I. J. Education and Management Engineering, **2026, 1, 34-62**
Published Online on February 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijeme.2026.01.03

Exploring the Performance of Students in Programming Courses: A Study of Ghana's Sandwich Computing Education

Kofi Sarpong Adu-Manu*
Department of Computer Science, University of Ghana, Legon-Accra, Ghana
Email: ksadu-manu@ug.edu.gh
ORCID iD: https://orcid.org/0000-0001-6148-9127
*Corresponding Author

Charles Adjetey
Department of Computer Science and Information Systems, Ashesi University, Brekuso-Accra, Ghana
Email: cadjetey@ashesi.edu.gh
ORCID iD: https://orcid.org/0009-0007-8290-0457

John Kingsley Arthur
Department of Computer Science, Valley View University, Oyibi-Accra, Ghana
Email: jkarthur@vvu.edu.gh
ORCID iD: https://orcid.org/0000-0001-8557-1336

Received: 09 May, 2025; Revised: 11 July, 2025; Accepted: 13 November, 2025; Published: 08 February, 2026

Abstract: This study investigates the performance of students enrolled in programming courses offered under the Sandwich Educational System (SES) in Ghanaian universities and colleges. SES is a unique educational approach that combines academic studies with practical work experience. This study examines various teaching models employed within the SES for programming education to identify any significant relationship between teaching methods and student academic performance. The target population for this study comprised students enrolled in computing education-related programs within the SES, with a specific focus on those undertaking programming courses. A single study group of students pursuing the Bachelor of Education programme in Information Technology (B.Ed. IT) under the sandwich mode at University X was selected to ensure efficient research management in this study. Employing a mixed-methods research design, quantitative and qualitative data were collected and analysed using descriptive and inferential statistics. A survey was administered to 218 of the 357 students in the study group during the designated survey period. Additionally, a seven-year longitudinal quasi-experiment involving five different year groups in the B.Ed IT sandwich programme at University X was conducted to examine the relationship between student performance and teaching methods within SES. The findings of this study do not demonstrate a significant difference in academic performance among students taught using different teaching methods in SES. However, it is crucial to acknowledge the study's limitations, which necessitate considering the findings as insightful observations rather than as conclusive results. This study recommends enhancing students' prior exposure to programming and adopting innovative teaching methods to improve their academic performance. Future research should address the limitations of this study by utilising a more rigorous experimental design, such as a randomised controlled trial, and exploring additional factors that may influence student performance within the SES. Such endeavours would enable more robust causal inferences to be drawn.

Index Terms: Programming Language Design, Teaching Method, Grade Analysis, Programming Logic, Students' Performance, Sandwich Educational System (SES), Academic Performance, Teaching Methods, Ghana, Mixed-Methods Research, Quasi-Experiment, Randomised Controlled Trial

1. Introduction

Programming courses play a crucial role in the education of students pursuing computing-related degrees such as computer science, software engineering, and information technology. However, many students struggle with programming, leading to apprehension and a lack of motivation to continue their studies in the field. Various factors,

including teaching methods, learning styles, available resources, and the learning environment, contribute to this fear and hinder student success.

In Computer Science and related disciplines, a firm grasp of programming concepts is vital, especially for beginners. Programming requires critical thinking, systematic problem-solving, logical analysis, and construction, which presents a unique challenge to many students. It also stimulates creativity and enhances the problem-solving skills. Success in programming courses and the overall field of computer science relies on understanding programming logic, analysing problems, and constructing code effectively.

Previous studies have investigated the factors contributing to undergraduate success or failure in computer programming courses. These studies have explored failure rates in introductory programming courses [1,2], causal attributions of students taking introductory programming [3,4], motivating factors affecting students in e-learning environments [4, 5], and factors impacting students' attitudes, grades, and course outcomes [6]. Additionally, research has examined novice programmers' affective states and evaluated teaching approaches for programming courses. [7,8,9]. These studies emphasise the significance of effective teaching methodologies and learning environments in improving student performance.

Despite the potential for enjoyment in programming, many students perceive it as frustrating because of difficulties in problem comprehension, component analysis, solution design, and coding. Programming logic, analysis, and construction complexities can pose significant obstacles for students in deciphering the solutions. This leads to alarmingly high attrition rates in Computer Science programs, particularly in the initial years of study [7,10]. However, implementing appropriate pedagogical approaches can mitigate these challenges and dramatically enhance student success rates. Conversely, unsuitable or inapplicable teaching methods may increase attrition rates and hinder students' progress.

In Ghana, recent developments in Computer Science Education have given rise to online, distance, and sandwich (ODS) systems. Students in ODS programs face challenges in programming courses, which are amplified by the need to learn a large amount of material in a limited timeframe. While several studies have investigated teaching methods for programming courses in traditional college education systems [10,11,12,13,14,15], there is a notable lack of research exploring the unique context of Ghana's Sandwich Education System (SES).

This study presents a comprehensive investigation of students' performance in programming courses within Ghana's SES. The primary aim of this study was to determine whether a significant relationship exists between students' pass and failure rates in programming courses and the teaching methodologies employed by their instructors within the SES. The study also explores the views, attitudes, perceptions, and beliefs of sandwich students on various matters (especially in the context of their performance in programming courses), such as their preferred learning styles, learning environments, and opinions about the effectiveness of various teaching methods in reducing complexity in programming courses.

A mixed-methods approach was employed in this study. Quantitative data were collected through a survey to gather relevant information about teaching methodologies, learning environments, and learning styles. Additionally, qualitative data obtained through observations and surveys provided in-depth insights into the experiences and perceptions of students. The target population for this research comprised students enrolled in any computing education-related program in Ghana's SES, specifically, those taking programming courses.

We evaluated the effectiveness of various teaching methods in improving student performance. Data obtained from the experiment were analysed to assess the pass and failure rates in elementary, procedural, and object-oriented programming courses taught to the students. The study spanned seven years and involved five different year groups of sandwich students.

The findings of this study have significant implications for educators and policymakers in Ghana and other countries with similar educational systems. This study provides valuable insights into effective pedagogical approaches that can be used to enhance student outcomes in programming courses. Furthermore, the findings emphasise the importance of providing students with diverse learning opportunities and support, such as access to online resources and collaboration.

1.1 Outline

In Section 2, we provide an all-inclusive literature review and contribution to knowledge. The methodology is presented in Section 3. In Section 4, we provide and discuss the context of Ghana's Sandwich Education System. In Section 5, we present an analysis of student characteristics. In Section 6, we discuss the analysis of students' viewpoints. In Section 7, we discuss the grade analysis to understand student performance. Section 8 discusses the threats to validity. Section 9 provides future work and research directions. Recommendations for educators for the effective design of SES are presented in Section 10. Finally, we conclude the paper in Section 11.

2. Literature Review

Numerous scholars have examined the determinants of student performance in computer programming courses. Despite students' difficulties in these courses, enrolment remains high due to intense competition for ICT-related programs. Nevertheless, a sizeable proportion of students discontinue their studies during their first year, as documented

by [16]. These authors identified fourteen factors that influence students' decisions to pursue ICT courses, including interest, necessity at work, continuation of studies, salary expectations, labour market demand, future importance, field development, scholarship availability, suitability, likeability, personal development opportunities, prior experience, self-realisation potential, and other considerations. However, this study focused on students' motivation for enrolling in ICT courses rather than the reasons for their subsequent withdrawal.

