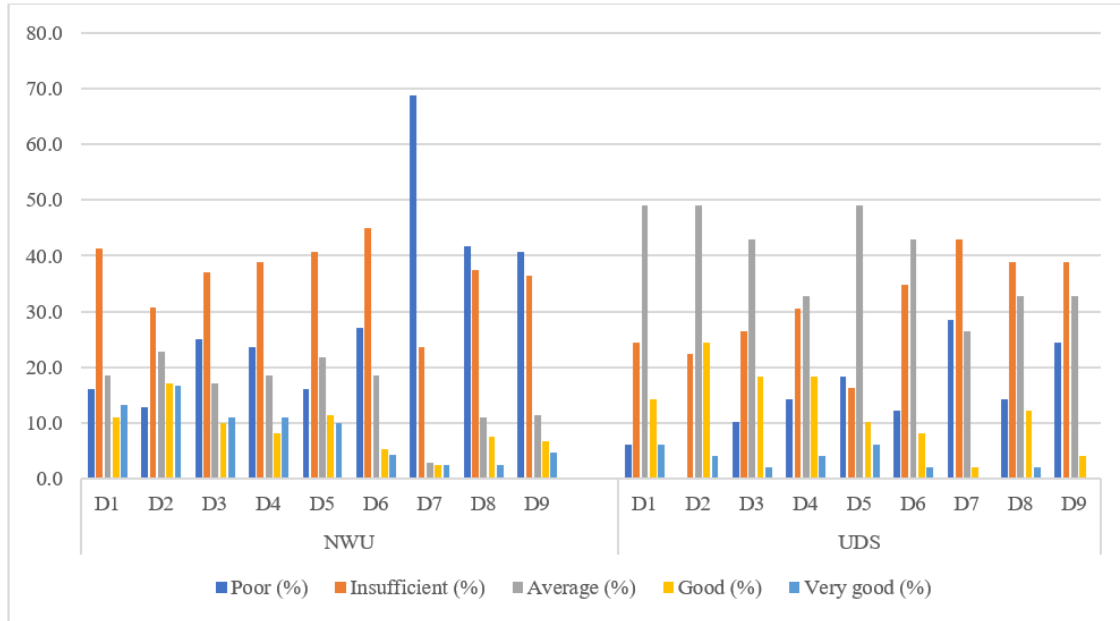


Figure 5.6: Students' General Opinion of Computing Programming [20]

On the other hand, analysis of the results in Figure 5.7 reveals that most Group 1 respondents assess their comprehension of basic programming concepts as "insufficient" at best. The concept of the comparison operator was the only one that received an "average" rating, of 66%. Respondents in Group 2 ranked the majority of concepts as "average". This group of respondents assessed recursion and error handling as "insufficient" at most. Group 1 respondents have a low degree of appreciation for basic programming topics, and the difficulties they face are similar to those encountered by most beginners. More specifically, both groups appear to struggle with the following programming concepts: recursion, arrays, and error handling. Some studies have identified recursion and error handling as challenging programming concepts [29], [30], [185]. These findings cover the majority of the issues that make programming challenging for these students, who require suitable strategies from the instructor during class to help them learn. In addition, we asked participants to choose the programming topics they learned in class. Figure 5.8 shows that some respondents struggled to remember core programming notions. This could indicate that they were uninterested or engaged in the lesson, as you cannot teach programming without including variables, for instance.



D1: Variables; D2: Comparison operator; D3: Loop structures; D4: Sequential and conditional structure; D5: Input/output instructions; D6: Functions and procedures or Methods; D7: Recursion; D8: Arrays; D9: Error handling

Figure 5.7: Students' Proficiency in some Key Concepts of Programming [20]

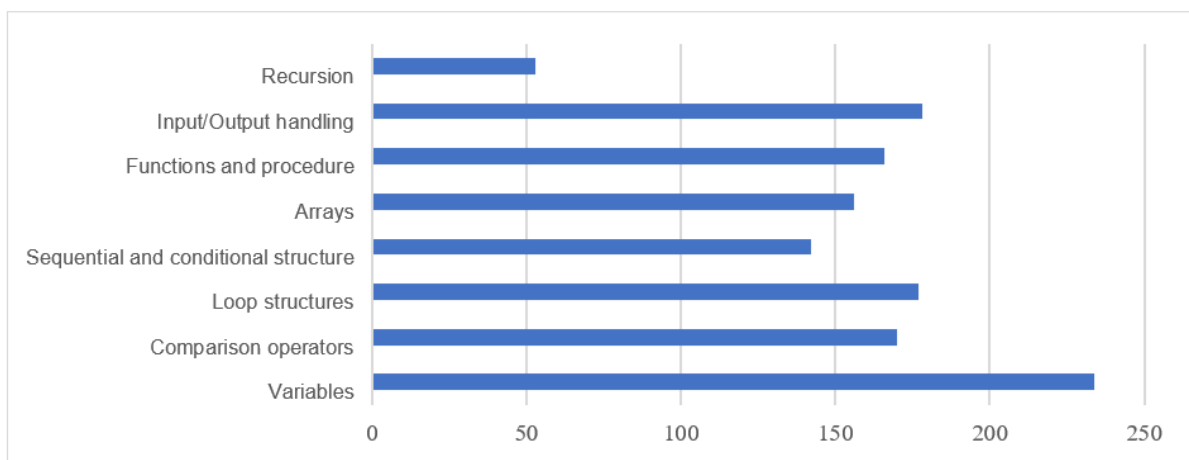


Figure 5.8: Selecting Frequency for Certain Programming Concepts [20]

C. Comparative Analysis

By comparing the data from the two groups and assessing the results of the data shown in Figure 5.7, we were able to identify the extent to which obstacles affect non-CS and CS students at NWU and UDS. Testing the first nine hypotheses presented in Chapter 3 determined the level of significance.

We used the Mann-Whitney U-test for the two independent groups. Because of the percentages involved in the groups, we randomly picked 49 respondents from group 1 to equal the number of respondents in group 2. The variables exhibited a non-normal distribution, as shown by the Shapiro-Wilk test. A significance level of $\alpha = 0.05$ was used, with acceptable p-values, $p > 0.05$.

Table 5.1: Mann-Whitney U Test Statistic [20]

	D1	D2	D3	D4	D5	D6	D7	D8	D9
Mann-Whitney U	772.50	965.00	661.50	846.50	879.00	804.00	611.00	543.00	852.00
P-value Asymptotic. Sig.	0.001	0.081	0.000	0.008	0.016	0.003	0.000	0.000	0.008

The Mann-Whitney U-test results (Table 5.1 **Erreur ! Source du renvoi introuvable.**) indicate that hypothesis H_{01} is rejected for variable D1 with $p = 0.001 < 0.05$. This indicates that there is a considerable difference between the two groups. Group 2 (UDS CS students) reported fewer difficulties than Group 1 (NWU non-CS students), with Group 1 having a mean T_1 rank of 44.77 and Group 2 having a mean T_2 rank of 58.23. For the comparison operators (D2), H_{02} is approved because $p = 0.082 > 0.05$, showing that the level of difficulty is similar for both groups. We found $p < 0.05$ for the following variables, indicating rejection: H_{03} , H_{04} , H_{05} , H_{06} , H_{07} , H_{08} , and H_{09} . Thus, students in Group 1 struggled more than students in Group 2 to understand the following basic programming concepts: input/output instructions, functions and procedures/methods, recursion, arrays, and error handling. These findings are consistent with those reported above, indicating that computer science students had fewer difficulties in the IP modules than non-CS students.

5.1.2 Phase's 2 Results and Analysis

The results presented in this subsection address RQ3.1, which pertains to our study. This study organized students' evaluations according to the ARCS model categories discussed in Chapter 2 (see Table 5.2). We therefore analysed each unit's outcomes in the categories as follows:

Table 5.2: Outcomes of the Descriptive Analysis of Participants' Motivation

Code	Criteria	Mean	Mode	SD*
Attention		3.88	4	0.81
A1	The content has things that stimulated my curiosity	3.94	4	0.92
A2	The quality of feedback helped to hold my attention	3.69	4	1.00
A3	The variety of examples helped keep my attention	3.69	4	0.96
A4	The explanations were so abstract that it was hard to keep my attention	2.43	2	1.10
A5	The integration of misconceptions in the course content helped keep my attention	3.82	4	0.99
Relevance		3.65	4	0.81

R1	The way in which the content was presented conveyed the impression that the related programming concepts are worth knowing	3.73	4	0.91
R2	The strategies developed during classes were useful to me	3.47	4	1.14
R3	The examples or exercises matched my interest	3.35	3	0.78
R4	The examples or exercises were all relevant to me	3.67	3	0.94
R5	There were examples or exercises that showed me how programming solves a real-life problem	4.14	4	0.87
Confidence		3.86	4	0.76
C1	As I worked in pairs and individually as well on programming problems, I was confident that I could learn programming	4.27	5	0.78
C2	I felt confident in solving a real-life problem	3.61	4	0.93
C3	After attending classes for a while, I was confident that I would be able to pass a test or exam	3.65	4	0.97
C4	I could not understand anything after multiple attempts working on programming problems whether in pairs or individually	2.29	2	1.04
C5	The strategies were more difficult to understand	2.16	2	1.16
Satisfaction		3.94	5	0.97
S1	I enjoyed the classes so much that I would like to know more about programming	3.73	4	0.97
S2	This teaching approach improved my programming learning experience	3.49	4	1.12
S3	It was a pleasure to participate in such an experience	4.02	4	0.88
S4	Feedback provided during guided activities was well appreciated	3.71	4	1.04
S5	Completing the exercises individually gave me a satisfying feeling of accomplishment	3.96	4	0.89
S6	This teaching approach does not provide more satisfaction than traditional teaching	2.33	2	1.21

*SD: Standard deviation

1. *Attention*: This study assessed respondents' attention based on five statements, and analysis of the results presented in Figure 5.9 shows that most participants agreed with the following points: the content has things that stimulated their curiosity; the quality of feedback helped them hold their attention; the variety of examples helped keep their attention; and the integration of misconceptions helped keep their attention. Conversely, with an average score of 2.43 (SD = 1.10), the majority disagreed that the explanations provided during classes were abstract and difficult to maintain their attention. According to the results in Table 5.2, the average attention score is 3.88 (SD= 0.81), indicating that the majority of participants agree that the EI framework captivated their attention.

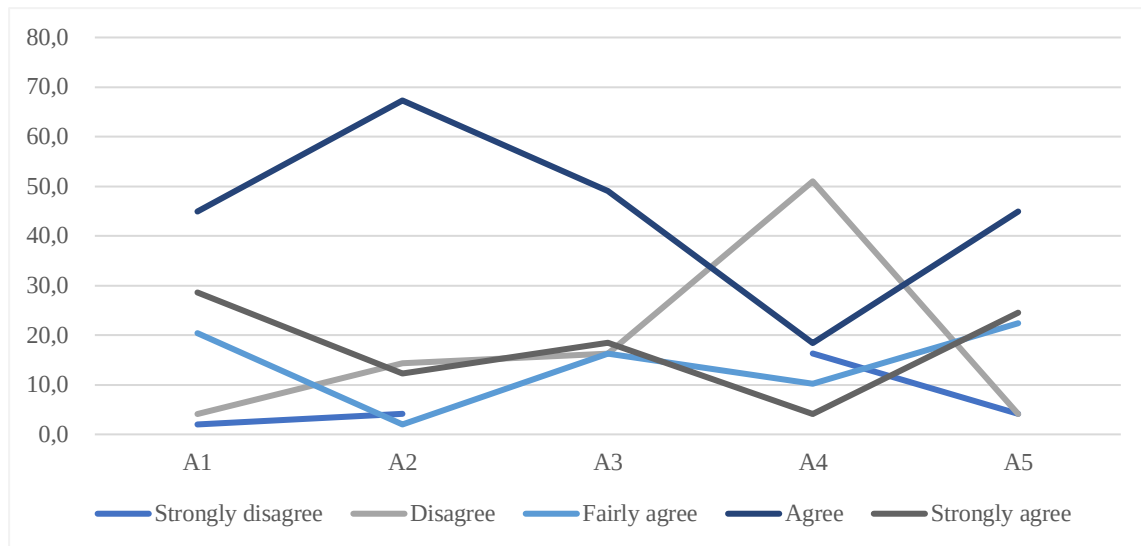


Figure 5.9: Attention

2. *Relevance*: This paper assessed participants' relevance based on five statements. Analysis of the results presented in Figure 5.10 reveals that most participants agreed with statements R1, R2, R4, and R5, with the following respective average scores 3.73 (SD = 0.80), 3.47 (SD = 0.91), 3.67 (SD = 0.94), and 4.14 (SD = 0.87). On the other hand, most respondents fairly agreed with the statement R3, with an average score of 3.35 (SD=0.94), meaning that examples or exercises need to be improved to better hold students' interest and relevance.

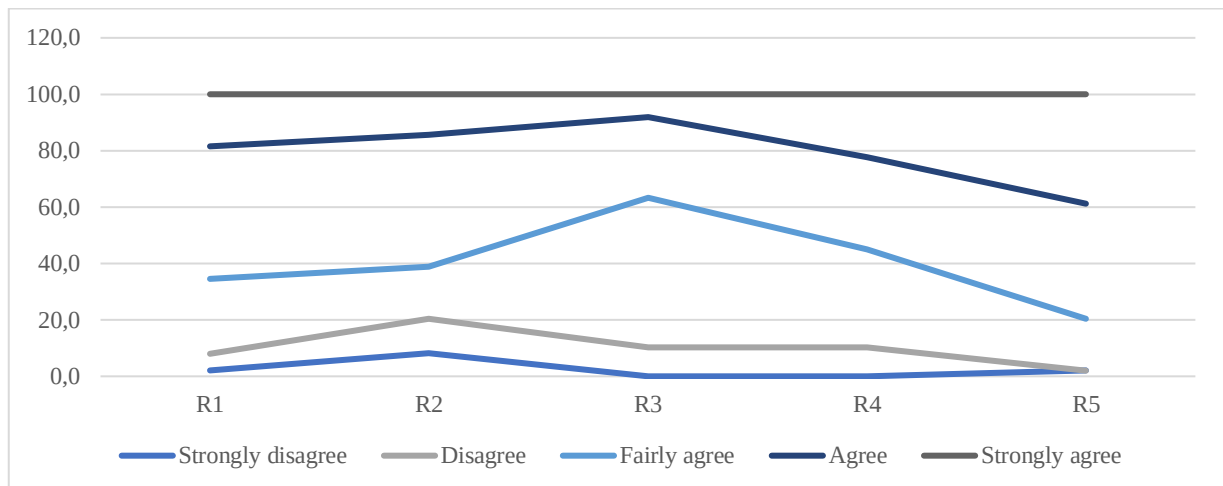


Figure 5.10: Relevance

3. *Confidence*: This study measured participants' attention on the strength of five criteria, and an examination of the findings as shown in Figure 5.11 indicates that students mostly strongly agreed

that the experience during the guided programming step improved their confidence in programming. The analysis of the results in Table 5.2 shows in detail that participants mostly agree with C1, C2, and C3, with the following respective average scores: 4.27 (SD = 0.78), 3.61 (SD = 0.93), and 3.65 (SD = 0.97). Regarding statements C4 and C5, the majority of participants expressed disagreement, with the value of 2 being the most frequently mentioned.

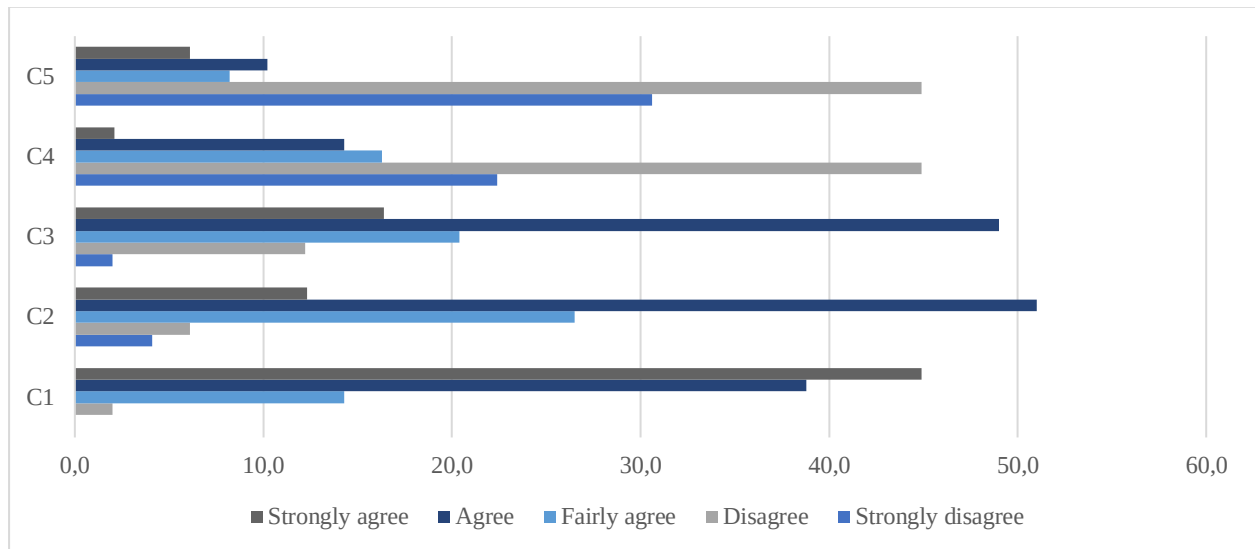


Figure 5.11: Confidence

4. Satisfaction: The examination of the results shown in Figure 5.12 of the six criteria used to measure students' satisfaction after the experiments highlighted that the majority of participants at least agreed with S1, S2, S3, S4, and S5. According to the results in Table 5.2, the majority of participants rated these statements as 4. On the contrary, most of them disagreed that the novel teaching framework did not provide more satisfaction than traditional teaching, with the value of 2 being the most frequently mentioned.

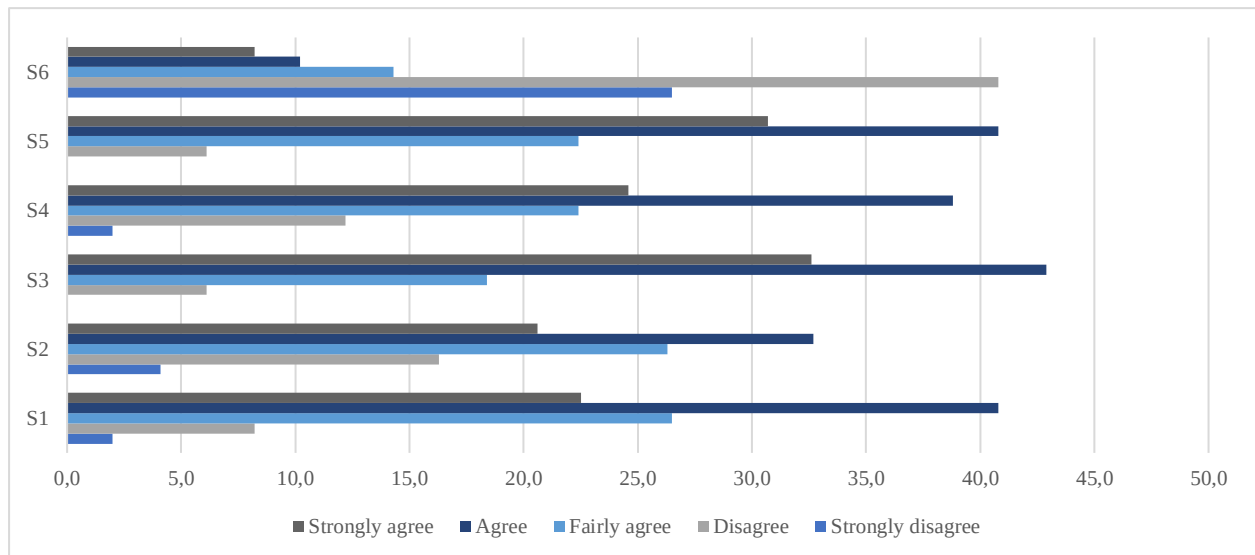


Figure 5.12: Satisfaction

5. *Correlation between ARCS model categories:* The Pearson inter-factor correlation coefficients obtained show a substantial positive link between all ARCS dimensions (see Table 5.3). The categories of relevance and satisfaction have the strongest association ($r = 0.561$), whereas attention and relevance have the lowest ($r = 0.414$).

Table 5.3: Correlation between ARCS dimensions

Scale	Attention	Relevance	Confidence	Satisfaction
Attention	1	0.414*	0.478*	0.418*
Relevance	0.414*	1	0.528*	0.561*
Confidence	0.478*	0.528*	1	0.440*
Satisfaction	0.418*	0.561*	0.440*	1

*Correlation is significant at the 0.01 level (2-tailed)

5.1.3 Phase's 3 Results and Analysis

This section organises the findings following the two goals formulated above: effectiveness results and usability and potential impact on instructional process results.

1. *PyLe's effectiveness:* As presented in Figure 5.13 and Figure 5.14, the participants' grades, reflecting their ability to learn and master problem syntax in Python, are highly satisfactory for the experimental groups (EGs), with the top score reaching 100%. Furthermore, the analysis of means and standard deviations reveals that EGs generally surpass CGs in terms of grades. Specifically, EGs mean grade is 61.5%, as shown in Table 5.4, which is significantly higher than

CGs mean grade of 37%. Conversely, when it comes to the average time spent on the test, CGs slightly outperform EGs, with a percentage of 23.6% compared to 21.6%.