Other studies have explored students' pass rates in computer science programs globally. [1, 17] estimated the pass rate to be 67% and identified significant variations in the pass, fail, abort, and skip rates. They found that smaller-sized classes of less than 30 students had a higher pass rate of about 82% than large-sized classes, which recorded approximately 69%. [18] reported that colleges had a higher pass rate of about 88% compared to universities at 66%. These pass rates were independent of the language taught. However, the authors did not explain the 30.3% failure rate.

[19] conducted a study that examined the challenges encountered by students at Tabuk University in learning programming skills and the influence of cultural practices and environmental factors on their engagement in the learning process. The authors employed a quantitative approach to evaluate students' comprehension of programming concepts. Of the 165 students surveyed, 75% received scores ranging from 0-3, indicating difficulties with fundamental concepts and an inability to solve problems using language constructs. The study further revealed that the learning environment has a more significant impact on students' outcomes in computer programming courses than the effort they invest in mastering the material.

In a 2015 study by [20], the authors investigated teaching methodologies that could address the difficulties faced by undergraduate students in computer programming at Rajamangala University of Technology. For a 16-week semester, the authors employed lectures, media, and workshops as their instructional approaches. Despite these efforts, 50% of the students (43 out of 86) received grades of D+ or D, while 13 students (15.11%) received an F during the 2013 academic year. The following year saw similar results, with 22 students (20.75%) out of a total of 106 receiving grades of D+ or D and 32 students (30.18%) receiving an F. Interviews conducted with the 110 students who received grades of D+, D, or F revealed that the student felt that programming was daunting. The authors recommended using laboratory sheets to improve student outcomes and increase participation in lectures, formative assessments, and pair projects.

[10] in their 2013 study, sought expert opinions on the reasons behind the deficient performance of students in programming courses. The findings indicate that the primary causes were the use of inappropriate teaching methodologies by some lecturers and the ineffective selection of teaching strategies by instructors. The programming and scripting languages taught in most universities include C, C++, C#, Java, HTML, and CSS, among others, and students encounter shortfalls in performance. The challenges of programming courses are not limited to first-year students but extend to sophomores, juniors, and seniors as well.

2.1 Factors Influencing Student Performance in Programming Courses

2.1.1 The Teacher Factor

While certain cognitive, psychological, behavioural, or demographic traits may serve as performance indicators, they do not directly correlate with the standard programming behaviour expected of students [18,21]. This raises the question of whether other factors contribute to student performance in programming courses. Some researchers have suggested that instructors may also play a role in determining student performance. [13] argue that it is essential for instructors first to introduce basic programming concepts and then guide students through the implementation and strategic programming processes. In contrast, [1] contend that instructors with high failure rates may not always make significant efforts to improve student performance because they perceive that programming courses inherently have high failure rates.

[26] suggest that students' difficulties in understanding programming concepts may stem from ineffective teaching strategies employed during the problem-solving and coding phases. This can lead students to perceive programming courses as complicated. Even though many teaching strategies and methodologies are misapplied, some instructors may believe that programming courses inherently have high failure rates and may not take measures to address this issue. Instead, they may continue to rely on traditional teaching methods that do not improve student performance [18]. In proposing a generic rubric for assessing or grading programming students identified that students may be marked down inappropriately if their answers do not match the marking schemes set by instructors [19]. This can discourage originality among students who may use different constructs to produce the same output, but are marked down for not providing a solution like the one provided.

Recent approaches and methodologies proposed in the literature include: 1) the design and development of an interactive multimedia simulation [27]; 2) the problem-solving method [28]; 3) a computational approach using visual programming [29]; 4) the use of engineering design concepts in a blended learning environment [30]; 5) a social network service (SNS)-integrated collaborative learning system [31]; 6) blended learning in large classes [32]; and 7) an Enhanced Teaching Model (ETM) [33]. The most recent teaching approaches recommended in the literature are listed in Table 1.

Table 1. Recent Approaches and Techniques Proposed to Improve Performance in Programming Courses

Approaches/Methods	Description	Year	Authors
Adapting the Means of Non-Programming Disciplines and Activities	Learners must engage in other academic disciplines and extracurricular activities, such as solving number-theoretic and chess endgame problems, to build their programming capacities.	2020	[22]
A Metaphor-Based Approach	Metaphors are used as an alternative to learning concepts in programming languages. Students use metaphors to explain operators, data types, variables, etc. For example, variables can be described metaphorically as a named container of data in memory and data types as the form of a memory portion where data is stored.	2019	[11]
Problem-based learning (games)	Game development was used as an alternative to increasing students' interest in programming courses. Through this approach, the authors used problem-based learning associated with game development to teach programming. Learners were self-motivated and focused, and discovered knowledge in solving programming challenges.	2018	[23]
Development of Mobile-Interfaced Machine Learning Model	The use of machine learning techniques to model students' academic performance in programming courses was proposed. Here, factors relating to attitude (both students and faculty), perception, power supply, university facilities, health, and attendance, among others, significantly impacted students' performance in programming courses.	2018	[24]
Digital story design activities	Multimedia (videos, animations, sound, music, text, and audio-visuals) was employed in learning programming. Digital story design improved learners' creativity, code writing, and problem-solving skills and increased learners' motivation.	2018	[25]

2.1.2 Novice Programmer Perspective

Programming is not a singular skill but rather a complex operation that requires students to engage in high-level intellectual processes to solve the identified problems [34]. Novice students often face significant challenges in comprehending programming concepts [35]. While novice programmers struggle to apply fundamental programming concepts, expert programmers delve deeper into these concepts, creating a vast difference between the two groups [36, 37]. It has been observed that students who have not had exposure to more than one pre-college computing course tend to perform worse than those who have had such exposure.

Students with more computer experience tend to exhibit higher levels of computer proficiency due to frequent hands-on training, which can improve their performance on computer-related tasks [12]. [38] argue that knowing a programming language does not guarantee success in an introductory programming course. In their study, 12 out of 84 students who claimed to already know how to program failed the course.

One study on Learning Edge Momentum (LEM) theory identified a factor that could determine student success or failure. According to LEM theory, humans learn best when they build upon relevant previous knowledge. This makes it easier for individuals to learn closely related concepts once they have successfully comprehended earlier concepts. The theory asserts that programming concepts are built upon known concepts and that failure to grasp a concept may lead to an inability to comprehend new concepts being introduced [39].

Another issue related to the student factor is the tendency of students to copy the work of others rather than engage in critical thinking. Students may also heavily rely on readily available online information, avoiding the need to think critically as they would have had to do without such resources [40]. [1, 17, 41] suggest that students may become frustrated and lose self-confidence when more complex content and materials are introduced due to poor programming concept teaching.

2.1.3 Learning Style of Students and Approaches to Solving Programming Problems

Considering individual differences, students employ various methods to understand concepts during the learning process. According to [42] and [43], learning styles refer to the way learners adapt to existing learning environments through stable measures such as sense of feel, mental and emotional processes to achieve expected performance in their field of study.