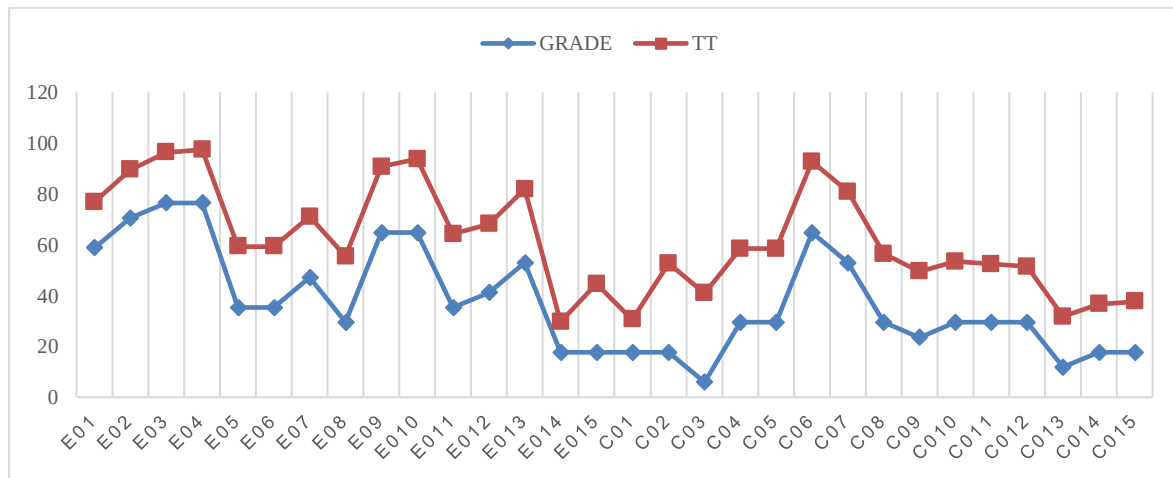
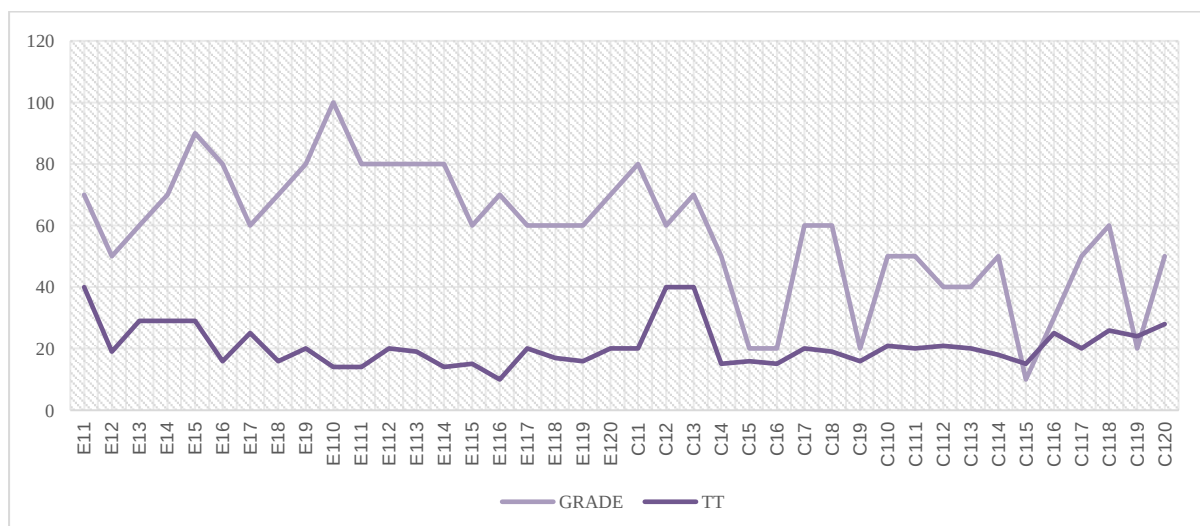


Figure 5.13: Post-test Results for Cameroon [21]

Table 5.4: Descriptive Analysis of Grade and TT [21]

		Mean	SD
Experimental Group	GRADE	61.53	19.59
	Time Taken (TT)	21.63	6.41
Control Group	GRADE	37.03	19.31
	Time Taken (TT)	23.34	6.79



In addition, to further determine the significance of the above results, we tested the formulated hypotheses H_{10} and H_{11} (see Chapter 3) using the independent-samples Mann Whitney U-test as shown in Figure 5.15 and Figure 5.16. The findings indicate that the p-value for the paired variable $GRADE_{CG} - GRADE_{EG}$ is 0.000, which is less than 0.05. This leads to the rejection of the H_{10} . Consequently, the alternative hypothesis that $GRADE_{CG}$ is superior to $GRADE_{EG}$ is accepted, given that the mean $\mu GRADE_{EG} = 61.53$ significantly exceeds the mean $\mu GRADE_{CG} = 37.03$. Moreover, the p-value for the paired variables $TT_{CG} - TT_{EG}$ is 0.361, which is greater than 0.05. This implies that the H_{11} cannot be rejected. Therefore, we can conclude that the difference between μTT_{EG} and μTT_{CG} is not statistically significant due to several factors that guide the experiment. As a result, the variance in the average time students take to complete the test between the two groups is not significant. Furthermore, a Pearson's correlation test was conducted, revealing that the p-value for the variables $GRADE_{EG}$ and TT_{EG} is 0.191, and for the variables $GRADE_{CG}$ and TT_{CG} is 0.233, both of which are greater than 0.05. This suggests that the student's grades are not dependent on the time taken to complete the test for participants in either group.

In light of these findings, while the use of PyLe did not consistently lead to students completing exams more quickly, it did significantly enhance their ability to learn and master Python's syntax. This is a reasonable outcome, considering that most of the students have not previously taken exams designed to identify and correct syntactic errors.

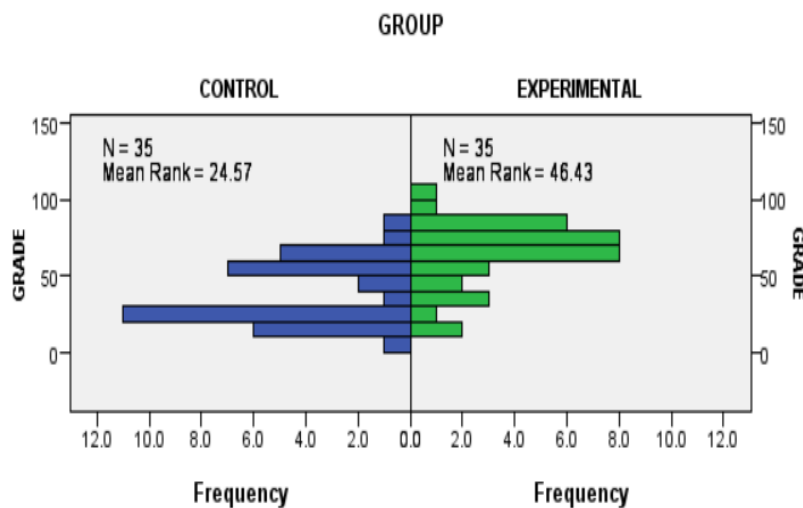


Figure 5.15: Independent-samples Mann-Whitney U Test on GRADE [21]

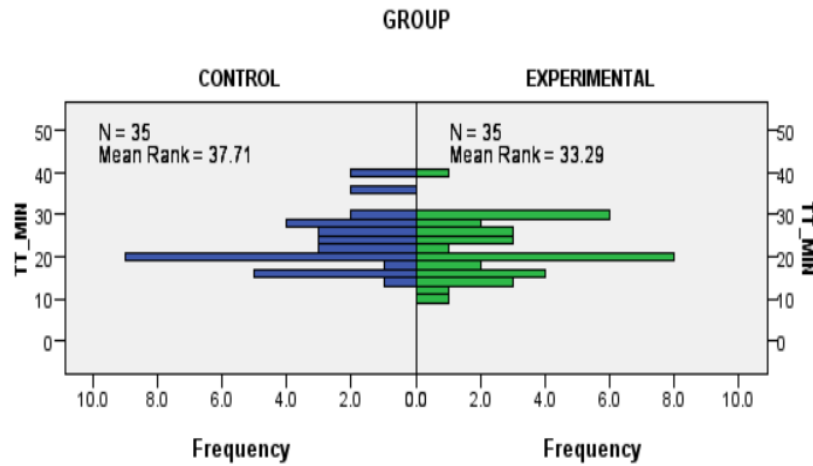


Figure 5.16: Independent-samples Mann-Whitney U Test on TT [21]

2. *Usability and potential impact on the teaching process:* The findings are detailed in Table 5.5. Findings analysis showed that students strongly endorse PyLe’s navigation, interactivity, self-assessment, and learnability features. However, they concluded that PyLe enhances their motivation to learn and that its visual design and accessibility are satisfactory. Furthermore, the majority of student responses centred around the corresponding means. Despite the acceptable responses, there is room for PyLe to enhance its accessibility. Particularly for the question regarding whether PyLe has no technical issues, the higher frequency value (mode) observed was 2, which corresponds to ‘Disagree’ on the Likert scale. This suggests a need to address potential technical issues in PyLe to improve the overall user experience.

Table 5.5: Descriptive Analysis Result of PyLe’s Usability [21]

Criteria	Mean	Median	Mode	Std. Deviation
Visual Design	4.11	4	4	0.79
Text, images, and visualization are easy to understand	3.91	4	5	1.12
Fonts (style, colour, etc.) are easy to read	4.03	4	4	0.86
Relevant information is placed in areas that catch your attention.	3.94	4	4	0.91
Navigation	4.60	5	5	0.49
You can decide which sections of the lesson you want to view	4.29	4	4	0.71
Lesson content is only a few clicks away	4.26	4	4	0.78
You can track your progress on PyLe	4.34	5	5	0.90
The menus for accessing content are organized	4.17	4	4	0.82
Accessibility	3.85	4	4	0.80

PyLe pages and other elements were easily accessible in a reasonable time	4.06	4	4	0.80
PyLe is easy to access from any platform	3.66	4	5	1.18
PyLe has no technical problems	3.17	3	2	1.24
Interactivity	4.40	5	5	0.73
PyLe offers facilities to make the learning process more engaging and motivating	4.20	4	4	0.75
PyLe provides access to a set of resources adapted to the learning context	4.31	4	4	0.67
PyLe engages learners in tasks that are closely linked to learning goals and objectives	4.20	4	4	0.90
Self-Assessment and Learnability	4.57	5	5	0.55
You can predict the overall result of clicking on each button or link	3.77	4	4	0.91
You can easily understand the purpose of using PyLe in the learning process	4.46	4	4	0.56
Each lesson offers you a set of activities to check your level of understanding	4.49	5	5	0.70
Several types of activities are offered in Pyle	4.51	5	5	0.50
PyLe activities prepare you to avoid a range of syntax errors	4.49	5	5	0.85
Motivation to learn	4.14	4	4	0.84
You've found the content very useful for understanding and avoiding syntax errors	4.20	4	4	0.86
PyLe's lessons encourage you to deepen your knowledge	4.26	4	4	0.78
You found PyLe's content enjoyable and interesting	4.03	4	5	1.07
PyLe content provides lessons that match your life experience	3.40	4	4	1.00
PyLe content provides frequent and various learning activities that set you up for success	4.17	4	4	0.89

On the other hand, the analysis of their interview responses revealed that the content was found to be factual, current, and comprehensive. The system layout was intuitive and easy to navigate, with real-world examples. The tools were deemed superior to comparable ones, with appreciated features like student progress tracking, content adaptability, relevant objectives, and logical task sequencing. The tool aligned with their ethical standards and values, and the content was consistent with the curriculum. Following the experiments, additional students were enrolled in PyLe, confirming its seamless integration into teaching and learning practices. However, suggestions for improvements included more visualizations, inclusion of other Python concepts, and expansion to other programming languages like C, C++, and Java.

Discussions

This section provides three main discussions corresponding to the study's results and analysis provided above.

5.1.4 Phase 1 Evaluation

According to the survey's findings, the respondents from NWU came from a variety of academic disciplines and are not students majoring in CS. Their Maths and English skills do not guarantee a good module performance. The challenges they have articulated and recognized contradict the results of various studies outlined here [69], [184], [186], which affirm that these elements are crucial for success in programming courses. We suspect that they have an inflated perception of their mathematical ability. Additionally, the majority of these students had no prior programming experience and reported that recursion, arrays, and error handling were the most challenging aspects of basic programming. UDS computer science students found these concepts hardest, but not as much as their peers. In addition, many of these students lack commitment and motivation in the module due to their inability to organize their time and effort to solve a programming problem or to remember certain programming concepts (Figure 5.8). Additionally, they reported learning less from lecture sessions and less from programming manuals and PowerPoint slides, as other studies mentioned above have also shown [189]. Once again, NWU respondents are less confident in working independently than UDS respondents. Based on these findings, we have developed specific needs, detailed here [20], and utilized them to create the frameworks showcased in Chapter 4. Here are some key points to consider: The teaching approach and learning materials should enhance students' motivation, engagement, and performance in IP courses, integrate interaction, foster students' self-efficacy and self-confidence, address common programming misconceptions, and so on.