In this study, two prominent learning style frameworks were referenced. The Felder-Silverman Learning Style Model categorises learners along four dimensions – active/reflective, sensing/intuitive, visual/verbal, and sequential/global – and helps identify how students prefer to process, receive, and understand information. Kolb's Experiential Learning Model includes four learner types: Convergers (problem-solvers who prefer practical applications), Assimilators (theorists who value logical reasoning), Divergers (creative thinkers who prefer group discussions), and Accommodators (hands-on learners who rely on intuition). These models are relevant to programming education because they explain how students approach learning in cognitively demanding contexts such as programming. For example, convergers may excel in debugging or algorithm design, whereas assimilators may prefer the theoretical aspects of programming, such as syntax and logic.

[42] researched various learning styles that could lead to either the success or failure of students in programming courses. The researcher identified learning style modules such as Myers-Briggs, Kolb, and Felder-Silverman, emphasising the Felder-Silverman model coupled with the instrument Index Learning Style-ILS [44]. The research revealed different learning styles and their resultant performances, as shown in Table 2 [44].

Table 2. Distribution of Students by Groups of Learning Styles and Rates of Success Measured by Final Grades

Groups of Learning Styles	Total Students	Percentage Passed	Percentage Failed
Active, Sensory, Visual, Sequential	35	31	69
Active, Sensory, Visual, Global	16	25	75
Reflective, Sensory, Visual, Sequential	15	40	60
Active, Intuitive, Visual, Global	8	50	50
Active, Sensory, Verbal, Sequential	5	60	40
Active, Intuitive, Visual, Sequential	4	25	75
Reflective, Sensory, Visual, Global	4	50	50
Reflective, Intuitive, Visual, Global	4	25	75
Reflective, Intuitive, Visual, Sequential	3	100	100
Reflective, Sensory, Verbal, Sequential	2	0	100
Active, Intuitive, Verbal, Sequential	1	100	0

In Table 2, the authors present various combinations of learning styles, the number of students who adopted these styles, and their corresponding pass and failure rates. Among the combinations, Active, Intuitive, Visual, Global; Active, Sensory, Verbal, Sequential; Active, Intuitive, Visual, Global; Reflective, Intuitive, Visual, Sequential; and Active, Intuitive, Verbal, Sequential recorded relatively high pass rates of 50%, 60%, 50%, 100%, and 100%, respectively. The remaining learning style combinations resulted in relatively low pass rates. However, it is important to acknowledge that the study's results are inconclusive because of the inconsistent sample sizes evaluated for each learning style. Some sample sizes were very small, whereas others were considerably large, potentially introducing biases into the data. Moreover, it is plausible that additional factors might have influenced student performance, and learning styles merely represent one aspect of their overall academic capabilities.

[45] applied Felder's model and found that certain groups of students are more favoured when a learning style approach is adopted. Therefore, the researchers suggested that equal opportunities should be provided for all learning styles to enhance students' performance in programming courses. Similarly, [46] classified students' performance according to their learning styles using Felder's Index of Learning Styles. They identified the visual, active, and sensing learning styles as the most recurring learning styles.

In a 2014 study, [47] used Kolb's experiential learning model to analyse students' learning styles in online distance learning programs. The study identified four distinct learning styles among the students: Convergers, Assimilators, Accommodators, and Divergers. Convergers learn most effectively through practical problem-solving and decision-making. This was the most common learning style exhibited by the students in this study. However, assimilators were more focused on abstract concepts and reflective observations. These students integrated their observations into cohesive explanations. Accommodators tend to rely more on intuition than logic and are less likely to learn through experimentation and risk-taking. Divergers are characterised by their ability to generate new ideas and observe situations from multiple perspectives. These students excelled in bringing diverse ideas together.

A study by [10] investigated the effectiveness of various pedagogical approaches for teaching programming courses. The study found that 70% of the surveyed students considered Action, Collaborative, Peer-Assisted, and Problem-based learning to be highly effective. In contrast, independent and enquiry-based learning were deemed ineffective by 50% of respondents. These findings were supported by a separate study conducted by [48] and in terms of specific learning methodologies, pair programming or collaborative learning resulted in a pass rate of 72%, compared with a pass rate of 63% for solo learning. The use of media computation to teach programming concepts through the manipulation of digital media also significantly impacted performance, with one class experiencing an increase in performance from below 50% to 85% in the post-test. Furthermore, implementing peer instruction led to a 61% reduction in failure rates and outperformed standard lectures by 5% on similar final examinations [48].

2.1.4 Learning Mode and Environment

In a 2007 study conducted at Monash University, [35] investigated the learning difficulties faced by students in introductory programming courses. The researchers identified three key factors that they believed could impact student performance and failure rates: the conceptual complexity of the material, the level of feedback provided to students, and study patterns.

According to [35], to better understand these factors, they surveyed 167 students. The survey results (on conceptual complexity) revealed that students found object-oriented concepts to be the most difficult, with an average difficulty rating of 4.92 out of 7. Object-oriented design and arrays were also rated as challenging, with average difficulty ratings of 4.52 and 4.17, respectively. In contrast, algorithms received an average difficulty rating of only 1.98, indicating that students had relatively little difficulty with this concept.

The study also reported on the feedback and study patterns of students. On average, students spent approximately one hour studying on campus, 2.5 hours studying off-campus, and 0.5 hours studying with peers. These study patterns included attending 5-6 hours of weekly lectures, laboratories, and tutorials.

3. Methodology

This study utilised a mixed-methods approach to comprehensively investigate the factors influencing students' academic performance within the Sandwich Education System (SES) in Ghana. Data were collected through surveys, observations, and experiments. Due to variations in SES implementation among universities and colleges in Ghana, this research was conducted within a single study group. This study serves as a preliminary investigation into this topic, providing valuable contextual understanding and laying the foundation for future in-depth research. It is important to note that the findings should be interpreted as insightful observations, rather than conclusive results. To enhance the robustness of the analysis, we performed an additional longitudinal tracking analysis of student performance data across individual cohorts. Although specific individual identifiers were not initially assigned, each cohort (year group) was delineated, allowing for a cohort-based longitudinal assessment. This enabled us to observe and analyse changes and trends in academic performance from entry-level courses to senior-level programming courses within the same student groups over time.

3.1 Target Population

The target population for this study consisted of students enrolled in computing education-related programs within the SES in Ghana, specifically those taking programming courses. To ensure the manageability of the research, the study focused on a single institution, University X. The participants were students enrolled in the Bachelor of Education program in Information Technology (B.Ed. IT) in the sandwich mode.

3.2 Survey Design

A survey was administered to 218 of the 357 students at University X during the designated survey period. Participants were drawn from all program levels. Before taking the survey, participants were provided with a clear understanding of its importance, and any misunderstandings or confusion were promptly resolved. Participation in the survey was voluntary and anonymised, and participants were required to sign an honesty pledge. The survey aimed to gather information from students on various aspects, including their motivations for choosing the sandwich mode, perspectives, attitudes, beliefs, and opinions related to learning environments, learning styles, teaching methods, instructor empathy, and recommendations for potential improvements in programming courses. The survey instrument comprised Likert-scale items, checklists, multiple-choice questions, rankings and open-ended questions.

3.3 Data Analysis for Survey Results

The survey data, comprising quantitative and qualitative responses, were systematically organised and analysed using spreadsheet software, specifically Microsoft Excel. Independent observations of the students and their curriculum provided additional context for interpreting and discussing results. Thematic analysis was applied to the qualitative responses to identify the salient themes and patterns. To ensure a more balanced mixed-methods approach, additional qualitative analyses were performed on the collected student perceptions and observational data. Qualitative responses were subjected to thematic analysis, which identified recurring themes regarding students' experiences, attitudes, beliefs, and challenges encountered during programming courses. Key qualitative findings included students' specific difficulties in understanding programming logic, their preferences for particular instructional methods, and their perceptions of instructor empathy and course pacing. Representative student quotes were selected to illustrate these themes, providing deeper insights into students' experiences within the SES. The findings were also compared with the existing literature to determine any similarities or discrepancies with the results of other published studies.

3.4 Quasi-Experiment Design

A quasi-experimental design was used to examine the relationship between students' pass and failure rates in programming courses and the teaching methodologies utilised within the SES. The study involved five classes (or year groups) of sandwich students from the B.Ed. Instructors were assigned to courses by the university and were free to choose their teaching methods, maintaining a natural classroom environment and avoiding disruption to teaching and learning activities.

3.5 Data Analysis for Quasi-Experiment Results

The data collected from the quasi-experiment primarily consisted of quantitative data. The data were organised and analysed using Microsoft Excel. Various data visualisation techniques and descriptive statistics were employed to gain insights into the data, creating graphs and tables for the grade trend analysis. Due to the relatively small sample size, a non-parametric hypothesis test, specifically the Kruskal-Wallis test, was selected for inferential statistics to examine any significant differences in student performance among the different teaching methodologies. The Kruskal-Wallis H test was applied to compare grade distributions across the teaching methods used in the five-year groups. The test examined whether differences in students' academic performance (classified as high pass, low pass, or fail) were statistically significant across instructional strategies (e.g. pair programming, task-based, problem-based, and lab

practice). This inferential analysis complemented the descriptive statistics by evaluating whether the observed performance differences across groups could be attributed to chance.

3.6 Ethical Considerations

Ethical considerations were of utmost importance throughout the study. The research adhered to all relevant protocols and guidelines for researching the student population at University X. Informed consent was obtained from all participants, and their anonymity and confidentiality were strictly ensured.

3.7 Limitations

This study has several limitations that should be acknowledged.

The focus on a single institution may restrict the generalisability of the findings to other SES contexts. Reliance on self-reported data through surveys may introduce response bias. The quasi-experimental design limits control over independent variables, such as teaching methodologies, potentially impacting the internal validity of the study.

This study does not explicitly control for or quantitatively assess several critical external factors that may influence student performance, such as individual motivation, workload outside the classroom, family responsibilities, and access to learning resources (e.g. stable Internet and computing devices). These factors are particularly relevant in the SES context, where most students are full-time professionals balancing the academic and occupational demands of their jobs.

Moreover, the study's reliance on surveys and self-reported data introduces the potential risk of response bias, as students may not always provide fully honest or accurate answers, particularly regarding sensitive topics such as their struggles with programming. To address this limitation, future research should integrate alternative data collection methods such as observational assessments, interviews, or performance metrics derived directly from students' programming assignments or examinations. Another significant limitation of this study was the lack of a clearly defined control group. This study focused exclusively on students enrolled in the Sandwich Education System (SES) without parallel groups experiencing traditional or alternative educational approaches. Consequently, direct comparisons to evaluate the unique impacts or benefits of SES over other educational systems could not be made.

Additionally, while the study identifies various teaching methods utilised by instructors, it does not provide a detailed explanation of how each technique was consistently applied across all the student groups. Variability in the implementation and execution of these instructional approaches could introduce inconsistencies, potentially affecting the reliability of the results. While this study identifies multiple instructional methodologies employed in programming courses, it does not explore the differential impacts these methods may have on specific student groups based on their distinct learning styles or levels of prior programming experience. Such a detailed analysis could provide critical insights into customising pedagogical approaches to optimise learning outcomes.

Addressing these limitations in future studies is crucial for refining the research methodology and enhancing clarity, rigour, and transparency.

4. Ghana's Sandwich Education System

Students can undertake computing-related courses (that is, Computer Science and Information Technology) in different study modes, including regular/full-time, sandwich, and distance/online courses at many universities and colleges in Ghana. Ghana's sandwich education refers to education in which students move into various tertiary institutions during vacation periods when full-time students' academic activities are not in session. The term "sandwich" describes this educational model because it fits between regular academic sessions. Sandwich students are in accelerated programs, and education is offered during vacations. Distance and sandwich education involve practical work experience and academic study in an e-learning environment.

The sandwich system is widely adopted in Ghanaian universities, particularly for professional programs such as education, nursing and engineering. It is designed to cater to the needs of working professionals who wish to upgrade their qualifications while maintaining their employment. In the sandwich system, students typically engage in intensive coursework during their work breaks. This could occur during long vacations, weekends, or other designated time frames. Sandwich programmes typically last two to four years, depending on the level and field of study, with varying schedules and arrangements across universities and departments.

One of the primary benefits of the sandwich education system is its flexibility and adaptability. This allows students to balance their academic pursuits with their professional responsibilities. Working professionals, such as teachers and healthcare workers, can continue their employment while pursuing higher education, minimising potential career disruptions.

Moreover, the sandwich system often offers a more practical approach to education than the traditional system. It emphasises hands-on learning, real-life case studies and experiential activities. This practical orientation helps students bridge the gap between theory and practice, enabling them to apply their knowledge and skills directly to professional settings.

The sandwich education system in Ghana also aims to make education accessible and affordable. It provides an opportunity for individuals without full-time studies because of financial constraints or work commitments. Compared to a full-time program, the shorter duration of the sandwich program reduces the overall cost of education and allows students to earn their degrees more quickly. Table 3 compares the typical academic schedules for regular and sandwich Bachelor of Education in Information Technology (B.Ed. I.T) programmes at University X in Ghana.

Table 3. Typical Academic Schedule for Regular and Sandwich B.Ed. I.T Programmes at University X in Ghana

Study Mode	Semester/Session	Duration	Total Number of Weeks
Regular	Semester 1	3-4 months (16 weeks)	32
	Semester 2	3-4 months (16 weeks)	
Sandwich	Session 1	5 weeks	11
	Session 2	3 weeks	
	Session 3	3 weeks	

The sandwich programme offered by many universities and colleges in Ghana fits perfectly into the academic calendar (during the vacation periods) of the basic school system, making it ideal for basic school teachers (BSTs) who want to upgrade their skills while working. This program allows them to take time off to study and gain relevant skills for career advancement and possible promotion. BSTs typically have vacations lasting between three and five weeks. However, the sandwich programme can be more demanding and stressful because of the limited time to study the same content taught in a conventional educational setting. Consequently, students must work hard within the short time frame provided for their studies.