5.1.5 Phase 2 Evaluation

This part discusses Subsection 5.1.2's results, as well as how they align with or differ from previous research. Results in Section 5.1.1 showed that they were not attentive during lessons and were largely unconfident. As a response to this, the results of the section demonstrated that using the IPEI framework overcame these difficulties, as it significantly improved their attention and confidence (see Figure 5.17). This is probably due to the involvement of students through questions during the model-based programming phase and the feedback they received during the guided programming phase. Similarly, the integration of peer programming followed by individual programming in its stages is a positive contribution to teaching practices, and most participants found it a great source of confidence. On the other hand, although most participants found the implementation of the IPEI framework relevant, we noted that a significant proportion of them fairly agree with the relevance of the exercises capable of addressing their interests. Other research found similar outcomes [96]. On the other hand, Horváth and Javorsk [187]

observe a frequent mismatch between the teacher's explanations and the students' understanding, indicating a significant failure factor in education. The EI framework fixes this problem by looking at criteria R2 (mean = 3.47, SD = 1.14) and R5 (mean = 4.14, SD = 0.87). Furthermore, our findings indicate that quality feedback is critical for attracting students' attention (mean = 3.69, SD = 0.96) and increasing their satisfaction (mean = 3.71, SD = 1.04), which is consistent with Jiau et al.'s [118] findings on the influence of feedback on students' self-motivation. Additionally, the varying standard deviation values for the criteria in Table 1 suggest that the participants' answers are homogeneous, implying that their responses are very close to the computed averages. As a result, the EI framework improves student motivation in all four areas of the ARCS motivation model, as does Santana et al.'s mixed-context teaching method [19]. Unlike Souza and Bittencourt's problem-based learning technique [96], which solely affected the categories of attention and satisfaction. Moreover, this novel framework provided satisfaction to participants, as most of them enjoyed the classes so much, found the framework more efficient than the traditional teaching method and were pleased to participate in such a teaching experience.

The development of a novel teaching approach to boost the skills of non-computer science students in the IP module has sparked renewed interest in computer science education [19], [25]. This study contributes to this trend by proposing a novel teaching framework based on cognitivist theory, which aims to influence the motivation of non-computing students who primarily take this course as an elective. The efficiency of the approach is dependent on the extent to which the instructor can effectively manage the time, prepare for the course, and successfully operate the classroom. If the modelling stage of the EI framework is not well organized, for example, we have discovered that there will not be enough time to finish the remaining framework stages. Despite the high levels of motivation demonstrated by the study's results, further research is necessary to validate and enhance the findings. This testing should take place over a semester, with a larger student population.

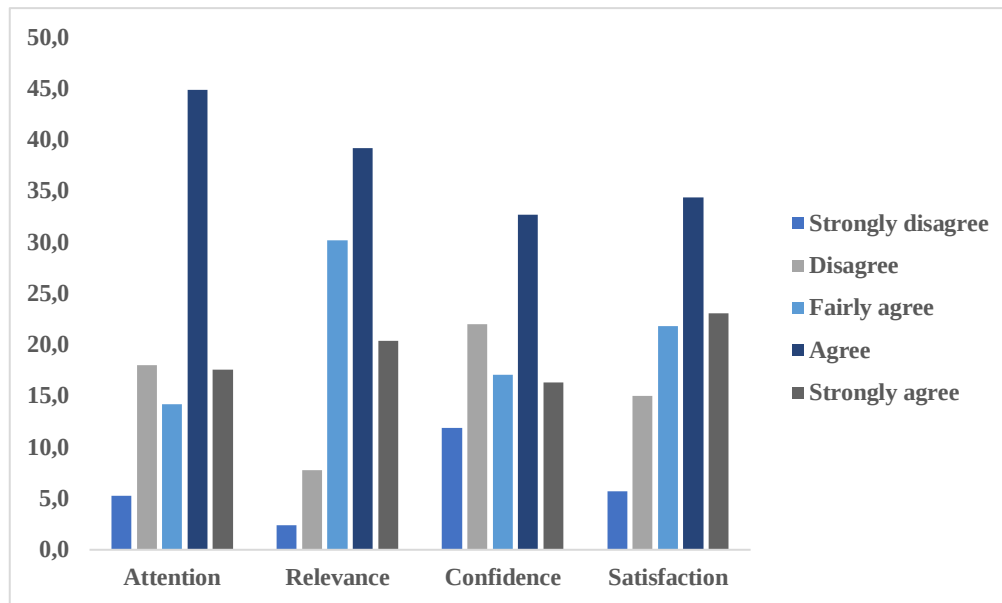


Figure 5.17: ARCS-model Summary Results

5.1.6 Phase 3 Evaluation

This part discusses Subsection 5.1.3's results. We have implemented (Subsection 5.1.6) and evaluated (Subsection 5.1.3) a novel learning environment for programming in Python called PyLe. PyLe underwent implementation and assessment on two modules across Cameroon and South Africa to determine its effectiveness in teaching and learning. The post-experimental evaluations conducted revealed that students who used PyLe outperformed their peers in assessing their ability to learn and solve syntactic programming problems. Interestingly, the TT by students to complete the task given did not significantly impact their final scores (Grade). This suggests that PyLe's unique assessment design effectively helps students learn, master, identify, and correct Python syntax. Furthermore, post-experiment surveys indicated high usability for PyLe. Most lessons were fully read, and participants completed the activities as shown in Figure 5.18 and Figure 5.19. In addition, an analysis of interviews with teachers confirmed their positive perception of PyLe, leading to its incorporation into their teaching practices.

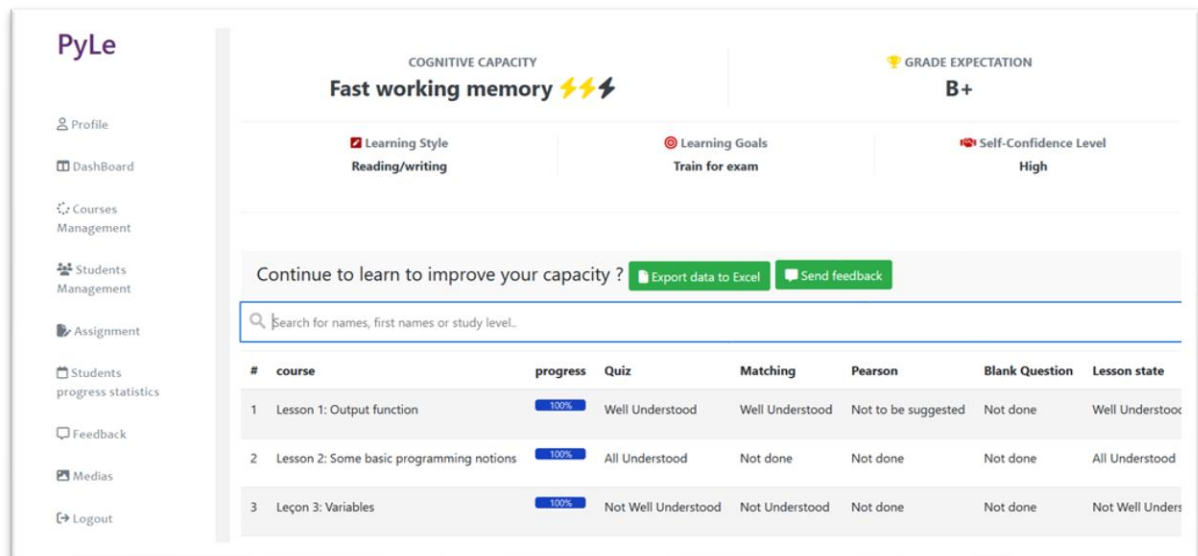


Figure 5.18: Overview of a Student Progress Report [21]



Figure 5.19: Student's Activities Report [21]

Validity Threats

This section discusses some of the sources of errors that can affect accuracy, as well as how we dealt with them to ensure valid results.

5.1.7 Phase 1

The survey in Subsection 5.1.1 did not gather further information from all students, particularly NWU foundation students. Foundation and first-year students complete the same module at different times and with different instructors. We did not include students from NWU Vaal, and the design of the question may have influenced the results and generalizations. Some measures reduced the threat. For instance, we used comparative studies to generalize the findings. Simple questions ensured data dependability and reduced response bias. To avoid confusion, the module instructors addressed specific questions during distribution and requested participants to respond honestly and without fear. This in-person survey was voluntary, and each participant received a paper questionnaire to prevent having to answer the same questions multiple times. The lecturers sent a link to their WhatsApp group with instructions for the online survey, which students could only take once on the same computer. All participants answered all questions and were about the population's age, and we do not think Foundation students' absence will affect results.

5.1.8 Phase 2

To ensure that the validity of the results presented in Subsection 5.1.2 is not compromised, several measures have been taken. Our sample consists of 49 students out of a population of around 88, giving us a representativeness rate of 55.6%, which, according to Lokesh's theory [188], is representative of the population, since this rate is higher than 20%. Moreover, to avoid response bias, questions were asked about the various teaching activities that took place during the lectures. Also, we provided and clarified certain incomprehensible questions. In addition, to avoid measurement bias, responses were based on a 5-point Likert scale to accurately measure participants' consent. To ensure data quality, a questionnaire was given to each participant, and only volunteer students were surveyed, those who were present for the entire duration of the experiment. The results analysis showed that all participants answered all the questions, and we are therefore confident in the quality of the results.

5.1.9 Phase 3

To ensure the veracity of the results in Subsection 5.1.3, we implemented various measures. We carefully recorded the reactions of PyLe users, and only those students who actively engaged with the environment by reading information and participating in activities were included in the post-test and the subsequent questionnaire. Moreover, the teachers who were interviewed were exclusively those who participated in the trials. The post-test questions were carefully curated to

only include subjects that were studied using the PyLe environment. To accurately estimate the consent of participants and prevent measurement bias, the responses to the questionnaire were evaluated on a 5-point Likert scale. Furthermore, to ensure the quality of the data, a reliability test was conducted using the data obtained from the students' usability survey. It is important to note that every participant completed the questionnaire and took part in the post-test. The analysis of these results has bolstered our confidence in the accuracy of the previously reported findings. Consequently, we are confident in the quality of the outcomes presented above.

Chapter Summary

This chapter implemented and evaluated the EI framework and PyLe environment presented in Chapter 4, discussing the results and analysis, as well as the survey's findings on the challenges first-year computer science and non-computer science students face when programming. The survey's findings show that non-computing students are young, aged 17 to 20. They lack the self-confidence to study alone and show more proficiency in group work and practical sessions than in traditional lectures or assignments. Again, non-computing students faced more programming difficulties than computing students, with understanding fundamental concepts such as recursion, arrays, error handling, and functions, procedures, or methods being considered the greatest difficulty for both groups. In this chapter, two case studies of the EI framework and PyLe implementation were presented in two IP courses in Cameroon and South Africa. This chapter also conducted post-experimental surveys using structured questionnaires and a written post-test to evaluate the impact of these frameworks on the programming skills of non-computing students. The overall results presented in this chapter showed that the EI framework positively impacted students' motivation in the following ARCS model categories: attendance, relevance, confidence, and satisfaction. In the same vein, students and instructors showed that PyLe was a valuable and acceptable teaching tool. The next chapter will provide a general conclusion about this study.

CHAPTER 6 SUMMARY, CONCLUSION, AND RECOMMENDATIONS

Summary

This section presents a summary of each chapter's research and illustrates the achievement of the study's objectives. As a result, in the first chapter, we introduced our research by highlighting the challenges beginners face in IP modules and how previous studies have addressed this issue. This led us to formulate our problem and our approach to solving it, which was based on cognitivism theory and targeted non-CS students. We further defined the three main RQs: *RQ1: What are the state-of-the-art methods and tools for teaching-learning programming? RQ2: How can we design and develop a teaching approach to motivate non-CS students to study computer introductory programming? RQ3: How can we implement this environment?* We further aligned these RQs with the following three ROs: *RO1- Conducting a comprehensive literature review on the methods and tools for teaching and learning computer programming; RO2- Conducting a survey to identify challenges and main features of the target population, to develop a cognitive-based teaching approach, and elaborate a unified learning system process building that coupled a software engineering technique and a learning system design method to ease the conception of the proposed interactive learning environment devoted to the target population; RO3- Implementing and experimenting with the interactive environment and the proposed teaching method, elaborating a questionnaire and conducting a survey to assess their effectiveness on students' motivation and performance.* In this chapter, we also outlined the study's aim, which was to design and implement a cognitive-based method and an interactive environment for introductory computer programming to assist non-CS students. We also provided in this chapter a brief introduction to DSR, the research methodology, and the methods employed to address the three primary RQs. This chapter has also highlighted the main contributions of this research to the field, namely, new teaching methods and learning material. This chapter, among other things, introduced this work's fourth publication and briefly presented each of the seven chapters of this thesis.