Table 4 presents the distribution of hours spent on academic work over a semester or session for the B.Ed. IT program at University X, comparing the regular and sandwich modes. According to the data, regular students spend approximately 32 weeks (about seven and a half months) on academic work per academic year, whereas sandwich students spend approximately 12 weeks (about three months). Regular students dedicate time to academic work, whereas sandwich students attend lectures during occupational leave and vacations. Despite the difference in time allocation, sandwich students must cover the same course content and credit load as regular students, with some exemptions.

The condensed schedule of the sandwich program may result in students having to memorise course content without fully understanding it; for example, in the B.Ed. IT sandwich program at University X, sandwich students typically have three weekly meetings per course, compared to the single weekly meeting for regular students, as shown in Table 4. The total lecture time was the same for both programs, but the sandwich program had more frequent lectures with shorter durations. This could benefit students who learn better in shorter and more focused sessions. However, it is essential to note that the sandwich program may also have less contact time with lecturers and tutors than the traditional program. This is because the program is shorter, so students have less time to meet their lecturers and tutors outside of lectures. Additionally, sandwich students do not benefit from the additional two-hour tutorial sessions enjoyed by those in the conventional model.

Table 4. Distribution of Hours Spent per Semester or Session in Regular and Sandwich B.Ed. IT Programmes at University X

Semester/Session	Number of weeks (W) for teaching	Number of meetings in a week (M)	Total meeting times, T (T = W*M)	Lecture Hours, L (L = T*3)	Tutorials, TR (TR = T*2)	Total hours spent, TC (TC = L+TR)
Regular Mode						
Semester 1	12	1	12	36	24	60
Semester 2	12	1	12	36	24	60
Sandwich Mode						
Session 1	4	3	12	36	-	36
Session 2	2 weeks, 2 days	3	12	36	-	36
Session 3	2 weeks, 2 days	3	12	36	-	36

Academic schedules within SES are typically rigorous. Students are expected to effectively assimilate the material presented during the day, complete various assignments and projects, and prepare adequately for the next day's activities. Table 5 presents the lecture schedule for the first session of the 2018/2019 academic year in the B.Ed. IT sandwich program at University X. Notably, students enrolled in computer programming courses are assessed using similar metrics regardless of their mode of learning (i.e., sandwich, conventional, distance, or online).

Despite the similarities in the evaluation metrics, the course duration varies across different study modes. This variation necessitates that lecturers adjust their teaching methods and learning strategies. Consequently, students must adapt to various learning styles to effectively comprehend course material, particularly in programming courses. However, many students continue to struggle with understanding the logic of programming. This study focuses on the teaching and learning methods employed in the sandwich mode of study. The target population for this research comprised students enrolled in any computing education-related program in Ghana's Sandwich Education System who were taking programming courses.

Table 5. Sample Sandwich Schedule for B.Ed. IT at University X (First Session, 2018/2019). Programming-oriented courses are shown in bold.

Days	6:00 AM - 9:00 AM	9:00 AM - 12:00 PM	12:00 PM - 3:00 PM	3:00 PM - 6:00 PM
Level 200				
Sunday	-	**Elements of Programming**	Educational Psychology	Principles and Practice of Education
Tuesday				
Thursday				
Monday	Introduction to Educational and Instructional Technologies	Information Technology Foundation I	Principles of Christian Faith	-
Wednesday				
Friday				
Level 300				
Sunday	**Programming With Java**	School Organisation, Administration and Supervision and General Principles	Methods of Teaching	-
Tuesday				
Thursday				
Monday	Introduction to Biblical Foundation of Ethics	Data Communication and Computer Network I	Research Methods	Information Technology System
Wednesday				
Friday				
Level 400				
Sunday	**Advanced Programming with Application Development**	Distributed Computing	-	Educational Courseware and Instructional Materials Development I
Tuesday				
Thursday				
Monday	-	Enterprise Info. Security	-	Senior Research Project 1
Wednesday				
Friday				

4.1 Case Study: Student Motivation and Teaching Methods in the B.Ed. IT Sandwich Program at University X

In many universities and colleges, non-traditional modes of education, such as online, distance, sandwich, and weekend programs, are more affordable than traditional systems of education. We surveyed 218 students from the SES B.Ed IT program at University X to investigate various aspects of programming students' enrolment in the SES. As part of this survey, we aimed to understand their motivations for choosing a sandwich program in computing education. The participants, representing 61% of the total 357 students in the program, were distributed across academic levels: 20.2% sophomores, 38.1% juniors, and the remainder were seniors. Admission to the program, which lasted three years, required a diploma certificate from the College of Education in Ghana and began at the sophomore level.

According to our survey results, most respondents (97.7%) selected the sandwich mode because of financial constraints. A small percentage (2.3%) cited other reasons, such as the opportunity to continue their education while maintaining employment and positive recommendations from colleagues who had previously benefited from the SES Program. Additionally, 95.8% of the students were working-class individuals, mainly teachers employed by the Ghana Education Service, who wanted to enhance their knowledge and skills while still in service.

Practical courses are a significant part of the B.Ed IT program, and students must allocate time outside the class for assignments and preparation for upcoming lessons. To accommodate the sandwich model, flexible teaching methods include problem solving, case studies, group discussions, project-based learning, laboratory activities, simulations, practical demonstrations, and peer learning. The instruction is tailored to student needs, and course materials are provided in advance. Students study at home and attend classes for 3-5 weeks during their vacation. Face-to-face interactions between lecturers and students are encouraged to promote active participation and free exchange of ideas.

Due to limited practice time outside the class, lecturers utilise practical projects as instructional tools, and students must complete these projects before the subsequent class. The concepts taught in the class are designed to facilitate the completion of these projects. Lecturers also employ different methodologies and strategies to promote student comprehension of programming concepts, as many students admitted to the B.Ed. IT programs have little or no prior programming experience. Lecturers are expected to identify students who require additional support and provide opportunities for improvement outside the class. For example, senior students with sufficient knowledge and skills could be assigned to mentor their peers. In addition, lecturers must clearly explain programming concepts during class to promote understanding. Hands-on experience is considered a superior approach to teaching programming, as it enables learners to rapidly grasp concepts.

5. Student Profiling: An Analysis of Student Characteristics

In our survey, we investigated various demographic characteristics of the participants. The data collected included age, gender, IT proficiency, and prior experience with computing education. Additionally, we gathered information on their preferred teaching methodologies, learning environment, and learning styles, as well as their attitudes, behaviours, and beliefs about programming.

According to our survey results, most (96.8%) respondents reported having a solid background in Information and Communication Technology (ICT) but not programming. A small percentage (3.2%) reported forgetting their previous ICT knowledge and expressed uncertainty about how to begin learning programming. Of the total respondents, 95.4% reported having experience with programming outside the school system. Most of these respondents had experience

with HTML and Visual Basic, while a small percentage had experience with the basics of Java and C# programming languages.