Chapter two aimed to achieve R01 by addressing RQ1 of our research. In this vein, we conducted a thorough review of existing teaching and learning approaches and tools developed to improve students' skills in IP courses. We critically examined these methods and tools, categorizing them into four primary categories: teaching-based, content-based, tool-based, and assessment-based.

The essentials of interventions fell into the teaching-based and content-based categories. This chapter outlined the essential interventions for improving beginners' skills in IP modules, which targeted CS students more than non-CS students and were based on the constructivist approach. This could potentially explain why some students are still struggling to follow IP courses, which continue to have high failure and drop-out rates, thereby sufficiently justifying our research. On the other hand, this chapter further reviewed two agile methods, the Scrum approach and the DSDM, as well as two ID models, ASSURE and ADDIE, to introduce the suggested framework for educational software design, developed in Chapter 4.

In chapter three, we went into great detail about the chosen approaches, DSR, and the mixed research methodology (quantitative and qualitative) employed for the study, as well as the research design. The research design included the research methods, the data collection methods, the data analysis techniques, the measurements, the hypothesis, the ethical considerations, the validity and reliability, and the research limitations.

Chapter four answered RQ2 with answers to the following sub-questions of RQ2: *RQ2.2- What teaching method can be developed to suitably satisfy the concerned population when learning computer programming? RQ2.3- How can we design an effective interactive environment that can help students in their learning process?* Hence, we suggested an explicit-based methodology for teaching introductory computer programming. This methodology, which aims to increase students' motivation, is based on the cognitivism movement. We further suggested an integrated framework for efficient educational software development. This novel framework guided the development of PyLe, a learning environment. PyLe is an interactive tool that aims to enhance the proficiency of non-computer science students in Python syntax. Therefore, we developed these three frameworks in response to RQ2.

Chapter five has fully completed the answer to RQ2. As a result, it addressed the following RQ2's sub-question *RQ2.1: What are the features and challenges faced by the targeted population when learning computer programming?* This chapter also addressed the following RQ3 sub-questions: *RQ3.1- How can we implement and evaluate the performance of the proposed teaching method and the interactive environment? RQ3.2- How can we appreciate the dynamic of student progression in learning computer programming?* We therefore examined and presented the characteristics of a group of non-computer science students, the difficulties they encounter in IP classes, and their learning preferences and styles. Additionally, we compared their programming challenges to those of computer science students. All of this was done employing a quantitative survey using a questionnaire with first-year non-CS and CS students at NWU in South Africa and

UDS in Cameroon. Further, this chapter offered some suggestions for developing a teaching and learning strategy and tool for non-computer science students. This chapter also presented the contexts of the implementation of the EI framework, and the PyLe environment in two universities, and discussed the results. For the EI framework, two trials were conducted at HTTC, at UY1, Cameroon, and at NWU, South Africa, to assess the efficacy of this strategy. The findings were reported. On the other hand, we assessed the efficacy and usability of PyLe at NWU and HTTC and presented our findings. *The surveys and experiments conducted in this chapter, as well as the corresponding results and analysis presented, have enabled us to fully achieve RO2 and RO3.* It is noteworthy that this research has led to the publication of four (4) papers. One article is now undergoing the review process. Figure 6.1 summarizes the research objective achieved, as well as the research question addressed in each chapter.

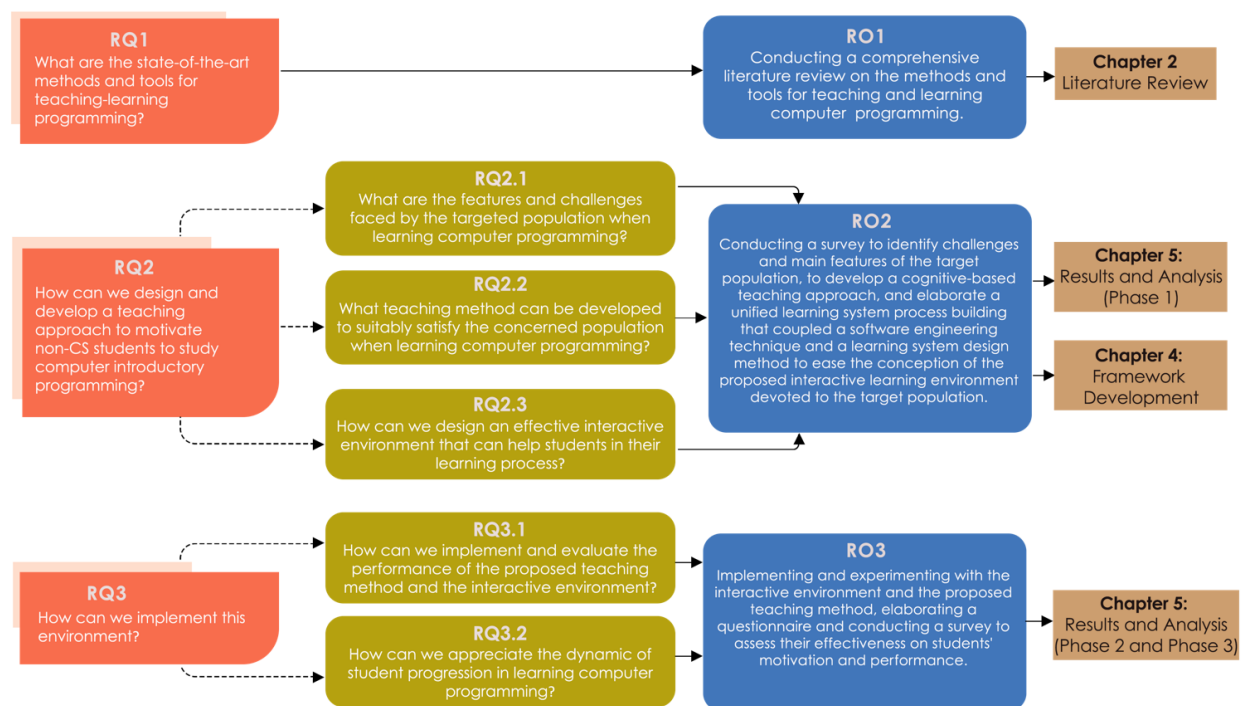


Figure 6.1: Mapping Research Questions to Research Objectives, and Chapters

Conclusion

CSE research tries to develop novel teaching and learning methodologies and tools, with a focus on improving students' programming skills. It was with this in mind that this research was undertaken. We based our intervention on the theory of cognitivism, and we have mainly targeted non-computing students. Indeed, the majority of existing interventions were geared toward CS

rather than non-computing students and were heavily inspired by constructivist philosophy. Additionally, the quantitative survey presented on students' challenges in IP modules in Chapter 5 sufficiently justified the choice of non-computing students, who faced greater programming difficulties than computer science students. This study first suggested an explicit-based methodology for teaching computing programming, designated as IPEI, to improve non-computing students' motivation. IPEI is organized into five stages: reviewing, model-based programming teaching, guided programming, non-guided programming, and report and transfer. IPEI is teacher-centric and enhances students' development of advanced cognitive abilities. It provides significant guidance to students, and its execution necessitates exceptional preparation in terms of concept selection, explanation strategy, research on student misconceptions, and so on. In contrast to other approaches that are student-centric and primarily focus on coding, it may be utilized for both lectures and practical sessions. The evaluation of the IPEI methodology using two post-experiment surveys on student motivation yielded positive results. Indeed, motivation is a key factor in the success of basic programming modules since it leads to students' increased enjoyment and engagement with the material. Thus, the ARCS model categories indicated that the IPEI had a favourable effect on students' motivation about attendance, relevance, confidence, and satisfaction (ARCS). We firmly believe that IPEI may be readily incorporated into the conventional approach to teaching programming. This is because each of its stages appropriately highlights the responsibilities and activities of both the instructor and the students. As a result, IPEI will become more widely used since, despite the abundance of methods available, many teachers continue to employ traditional methodologies.

On the other hand, non-CS have a particularly challenging issue understanding and comprehending programming language syntax. Most of the existing teaching techniques and materials are inadequate in this respect. As a result, to fill this gap, this study finally designed, implemented, and evaluated PyLe, a cognitive interactive learning environment tailored especially for Python introductory programming courses. It ought to highlight that PyLe was developed utilizing the educational software development methodology known as IIADPIE. IIADPIE consists of the following components: initial; instructional orientation; analysis; design; production; integration and implementation; and evaluation. These components reflect educational software development phases that use a combination of pedagogical methodologies and agile methods. IIADPIE guarantees that the final product fits the user's requirements and is effortlessly integrated into teaching and learning activities. PyLe explains the fundamentals of Python programming in simple terms. In mixed or online learning situations, teachers may use PyLe for introduction to Python courses, while students can use it for autonomous learning and evaluation. In contrast to

the majority of current tools, PyLe takes into account the misconceptions of learners while creating its material to reduce common frustration with coding. A post-test trial with 70 students from the above-mentioned universities was used to assess PyLe's efficacy. Groups for the experiment and the control of the students were formed. With a mean score of 61.53%, the collected data demonstrate that the control group fared better after utilizing PyLe. Additionally, quantitative surveys in the control group and teacher interviews with the modules were used to assess the usability of the proposed PyLe and its instructional impacts. The findings show that while students evaluate PyLe well for its capacity to assist them in learning and mastering programming syntax, teachers believe PyLe is acceptable and helpful for instruction. After the experiments, the instructors involved decided to employ PyLe in their entire class.

Recommendations

Due to time constraints, this study has encountered some limitations. We can document these limitations as recommendations and use them to guide future work as follows:

- 1. Additional Experiments:** We should test the IPEI approach across a variety of introductory programming courses at different universities to validate and enhance the findings of this study.
- 2. Expand PyLe's Content:** In this study, we limited PyLe's content to a few fundamental programming principles. This is although programming is a huge field, as well as time constraints. As a result, we could broaden the scope of PyLe's content by including other ideas such as dictionaries, polymorphism, modules, classes, objects, and so on. In addition, we ought to broaden the scope of PyLe so that it encompasses more programming languages, including C++, C, Java, and others. Furthermore, whenever possible, we should also create interactive visualizations that correspond with each new piece of content.
- 3. Improve PyLe's Interactivity:** This can be accomplished by implementing these suggestions: Incorporate an Integrated Development Environment into PyLe to enable learners to write code; Utilize artificial intelligence to incorporate a recommendation module that provides tailored suggestions for resources and exercises based on the learner's profile; Utilize automated tests to verify the learners' code and deliver prompt feedback; Implement a peer programming module to foster collaboration among learners; Utilize machine learning techniques to forecast student success and detect individuals who may want further assistance.

4. **Integrate Gamification into PyLe:** Create a system of badges that learners can earn as a reward for accomplishing specific goals, and include rankings to inspire learners to engage in healthy rivalry with one another.

REFERENCES

- [1] S. Fincher and M. Petre, *Computer science education research*. CRC Press, 2004.
- [2] C. A. Shaffer et al., 'Algorithm visualization: The state of the field', *ACM Trans. Comput. Educ. TOCE*, vol. 10, no. 3, pp. 1–22, 2010.
- [3] T. Crick et al., 'The impact of COVID-19 and "Emergency Remote Teaching" on the UK computer science education community', in *United Kingdom & Ireland Computing Education Research conference.*, 2020, pp. 31–37.
- [4] D. Weintrop, 'Block-based programming in computer science education', *Commun. ACM*, vol. 62, no. 8, pp. 22–25, 2019.
- [5] J. T. Anyango and H. Suleman, 'Teaching Programming in Kenya and South Africa: What is difficult and is it universal?', in *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*, 2018, pp. 1–2.
- [6] G. Hunter et al., 'Learn programming++: The design, implementation and deployment of an intelligent environment for the teaching and learning of computer programming', in *2013 9th International Conference on Intelligent Environments*, 2013, pp. 129–136.
- [7] C. S. Cheah, 'Factors contributing to the difficulties in teaching and learning of computer programming: A literature review', *Contemp. Educ. Technol.*, vol. 12, no. 2, p. ep272, 2020.
- [8] A. Robins et al., 'Learning and teaching programming: A review and discussion', *Comput. Sci. Educ.*, vol. 13, no. 2, pp. 137–172, 2003.
- [9] S. Wiedenbeck et al., 'Factors affecting course outcomes in introductory programming.', in *PPIG*, 2004, p. 11.
- [10] M. F. Ndjodo et al., 'Un profil de compétences pour les professeurs d'informatique de l'enseignement secondaire camerounais', *Int. Rev. Educ.*, vol. 59, no. 2, pp. 177–196, 2013.
- [11] E. Kolczyk, 'Algorithm–Fundamental Concept in Preparing Informatics Teachers', in *International Conference on Informatics in Secondary Schools-Evolution and Perspectives*, 2008, pp. 265–271.
- [12] R. P. Medeiros et al., 'A systematic literature review on teaching and learning introductory programming in higher education', *IEEE Trans. Educ.*, vol. 62, no. 2, pp. 77–90, 2018.
- [13] A. (1) Vihavainen et al., 'A systematic review of approaches for teaching introductory programming and their influence on success.', in *ICER 2014 - Proceedings of the 10th Annual International Conference on International Computing Education Research*, (1)Department of Computer Science, University of Helsinki, 2014, no. ICER 2014-
Proceedings of the 10th Annual International Conference on International Computing

- Education Research, pp. 19–26[Online]. Available <https://nwulib.nwu.ac.za/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-84905869869&site=eds-live>.
- [14] V. Mutiawani and others, 'Developing e-learning application specifically designed for learning introductory programming', in *2014 International Conference on Information Technology Systems and Innovation (ICITSI)*, 2014, pp. 126–129.
- [15] C. Samat et al., 'Design and development of constructivist multimedia learning environment enhancing skills in computer programming', in *2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, 2017, pp. 1023–1026.
- [16] S. B. Yusupova et al., 'The advantage of using e-learning in teaching students programming languages', in *2022 IEEE International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*, 2022, pp. 1910–1913.
- [17] J. Figueiredo and F. García-Peñalvo, 'Teaching and learning tools for introductory programming in university courses', in *2021 International Symposium on Computers in Education (SIIE)*, 2021, pp. 1–6.
- [18] J. Bennedsen and M. E. Caspersen, 'Failure rates in introductory programming: 12 years later', *ACM Inroads*, vol. 10, no. 2, pp. 30–36, 2019.
- [19] B. L. Santana et al., 'Motivation of engineering students with a mixed-contexts approach to introductory programming', in *2018 IEEE Frontiers in Education Conference (FIE)*, 2018, pp. 1–9.
- [20] A. Mbiada et al., 'A Comparative Study of Computer Programming Challenges of Computing and Non-Computing First-Year Students', *Indones. J. Comput. Sci.*, vol. 12, no. 4, 2023.
- [21] A. K. Mbiada et al., 'PyLe: An Interactive Tool for Improving Python Syntax Mastery in Non-Computing Students', *J. Inf. Syst. Inform.*, vol. 6, no. 2, pp. 1008–1034, 2024.
- [22] S. I. Malik and J. Coldwell-Neilson, 'A model for teaching an introductory programming course using ADRI', *Educ. Inf. Technol.*, vol. 22, no. 3, pp. 1089–1120, 2017.
- [23] N. Salleh et al., 'Cloud-based learning system for improving students' programming skills and self-efficacy', *J. Inf. Commun. Technol.*, vol. 17, no. 4, 2018.
- [24] N. Shi et al., 'Evaluating the effectiveness roles of variables in the novice programmers learning', *J. Educ. Comput. Res.*, vol. 56, no. 2, pp. 181–201, 2018.
- [25] A. K. Mbiada et al., 'Introductory Computer Programming Teaching and Learning Approaches: Review', in *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, 2022[Online]. Available <https://doi.org/10.1109%2Ficetecet55527.2022.9873427>.

- [26] E. Lahtinen et al., 'A study of the difficulties of novice programmers', *Acm Sigcse Bull.*, vol. 37, no. 3, pp. 14–18, 2005.
- [27] I. Milne and G. Rowe, 'Difficulties in learning and teaching programming—views of students and tutors', *Educ. Inf. Technol.*, vol. 7, pp. 55–66, 2002.
- [28] B. Thomasson et al., 'Identifying novice difficulties in object oriented design', *ACM SIGCSE Bull.*, vol. 38, no. 3, pp. 28–32, 2006.
- [29] P.-H. Tan et al., 'Learning difficulties in programming courses: undergraduates' perspective and perception', in *2009 International Conference on Computer Technology and Development*, 2009, vol. 1, pp. 42–46.
- [30] M. Piteira and C. Costa, 'Computer programming and novice programmers', in *Proceedings of the Workshop on Information Systems and Design of Communication*, 2012, pp. 51–53.
- [31] A. Ahadi et al., 'Learning programming, syntax errors and institution-specific factors', in *Proceedings of the 20th Australasian Computing Education Conference*, 2018, pp. 90–96.
- [32] A. K. Veerasamy et al., 'Identifying novice student programming misconceptions and errors from summative assessments', *J. Educ. Technol. Syst.*, vol. 45, no. 1, pp. 50–73, 2016.
- [33] K. S. Taber, *Modeling learners and learning in science education*. Springer, 2013.
- [34] Y. Qian et al., 'Teachers' perceptions of student misconceptions in introductory programming', *J. Educ. Comput. Res.*, vol. 58, no. 2, pp. 364–397, 2020.
- [35] Ž. Žanko et al., 'Mediated transfer: impact on programming misconceptions', *J. Comput. Educ.*, pp. 1–26, 2022.
- [36] F. Johnson et al., 'Experience Report: Identifying Unexpected Programming Misconceptions with a Computer Systems Approach', in *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, 2022, pp. 325–330.
- [37] F. Johnson et al., 'Analysis of student misconceptions using Python as an introductory programming language', in *Proceedings of the 4th Conference on Computing Education Practice 2020*, 2020, pp. 1–4.
- [38] Ž. Žanko et al., 'Analysis of school students' misconceptions about basic programming concepts', *J. Comput. Assist. Learn.*, vol. 38, no. 3, pp. 719–730, 2022.
- [39] A. Altadmri and N. C. Brown, '37 million compilations: Investigating novice programming mistakes in large-scale student data', in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 2015, pp. 522–527.
- [40] C. Taylor et al., 'Computer science concept inventories: past and future', *Comput. Sci. Educ.*, vol. 24, no. 4, pp. 253–276, 2014.

- [41] G. L. Herman, 'The development of a digital logic concept inventory', PhD Thesis, University of Illinois at Urbana-Champaign, 2011.
- [42] J. Libarkin, 'Concept inventories in higher education science', in *BOSE Conf.*, 2008.
- [43] V. L. Almstrum et al., 'Concept inventories in computer science for the topic discrete mathematics', in *ACM SIGCSE Bulletin*, 2006, vol. 38, pp. 132–145.
- [44] K. Longi and others, 'Exploring factors that affect performance on introductory programming courses', 2016.
- [45] C. Iriarte and S. Bayona, 'IT projects success factors: a literature review', *Int. J. Inf. Syst. Proj. Manag.*, vol. 8, no. 2, pp. 49–78, 2020.
- [46] R. M. Ryan and E. L. Deci, 'Intrinsic and extrinsic motivations: Classic definitions and new directions', *Contemp. Educ. Psychol.*, vol. 25, no. 1, pp. 54–67, 2000.
- [47] G. Nel and L. Nel, 'Motivational value of code. Org's code studio tutorials in an undergraduate programming course', in *ICT Education: 47th Annual Conference of the Southern African Computer Lecturers' Association, SACLA 2018, Gordon's Bay, South Africa, June 18–20, 2018, Revised Selected Papers 47*, 2019, pp. 173–188.
- [48] J. M. Keller, 'Development and use of the ARCS model of instructional design', *J. Instr. Dev.*, vol. 10, no. 3, pp. 2–10, 1987.
- [49] D. Bédard et al., 'Problem-based and project-based learning in engineering and medicine: determinants of students' engagement and persistence', *Interdiscip. J. Probl.-Based Learn.*, vol. 6, no. 2, pp. 7–30, 2012.
- [50] W. B. Schaufeli et al., 'Utrecht work engagement scale-9', *Educ. Psychol. Meas.*, 2003.
- [51] C. O'Malley and A. Aggarwal, 'Evaluating the Use and Effectiveness of Ungraded Practice Problems in an Introductory Programming Course', in *Proceedings of the Twenty-Second Australasian Computing Education Conference*, 2020, pp. 177–184.
- [52] G. Sharma, 'Computer Programming Comp103: Who Does Better?', 2019.
- [53] D. C. Leonard, *Learning theories: A to Z*. Bloomsbury Publishing USA, 2002.
- [54] J. Piaget, 'Part I: Cognitive Development in Children—Piaget Development and Learning.', *J. Res. Sci. Teach.*, vol. 40, 2003.
- [55] A. L. Archer and C. A. Hughes, 'Explicit Instruction: Effective and efficient teaching (what works for special-needs Learners)', *J. Spec. Educ.*, vol. 36, no. 4, pp. 186–205, 2011.
- [56] J. M. Fletcher et al., *Learning disabilities: From identification to intervention*. Guilford Publications, 2018.
- [57] M. Pressley and K. R. Harris, 'Cognitive strategies instruction: From basic research to classroom instruction', *J. Educ.*, vol. 189, no. 1–2, pp. 77–94, 2009.
- [58] C. Gauthier et al., 'L'enseignement explicite', 2007.