In terms of age distribution, 3.2% of respondents were below 20 years old, 15.6% were between 20-25 years old, 61.0% were between 26-30 years old, 14.2% were between 31-35 years old, 5.0% were between 36-40 years old, and 0.9% were above 40 years old. As previously mentioned, 95.8% of students enrolled in the Sandwich program were teachers in the Ghana Education Service, with the majority (61.0%) falling between the ages of 26-30. These respondents typically worked for 1 to 3 years as teachers after completing a Diploma in Basic Education at a College of Education in Ghana.

Our study also found that only 9.2% of the respondents were female, while the remaining 90.8% were male. The low percentage of female representation observed in this study is consistent with trends reported in the literature on computing education in general. This remarkable gender imbalance highlights a critical equity gap in computing education within the SES. The underrepresentation of women may be influenced by a combination of cultural norms, limited exposure to computing fields at earlier educational stages, and a lack of targeted encouragement for women to pursue technology-focused careers. Failing to address this imbalance may perpetuate systemic barriers to computing participation and innovation. For example, [49] and [50] reported similar findings in their studies of computing science-related programs. These findings highlight the need for continued efforts to address the underrepresentation of women in these fields.

In the survey, 87.6% of respondents reported utilising online resources to learn about programming since their enrolment in the B.Ed. IT program, while the remaining 12.4% did not. Of the students who admitted to using online resources, 90.8% relied primarily on video tutorials, while the remaining 9.2% relied on resources other than videos. This finding is consistent with the literature on novice programmers, including studies by [26, 36, 37] which suggest that they often face challenges in understanding programming concepts. Video tutorials can be a helpful tool for novice programmers because they can provide a visual representation of what is being explained.

Additionally, video tutorials can be accessed anytime and from any location, which is convenient for students juggling multiple commitments. The use of online resources by students in the B.Ed IT program is a positive development, suggesting that they are taking advantage of the many available resources. Online resources can help students learn at their own pace and tailor their learning to their individual needs. Additionally, online resources can provide students access to experts in the field, which can help them obtain the support they need to succeed.

This study evaluated the effectiveness of various teaching approaches in reducing complexity and facilitating learning in programming courses. To assess the impact of these approaches, we asked the students to provide feedback on their perceived effectiveness. The results of this assessment are presented in Table 6.

Table 6. Students' Views on Instructional Methods' Effectiveness in Decreasing Programming Courses' Complexity

Instructional method	**(%) Agrees**	**(%) Disagrees**
Problem-based Learning	93.6	6.4
Demonstrations	90.4	9.6
Lab tutorials	86.7	13.3
Lecturer-led discussions	97.7	2.3
Free-flow discussions	91.7	8.3
Small group discussions	93.1	6.9
Group discussions	93.1	6.9
Pair programming	54.6	45.4
Group assignments	92.2	7.8
Guided discovery	75.7	24.3
Case studies	73.4	26.6
Practical illustrations	80.3	19.7

The survey results indicate that most students believe that teaching approaches such as problem-based learning, demonstrations, lab tutorials, lecturer-led discussions, free-flow discussions, small group discussions, group discussions, guided discovery, case studies, and practical illustrations reduce the complexity and difficulty of programming courses. However, pair programming received the least amount of support. These findings suggest that students prefer interactive, collaborative, and hands-on teaching approaches and appreciate opportunities to discuss and explore programming concepts with their peers and instructors. While these findings are preliminary and require further validation through well-designed experiments, they have significant implications for designing and delivering programming courses. Instructors should consider incorporating a variety of teaching approaches that are likely to engage students and facilitate effective learning.

Subsequent investigations in the survey concerning assessment strategies in programming courses indicated that 91.2% of respondents believed that setting clear goals in programming courses increased students' understanding and promoted innovation and confidence in problem-solving. The remaining students had different opinions. However, the

students unanimously agreed that lecturers' mastery of the subject area increased their confidence in the fairness of assessments. This is because the different solutions students provide to programming challenges are less likely to be marked incorrectly by knowledgeable lecturers. Additionally, 97.2% of respondents agreed that lecturers promoting critical thinking enhance their programming originality, while 2.8% disagreed. These findings vary from those of [19], who asserted that students are often marked down inappropriately if their answers do not match the instructors' marking schemes. In such cases, students' originality is not encouraged because they may use different constructs to produce the same output, but still be marked down for not providing a solution like the one provided. The discrepancy between the two findings may be attributed to the fact that when instructors utilise methods that foster critical thinking and innovation, their extensive expertise and proficiency in programming allow them to recognise and value their students' originality.

In our study, we determined that the predominant learning style among the students was assimilators, accounting for 36.2% of the sample. Assimilators, based on Kolb's model, prefer structured learning and abstract conceptualisation, which aligns with programming tasks that require deep logical thinking and the ability to synthesise rules and algorithms. Understanding the distribution of learning styles enables instructors to match instructional strategies with student preferences more effectively, thereby enhancing comprehension and retention in programming courses.

This suggests that many students prefer to learn through observation, experimentation, and reflection, indicating a preference for abstract conceptualisation over concrete experiences. Assimilators focus on concepts and ideas rather than emotions or feelings and are interested in following logical theories to derive meaning through practice. This learning style is particularly well suited to science and information technology students. These results may be interpreted as a consequence of most students having limited prior programming experience.

Programming courses often employ a problem-solving approach, requiring students to have a strong foundation of knowledge and understanding to effectively solve problems. This study found that the second-largest group of students was convergers, accounting for 27.5% of the sample. Convergers gather information and apply theories and ideas to find practical solutions to problems. They are task-oriented learners who focus less on interpersonal issues than their counterparts. Divergers, who accounted for 20.2% of the sample, preferred group learning. They often use their imagination to solve problems after gathering information from different perspectives and are comfortable in situations that require idea generation. Divergers tend to learn best by observing rather than by doing.

Moreover, a smaller % of students (16.1%) exhibited an accommodator learning style. Programming courses often involve logical thinking and problem-solving, which may not align with the preferences of accommodators, who tend to learn through hands-on experience and intuition. While this learning style may present challenges in programming courses, it is essential to note that, with appropriate teaching methods and support, accommodators can still excel in these courses.

The learning environment is critical in determining students' success in a course. A supportive and enabling environment can facilitate student engagement and improve their academic performance. Our study investigated the learning environment provided by lecturers in programming courses. Our results indicate that most respondents (91.7%) felt that they could openly express their ideas in class, while 8.3% disagreed. Additionally, 89.4% of respondents reported having equal opportunities to participate in class, while 10.6% had a different view. These findings suggest that equal opportunities and an open environment can enhance student motivation and performance in programming courses.

In contrast to our findings, [51] reported that over 70% of their students rated lecturer interaction as weak due to the pandemic. This discrepancy may be attributed to differences in the learning environments and teaching methods between our study and that of [51,52]. It is important to consider the specific context and factors that may influence student-lecturer interactions and the effectiveness of teaching methods.

6. Students' Viewpoints and Grade Trend Analysis

6.1 Analysis of Student's Viewpoints

Our survey delved deeper into students' perceptions of their lecturers' acknowledgement and comprehension of their life experiences and how these relate to coursework. The findings revealed that most students (72.9%) believed their lecturers were cognizant of their coursework challenges, while a minority (27.1%) did not share this viewpoint. In addition, we assessed students' opinions on the appropriateness of time allocation for teaching programming courses and the adequacy of teaching contact hours. The results indicated that most respondents (88.1%) agreed that the time allocation was suitable, while a small minority (11.9%) disagreed on this point. Similarly, most respondents (63.3%) considered the teaching contact hours to be sufficient, although a considerable minority (36.7%) expressed the opposite view.