- [59] M. Bocquillon et al., 'Faut-il utiliser l'enseignement explicite en tout temps? Non... mais oui!', *Apprendre Enseigner Aujourd'hui*, vol. 8, no. 2, pp. 25–28, 2019.
- [60] B. Newmark, 'Ten principles for great explicit teaching'. [Online]. Available <https://bennewmark.wordpress.com/2017/10/07/ten-principles-for-great-explicit-teaching/>.
- [61] M. A. Sana'a et al., 'Integrated learning development environment for learning and teaching C/C++ language to novice programmers', in *2020 IEEE Frontiers in Education Conference (FIE)*, 2020, pp. 1–5.
- [62] B. Isong, 'A Methodology for Teaching Computer Programming: first year students' perspective', *Int. J. Mod. Educ. Comput. Sci.*, vol. 6, no. 9, p. 15, 2014.
- [63] L. E. Margulieux et al., 'Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples', *Int. J. STEM Educ.*, vol. 7, pp. 1–16, 2020.
- [64] R. A. Alturki and others, 'Measuring and improving student performance in an introductory programming course', *Inform. Educ.- Int. J.*, vol. 15, no. 2, pp. 183–204, 2016.
- [65] U. Nikula et al., 'A motivation guided holistic rehabilitation of the first programming course', *ACM Trans. Comput. Educ. TOCE*, vol. 11, no. 4, pp. 1–38, 2011.
- [66] A. Yadin, 'Reducing the dropout rate in an introductory programming course', *ACM Inroads*, vol. 2, no. 4, pp. 71–76, 2011.
- [67] R. R. Fragelli and T. B. O. Fragelli, 'Three Hundred: the human dimension of the method', *Educ. Em Rev.*, no. 63, pp. 253–265, 2017.
- [68] J. P. Aires et al., 'Active Methodologies in Incoming Programming Classes (Short Paper)', in *Second International Computer Programming Education Conference (ICPEC 2021)*, 2021.
- [69] Y. Ayalew et al., 'Factors affecting programming performance of first year students at a University in Botswana', *Afr. J. Res. Math. Sci. Technol. Educ.*, vol. 22, no. 3, pp. 363–373, 2018.
- [70] A. F. Elgamal et al., 'An interactive e-learning system for improving web programming skills', *Educ. Inf. Technol.*, vol. 18, no. 1, pp. 29–46, 2013.
- [71] X. Zhang et al., 'Teaching introductory programming to IS students: The impact of teaching approaches on learning performance', *J. Inf. Syst. Educ.*, vol. 24, no. 2, pp. 147–155, 2013.
- [72] M. Marimuthu et al., 'The Difference Between Attitudinal Factors Influencing Success of Commerce and Computer Science Students at a South African University', *Afr. J. Res. Math. Sci. Technol. Educ.*, vol. 24, no. 3, pp. 321–332, 2020.
- [73] U. Cakiroglu, 'Using a Hybrid Approach to Facilitate Learning Introductory Programming.', *Turk. Online J. Educ. Technol.-TOJET*, vol. 12, no. 1, pp. 161–177, 2013.

- [74] I. A. Khan et al., 'Redesign and validation of a computer programming course using Inductive Teaching Method', *PloS One*, vol. 15, no. 6, p. e0233716, 2020.
- [75] A. Edgcomb et al., 'Getting students to earnestly do reading, studying, and homework in an introductory programming class', in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 2017, pp. 171–176.
- [76] P. Maguire et al., 'Using automatic machine assessment to teach computer programming', *Comput. Sci. Educ.*, vol. 27, no. 3–4, pp. 197–214, 2017.
- [77] D. Dawar and M. Murphy, 'An Assignment a Day Scaffolded Learning Approach for Teaching Introductory Computer Programming.', *Inf. Syst. Educ. J.*, vol. 18, no. 4, pp. 59–73, 2020.
- [78] J. H. Sharp, 'Using Codecademy interactive lessons as an instructional supplement in a Python programming course', in *Proceedings of the EDSIG Conference ISSN*, 2018, vol. 2473, p. 3857.
- [79] R. Baskerville et al., 'Design science research contributions: Finding a balance between artifact and theory', *J. Assoc. Inf. Syst.*, vol. 19, no. 5, p. 3, 2018.
- [80] K. Peffers et al., 'A design science research methodology for information systems research', *J. Manag. Inf. Syst.*, vol. 24, no. 3, pp. 45–77, 2007.
- [81] A. K. Mbiada et al., 'Towards integrated framework for efficient educational software development', in *2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA)*, 2023, pp. 53–60.
- [82] B. Isong et al., 'Empirical Confirmation of Pair Programming Effectiveness in the Teaching of Computer Programming', in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2016, pp. 276–281 [Online]. Available: 10.1109/CSCI.2016.0060.
- [83] A. Luxton-Reilly et al., 'Introductory programming: a systematic literature review', in *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, 2018, pp. 55–106.
- [84] Kanika et al., 'Tools and techniques for teaching computer programming: A review', *J. Educ. Technol. Syst.*, vol. 49, no. 2, pp. 170–198, 2020.
- [85] D. Hooshyar et al., 'Flowchart-based programming environments for improving comprehension and problem-solving skill of novice programmers: a survey', *Int. J. Adv. Intell. Paradig.*, vol. 7, no. 1, pp. 24–56, 2015.
- [86] W. M. Kunkle and R. B. Allen, 'The impact of different teaching approaches and languages on student learning of introductory programming concepts', *ACM Trans. Comput. Educ. TOCE*, vol. 16, no. 1, pp. 1–26, 2016.

- [87] R. Gonzalez and A. Bjørn-Hansen, 'Web-Based Collaborative Learning in CS1: A Study on Outcomes of Peer Code Review', in *Norsk IKT-konferanse for forskning og utdanning*, 2020, no. 4.
- [88] M. Laal and M. Laal, 'Collaborative learning: what is it?', *Procedia-Soc. Behav. Sci.*, vol. 31, pp. 491–495, 2012.
- [89] M. Ayub et al., 'Utilising Pair Programming to Enhance the Performance of Slow-Paced Students on Introductory Programming.', *J. Technol. Sci. Educ.*, vol. 9, no. 3, pp. 357–367, 2019.
- [90] K. Al-Khalifa Amal and M. Devlin, 'Evaluating a Peer Assessment Approach in Introductory Programming Courses', in *United Kingdom & Ireland Computing Education Research conference.*, 2020, pp. 51–58.
- [91] C. McDowell et al., 'The effects of pair-programming on performance in an introductory programming course', in *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, 2002, pp. 38–42.
- [92] L. Williams and R. R. Kessler, *Pair programming illuminated*. Addison-Wesley Professional, 2003.
- [93] T. Linden, 'Scrum-based learning environment: Fostering self-regulated learning', *J. Inf. Syst. Educ.*, vol. 29, no. 2, pp. 65–74, 2018.
- [94] A. Carbonaro, 'Good practices to influence engagement and learning outcomes on a traditional introductory programming course', *Interact. Learn. Environ.*, vol. 27, no. 7, pp. 919–926, 2019.
- [95] C. Șerban and A. Vescan, 'Towards an Evaluation Process around Active Learning based Methods', in *2020 IEEE Frontiers in Education Conference (FIE)*, 2020, pp. 1–7.
- [96] S. M. Souza and R. A. Bittencourt, 'Motivation and engagement with PBL in an introductory programming course', in *2019 IEEE Frontiers in Education Conference (FIE)*, 2019, pp. 1–9.
- [97] A. Alammery, 'Blended learning models for introductory programming courses: A systematic review', *PloS One*, vol. 14, no. 9, p. e0221765, 2019.
- [98] H. Khosravi and K. M. Cooper, 'Using learning analytics to investigate patterns of performance and engagement in large classes', in *Proceedings of the 2017 ACM sigcse Technical Symposium on Computer Science Education*, 2017, pp. 309–314.
- [99] M. Sharma et al., 'Teaching programming to the post-millennial generation: Pedagogic considerations for an is course', *J. Inf. Syst. Educ.*, vol. 31, no. 2, pp. 96–105, 2020.

- [100] J. M. Borovina Josko, 'Mixing cognitive and affective approaches in teaching introductory programming', in *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 2*, 2021, pp. 641–641.
- [101] S. Razvi et al., 'Evolution of OAAA strategic planning: using ADRI as an analytical tool to review its activities and strategic planning', in *Proceedings of Seventh Annual International Conference on Strategic Planning for Quality Assurance and Accreditation of Universities and Educational Arab Institutions*, 2012.
- [102] S. I. Malik, 'Improvements in introductory programming course: action research insights and outcomes', *Syst. Pract. Action Res.*, vol. 31, no. 6, pp. 637–656, 2018.
- [103] C.-S. Cheah and L.-M. Leong, 'The Moderating Effect of Logic in the Learning of C++ Computer Programming Using Screencasting.', *Malays. Online J. Educ. Technol.*, vol. 7, no. 2, pp. 88–96, 2019.
- [104] O. S. Pabón and L. M. Villegas, 'Fostering motivation and improving student performance in an introductory programming course: An integrated teaching approach', *Rev. EIA*, vol. 16, no. 31, pp. 65–76, 2019.
- [105] A. Galton, 'Logic as a formal method', *Comput. J.*, vol. 35, no. 5, pp. 431–440, 1992.
- [106] T. Koulouri et al., 'Teaching introductory programming: A quantitative evaluation of different approaches', *ACM Trans. Comput. Educ. TOCE*, vol. 14, no. 4, pp. 1–28, 2014.
- [107] J. Kasurinen and U. Nikula, 'Lower dropout rates and better grades through revised course infrastructure', in *Proceedings of the 10th International Conference on Computers and Advanced Technology in Education*, 2007, pp. 152–157.
- [108] S. H. Edwards et al., 'The Relationship Between Voluntary Practice of Short Programming Exercises and Exam Performance', in *Proceedings of the ACM Conference on Global Computing Education*, 2019, pp. 113–119.
- [109] B. Hartanto et al., 'Enhancing the student engagement in an introductory programming: a holistic approach in improving the student grade in the informatics department of the University of Surabaya', *Dalam Commun. Comput. Inf. Sci.*, vol. 516, pp. 493–504, 2015.
- [110] T. H. Spangsberg and M. Brynskov, 'Code-labelling: A teaching activity encouraging deep learning in a non-STEM introductory programming course', in *2017 12th International Conference on Computer Science and Education (ICCSE)*, 2017, pp. 95–100.
- [111] A. G. S. Raj et al., 'Role of live-coding in learning introductory programming', in *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*, 2018, pp. 1–8.
- [112] T. Anastasiadis et al., 'Digital game-based learning and serious games in education', *Int. J. Adv. Sci. Res. Eng. Ijasre*, vol. 4, no. 12, pp. 139–144, 2018.

- [113] A. Soares et al., 'Teaching introductory programming with game design and problem-based learning.', *Issues Inf. Syst.*, vol. 16, no. 3, 2015.
- [114] M. Piteira et al., 'Computer programming learning: how to apply gamification on online courses?', *Comput. Program. Learn. Apply Gamification Online Courses*, no. 2, 2018.
- [115] D. Tsiotras and S. Xinogalos, 'Investigating the Perceived Player Experience and Short-term Learning of the Text-based Java Programming Serious Game "Rise of the Java Emperor"', *Inform. Educ.*, vol. 20, no. 1, pp. 153–170, 2021.
- [116] D. Hooshyar et al., 'Development and evaluation of a game-based Bayesian intelligent tutoring system for teaching programming', *J. Educ. Comput. Res.*, vol. 56, no. 6, pp. 775–801, 2018.
- [117] L. Facey-Shaw et al., 'Do badges affect intrinsic motivation in introductory programming students?', *Simul. Gaming*, vol. 51, no. 1, pp. 33–54, 2020.
- [118] H. C. Jiau et al., 'Enhancing self-motivation in learning programming using game-based simulation and metrics', *IEEE Trans. Educ.*, vol. 52, no. 4, pp. 555–562, 2009.
- [119] M. Rogers et al., 'Exploring personalization of gamification in an introductory programming course', in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 2021, pp. 1121–1127.
- [120] R. Smiderle et al., 'impac', *Smart Learn. Environ.*, vol. 7, no. 1, pp. 1–11, 2020.
- [121] V. F. RAMOS et al., 'Student and Lecturer Perceptions of Usability of the Virtual Programming Lab Module for Moodle', *Inform. Educ.*, vol. 20, no. 2, pp. 297–315, 2021.
- [122] G. Silva et al., 'Impact of Calango language in an Introductory Computer Programming Course', in *2020 IEEE Frontiers in Education Conference (FIE)*, 2020, pp. 1–9.
- [123] F. Saltan, 'The Impact of Online Algorithm Visualization on ICT Students' Achievements in Introduction to Programming Course.', *J. Educ. Learn.*, vol. 6, no. 1, pp. 184–192, 2017.
- [124] P. Brusilovsky et al., 'An integrated practice system for learning programming in Python: design and evaluation', *Res. Pract. Technol. Enhanc. Learn.*, vol. 13, pp. 1–40, 2018.
- [125] M. Virvou and S. C. Sidiropoulos, 'Collaborative tools in learning a programming language', in *2012 International Conference on E-Learning and E-Technologies in Education (ICEEE)*, 2012, pp. 162–165.
- [126] M. Stupina and V. Paniotova, 'An Educational Chatbot in a Blended Learning Environment', in *2023 3rd International Conference on Technology Enhanced Learning in Higher Education (TELLE)*, 2023, pp. 276–279.
- [127] T. Kakeshita and M. Murata, 'Application of programming education support tool pgtracer for homework assignment', *Int. J. Learn. Technol. Learn. Environ.*, vol. 1, no. 1, pp. 41–60, 2018.