Recognising the importance of acknowledging and addressing students' diverse perspectives and needs, lecturers should strive to foster an inclusive and supportive learning environment. This can be achieved through various means, such as creating a welcoming atmosphere, being receptive to student feedback, providing regular opportunities for students to seek assistance and ask questions, and offering flexible learning options, including online resources and tutorials. By implementing these measures, lecturers can enhance student success.

Furthermore, lecturers should actively recognise and appreciate students' unique life experiences. This awareness can inform and enhance teaching practices by enabling lecturers to tailor their instruction to meet students' needs. For instance, lecturers can consider the varied learning styles of their students, demonstrate cultural sensitivity, and provide opportunities for students to share their experiences with them. By taking these steps, lecturers can cultivate a more inclusive learning environment in which all students feel valued and respected.

Additionally, we evaluated students' satisfaction levels in the sandwich program and solicited their recommendations for pedagogical approaches and best practices to improve their performance in programming. Our findings revealed that 55.5% of respondents were satisfied with the program, attributing their satisfaction to factors such as the ability to continue their education while working, the opportunity to prepare for classes without workplace distractions, and increased motivation to work harder. However, 44.5% of respondents reported dissatisfaction with the program, citing concerns such as limited time to complete coursework, high-stress levels, incomplete coverage of course content, compressed schedules, and limited opportunities to apply learned concepts due to time constraints.

In open-ended responses, several students attributed these challenges to competing demands between work, family, and school. For example, one student wrote, "I often struggle to balance my classroom teaching job with evening programming assignments, which affects my concentration." Another noted, "Not having reliable internet or a personal laptop makes completing assignments on time very difficult." These responses highlight how external factors, including motivation, time constraints, and limited access to resources, can significantly impact student performance regardless of the teaching method employed.

The respondents provided valuable suggestions for improving teaching approaches and techniques in SES programming. These recommendations included incorporating laboratory practice, adopting student-centred teaching, implementing project-based instruction, integrating problem-solving and case studies, promoting increased lecturer-student interactivity, assigning group-based projects, encouraging peer teaching, utilising simulation-based demonstrations, and facilitating small group discussions.

Building on our initial analysis, we conducted an in-depth qualitative exploration of the open-ended survey responses and observational notes. The analysis revealed several salient themes that emerged.

- **Perceived Difficulty and Stress:** Students frequently expressed stress related to the accelerated learning schedule and the intensity of the programming content, citing examples such as, *"the short time frame makes grasping the logic extremely stressful,"* and "*I sometimes feel overwhelmed because we cover too much too fast."*
- **Teaching Method Preferences:** Qualitative data consistently highlighted students' appreciation for interactive and demonstrative teaching approaches. A representative comment was, *"I learn better through practical demonstrations; seeing the instructor code live helps."*
- **Instructor Support and Empathy:** Many students emphasised the critical role of instructor empathy, such as: *"Knowing that the instructor understands our struggles as sandwich students motivates us to work harder"* and *"We need lecturers who do not just teach but also support us emotionally".*

These qualitative insights significantly complement our quantitative analysis, offering a richer and more nuanced understanding of the SES educational experiences.

The students' recommendations for best practices in teaching programming courses within the SES program include providing pre-course learning materials to students, fostering hands-on experience through practical exercises, scheduling longer sessions for in-depth exploration of concepts, building upon students' existing knowledge when introducing new concepts, instituting open forums for addressing challenges, emphasising fundamental programming elements, allocating time for practical sessions, administering practical examinations, and providing access to Internet resources. By implementing these recommendations, lecturers can create an effective and supportive learning environment that enhances student outcomes and satisfaction in the future. Lecturers must consider and implement these recommendations to continuously improve the quality of education and meet their students' evolving needs.

6.2 Grade Trend Analysis

Teaching programming courses to students with little or no background in computing within short sessions is a challenge for facilitators. Facilitators must adopt appropriate teaching methods to effectively impart knowledge and foster an appreciation for the art of programming in the sandwich mode of the B.Ed. I.T programme at University X, students are admitted to the sophomore level and introduced to general programming principles such as programming logic, pseudocode, algorithms, flowcharts, selection, and iteration in the course "Elements of Programming," which lasts for at least five weeks. During the second and third sessions of the same level, students are introduced to procedural and object-oriented programming using C and C++, respectively. These sessions lasted three weeks each and focused on expressions, control structures, iterations, functions, arrays, and classes.

At the junior level, students learn object-oriented programming using Java, focusing on advanced classes, inheritance, polymorphism, and others. Finally, students take their last programming course at the senior level using C# to build desktop applications. Over the years, instructors assigned to teach programming have adopted multiple approaches and strategies to achieve the course objectives. This section presents a grade analysis of students who took

programming courses from their sophomore level (entry) to their senior level (exit) across five-year groups. Table 7 presents the grading scheme used for grading students. We classified grades as follows: high pass (Grades A to B+), low pass (grades B to C), and fail (grades C- to F).

Table 7. Grading Scheme

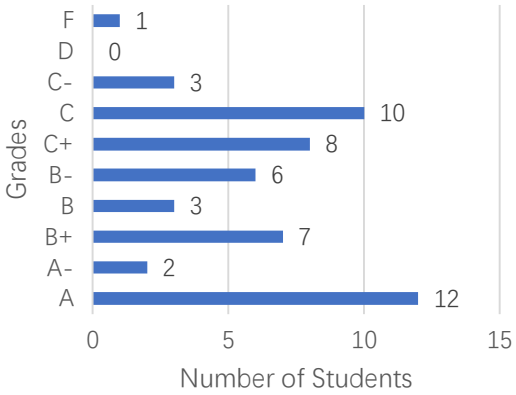
Score	Grade	Interpretation
80 − 100	A	High Pass
75 − 79	A-	
70 − 74	B+	
65 − 69	B	
61 − 64	B-	Low Pass
56 − 60	C+	
50 − 55	C	
45 − 49	C-	Fail
40 − 44	D	
0 − 39	F	

We conducted a quasi-experiment to gather data on student performance for our analyses. This approach was deemed most suitable because the participants in our study were students enrolled in a fixed-schedule learning program. Controlling the experiment's variables could have disrupted their teaching and learning experience. Additionally, the assignment of lecturers to courses was beyond our control, and some lecturers may have preferred certain teaching approaches. As such, we opted to observe the naturally occurring differences in teaching methodologies employed by different instructors, expecting that the quasi-experiment would provide a good approximation of a randomised controlled trial and yield valuable insights. In this analysis, we examined the teaching methods adopted by each instructor and assessed their impact on students' academic performance. To determine whether these differences were statistically significant, we conducted a Kruskal-Wallis H test on the grade distributions across various teaching methods. The results showed a statistically significant difference in student performance among at least one of the teaching methodologies ($H(4) = 10.86$, $p = 0.028$). This suggests that certain instructional approaches may be more effective than others in the SES context, validating the need for pedagogical alignment with learners' requirements. Although this test does not identify which specific methods differ, the significance indicates that teaching methodology has a measurable effect on outcomes, warranting further controlled research.