- [128] A. Ferreira et al., 'SICAS2: Interactive Tool to Support Programming Learning', in *2022 International Symposium on Computers in Education (SIIE)*, 2022, pp. 1–5.
- [129] J. Cárdenas-Cobo et al., 'Recommender Systems and Scratch: An integrated approach for enhancing computer programming learning', *IEEE Trans. Learn. Technol.*, vol. 13, no. 2, pp. 387–403, 2019.
- [130] C. Kyfonidis et al., 'Block-C: A block-based programming teaching tool to facilitate introductory C programming courses', in *2017 IEEE Global Engineering Education Conference (EDUCON)*, 2017, pp. 570–579.
- [131] G. Rowe and G. Thorburn, 'VINCE—An on-line tutorial tool for teaching introductory programming', *Br. J. Educ. Technol.*, vol. 31, no. 4, pp. 359–369, 2000.
- [132] R. Hijón-Neira et al., 'Improving CS1 Programming Learning with Visual Execution Environments', *Information*, vol. 14, no. 10, p. 579, 2023.
- [133] Y. Yan et al., 'A Web Service for C Programming Learning and Teaching', in *2016 10th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, 2016, pp. 414–419.
- [134] J. Majherová and V. Králík, 'Innovative Methods in Teaching Programming for Future Informatics Teachers.', *Eur. J. Contemp. Educ.*, vol. 6, no. 3, pp. 390–400, 2017.
- [135] D. Bau et al., 'Learnable programming: blocks and beyond', *Commun. ACM*, vol. 60, no. 6, pp. 72–80, 2017.
- [136] J. M. Allen et al., 'An analysis of using many small programs in cs1', in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 585–591.
- [137] J. M. Allen et al., 'Weekly programs in a CS1 class: Experiences with auto-graded many-small programs (MSP)', in *2018 ASEE Annual Conference & Exposition*, 2018.
- [138] R. A. Alturki and others, 'Measuring and improving student performance in an introductory programming course', *Inform. Educ. - Int. J.*, vol. 15, no. 2, pp. 183–204, 2016.
- [139] P. A. Kirschner et al., 'Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching', *Educ. Psychol.*, vol. 41, no. 2, pp. 75–86, 2006.
- [140] G. Rößling et al., 'Enhancing learning management systems to better support computer science education', *ACM SIGCSE Bull.*, vol. 40, no. 4, pp. 142–166, 2008.
- [141] D. F. Rico and H. H. Sayani, 'Use of agile methods in software engineering education', in *2009 Agile conference*, 2009, pp. 174–179.
- [142] D. Turk et al., 'Limitations of agile software processes', in *Third International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2002)*, 2002, pp. 43–46.

- [143] T. Sedano et al., 'The product backlog', in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 200–211.
- [144] P. Abrahamsson et al., 'Agile software development methods: Review and analysis', *ArXiv Prepr. ArXiv170908439*, 2017.
- [145] S. Al-Saqqah et al., 'Agile Software Development: Methodologies and Trends.', *Int. J. Interact. Mob. Technol.*, vol. 14, no. 11, 2020.
- [146] J. Stapleton, *DSDM, dynamic systems development method: the method in practice*. Cambridge University Press, 1997[Online]. Availablehttps://books.google.co.za/books?hl=en&lr=&id=kp656t4p7soC&oi=fnd&pg=PR11&dq=Dynamic+Systems+Development+Method+&ots=TzlhRwWURn&sig=PnlKizSAQgda5OBBF7BHLoAtgfk&redir_esc=y#v=onepage&q=Dynamic%20Systems%20Development%20Method&f=true.
- [147] C. M. Reigeluth, 'Instructional design: What is it and why is it', *Instr.-Des. Theor. Models Overv. Their Curr. Status*, vol. 1, pp. 3–36, 1983.
- [148] J. Basque et al., 'Introduction à l'ingénierie pédagogique', *TÉLUQ Consulté À L'adresse Httppted6313v2 Teluq CateluqDownload Php*, 2014.
- [149] S. Norbert et al., *Instructional Design for Learning: Theoretical Foundations*. Brill, 2017[Online]. Available<https://nwulib.nwu.ac.za/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=1505283&site=eds-live>.
- [150] R. Heinich et al., *Instructional media and the new technologies of instruction*. Macmillan, 1989.
- [151] instructionaldesign.org, (2022, July.), Instructional Design. [Online]. Available: <https://www.instructionaldesign.org/>.
- [152] D. E. Gray, 'Doing research in the real world', 2021.
- [153] DMReview, (2007), 'Glossary. SourceMedia, Brookfield, WI'. [Online]. Availablewww.dmreview.com/glossary/a.html[Accessed: 17September2021].
- [154] P. S. Daniel and A. G. Sam, *Research methodology*. Gyan Publishing House, 2011.
- [155] B. Pasian, 'The Constructive Research Approach: Problem Solving for Complex Projects', in *Designs, Methods and Practices for Research of Project Management*, Routledge, 2016, pp. 125–136.
- [156] V. Vaishnavi et al., (Jan. 20, 2004), 'Design Science Research in Information Systems'. [Online]. Available<http://www.desrist.org/design-research-in-information-systems/>[Accessed: 17September2021].

- [157] W. Huang et al., 'A preliminary validation of Attention, Relevance, Confidence and Satisfaction model-based Instructional Material Motivational Survey in a computer-based tutorial setting', *Br. J. Educ. Technol.*, vol. 37, no. 2, pp. 243–259, 2006.
- [158] I. S. Junus et al., 'Usability evaluation of the student centered e-learning environment', *Int. Rev. Res. Open Distrib. Learn.*, vol. 16, no. 4, pp. 62–82, 2015.
- [159] K. Kwon, 'Novice programmer's misconception of programming reflected on problem-solving plans', *Int. J. Comput. Sci. Educ. Sch.*, vol. 1, no. 4, pp. 14–24, 2017.
- [160] B. Sriram, 'Educational Software Development- Users Requirement Analysis', *Int. J. Comput. Appl.*, vol. 31, no. 10, 2011.
- [161] G. Pellone, 'Educational software design: a literature review', *Australas. J. Educ. Technol.*, vol. 11, no. 1, 1995.
- [162] A. Sarasa-Cabezuelo and C. Rodrigo, 'Development of an Educational Application for Software Engineering Learning', *Computers*, vol. 10, no. 9, p. 106, 2021.
- [163] E. Hinostroza et al., 'Developing educational software: a professional tool perspective', *Educ. Inf. Technol.*, vol. 5, no. 2, pp. 103–117, 2000.
- [164] C. Bastien and D. Scapin, 'Ergonomic criteria for the evaluation of human-computer interfaces', PhD Thesis, Inria, 1993.
- [165] A. P. Costa et al., 'Development methodologies for educational software: the practical case of courseware SERe', in *International Conference on Education and New Learning Technologies (EDULEARN09)*, 2009, pp. 5816–5825.
- [166] F. J. Lage et al., 'An extended methodology for educational software design: some critical points', in *31st Annual Frontiers in Education Conference. Impact on Engineering and Science Education. Conference Proceedings (Cat. No. 01CH37193)*, 2001, vol. 1, pp. T2G-13.
- [167] L. A. Johnson and T. K. Schleyer, 'Developing high-quality educational software', *J. Dent. Educ.*, vol. 67, no. 11, pp. 1209–1220, 2003.
- [168] R. Beale and M. Sharples, 'Design guide for developers of educational software', *Br. Educ. Commun. Technol. Agency Retrieved May*, vol. 23, p. 2004, 2002.
- [169] W. P. P. Filho, 'Requirements for an educational software development process', in *Proceedings of the 6th annual conference on Innovation and technology in computer science education*, 2001, pp. 65–68.
- [170] S. Somyürek, 'The new trends in adaptive educational hypermedia systems', *Int. Rev. Res. Open Distrib. Learn.*, vol. 16, no. 1, 2015.
- [171] E. Popescu, 'Dynamic adaptive hypermedia systems for e-learning', PhD Thesis, Université de Technologie de Compiègne, 2008.

- [172] F. Hlioui et al., 'A survey on learner models in adaptive E-learning systems', in *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, 2016, pp. 1–7.
- [173] S. R. Sobral, 'Bloom's Taxonomy to improve teaching-learning in introduction to programming', 2021.
- [174] J. C. Bastien and D. L. Scapin, 'Evaluating a user interface with ergonomic criteria', *Int. J. Hum.-Comput. Interact.*, vol. 7, no. 2, pp. 105–121, 1995.
- [175] M. Richards, *Software architecture patterns*, vol. 4. O'Reilly Media, Incorporated 1005 Gravenstein Highway North, Sebastopol, CA ..., 2015.
- [176] R. M. Gagne, 'Instruction and the conditions of learning', *Psychol. Sch. Learn. Views Learn.*, vol. 1, pp. 153–175, 1974.
- [177] T. H. Spangsberg and M. Brynskov, 'Code-labelling: A teaching activity encouraging deep learning in a non-STEM introductory programming course', in *2017 12th International Conference on Computer Science and Education (ICCSE)*, 2017, pp. 95–100.
- [178] F. J. Gallego-Durán et al., 'A low-level approach to improve programming learning', *Univers. Access Inf. Soc.*, pp. 1–15, 2020.
- [179] S. I. Malik et al., 'Promoting Algorithmic Thinking in an Introductory Programming Course.', *Int. J. Emerg. Technol. Learn.*, vol. 14, no. 1, 2019.
- [180] V. L. Almstrum et al., 'Concept inventories in computer science for the topic discrete mathematics', in *ACM SIGCSE Bulletin*, 2006, vol. 38, no. 4, pp. 132–145.
- [181] I. Jung et al., 'Interactive learning environment for practical programming language based on web service', in *2016 15th International Conference on Information Technology Based Higher Education and Training (ITHET)*, 2016, pp. 1–7.
- [182] W. E. Moussa et al., 'Proposing a 3d interactive visualization tool for learning OOP concepts', in *2016 13th Learning and Technology Conference (L&T)*, 2016, pp. 1–7.
- [183] M. Ayub and O. Karnalim, 'Predicting outcomes in introductory programming using J48 classification', *World Trans.Eng. Technol. Educ. WTETE*, vol. 15, no. 2, pp. 132–136, 2017.
- [184] Y. Qian and J. D. Lehman, 'Correlates of success in introductory programming: A study with middle school students.', *J. Educ. Learn.*, vol. 5, no. 2, pp. 73–83, 2016.
- [185] M. Othman and M. Othman, 'The proposed model of collaborative virtual learning environment for introductory programming course', *Turk. Online J. Distance Educ.*, vol. 13, no. 1, pp. 100–111, 2012.

- [186] J. Bennedsen and M. E. Caspersen, 'An investigation of potential success factors for an introductory model-driven programming course', in *Proceedings of the first international workshop on Computing education research*, 2005, pp. 155–163.
- [187] R. Horváth and S. Javorský, 'New teaching model for Java programming subjects', *Procedia-Soc. Behav. Sci.*, vol. 116, pp. 5188–5193, 2014.
- [188] K. Lokesh, *Methodology of educational research*. Vikas Publishing House, 1984.