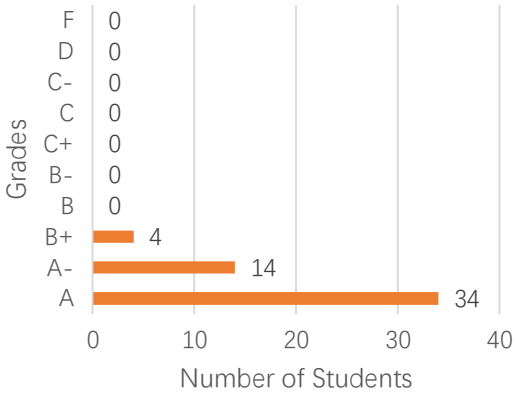
The independent variable was teaching methodology, and the dependent variable was students' academic performance. Our null hypothesis (H0) is that "there is no significant difference in students' academic performance between the different teaching methods." The alternative hypothesis (Ha) is that "there is a significant difference in students' academic performance between the different teaching methods." Tables 8-12 present the programming languages taught, the list of instructors, their teaching methods, and the number of students who sat for exams. From 2013 to 2019, our experiment involved eight distinct instructors who taught various programming courses. The lecturers employed various pedagogical approaches in their instruction, utilising a combination of methods to facilitate learning in their respective classes. The success rates for each programming language are presented in Fig. 1-8.

Table 8. Teaching Methodologies (Year-Group 1)

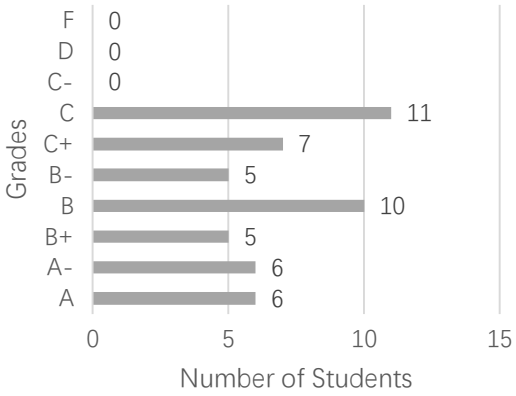
Programming Language	Instructor	Teaching Method	Number of Students	Percentage Passed
Elements of Programming (with Raptor)	A	Teaching by Task and Pair/Group Programming	52	92.31
Programming with C	A	Teaching by Task and Pair/Group Programming	52	100
Programming with C++	B	Puzzle-based, Problem-based Learning, Laboratory Practice, Teaching by Task	50	100
Programming with Java	C	Teaching by Task, Peer Tutoring, and Group Programming	44	100
Advanced Programming (C#)	D	Problem-Based Learning, Laboratory Practice, and Peer Tutoring	46	97.83



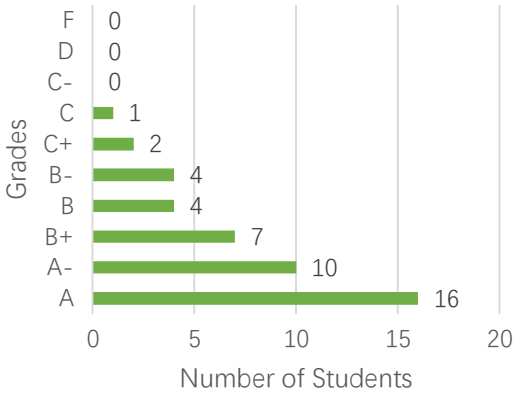
(a) Elements of Programming, Session 1 2013/2014



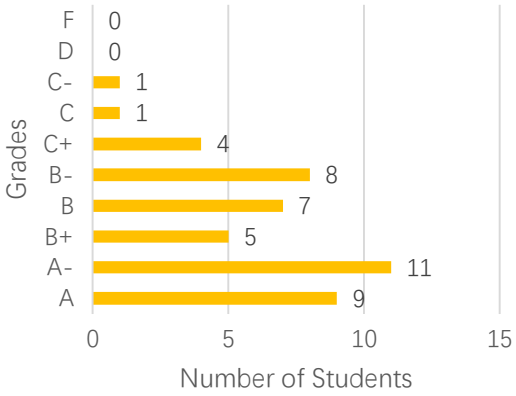
(b) Programming with C, Session 2 2013/2014



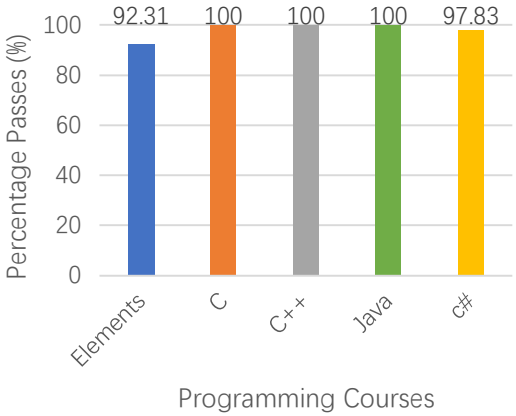
(c) Programming with C++, Session 3 2014/2015



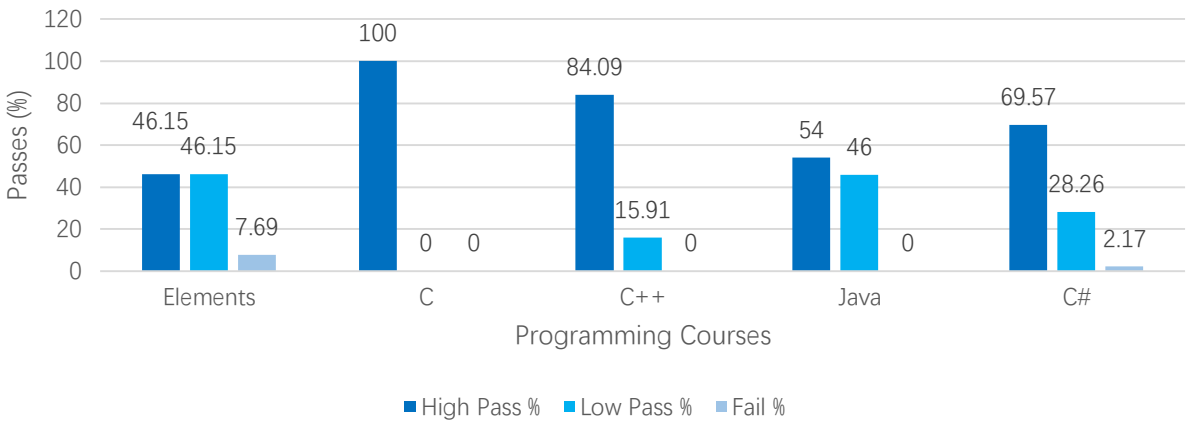
(d) Programming with Java, Session 1 2013/2014



(d) Programming with C#, Session 1 2015/2016



(e) Pass Rates in Programming Courses from Year-Group 1



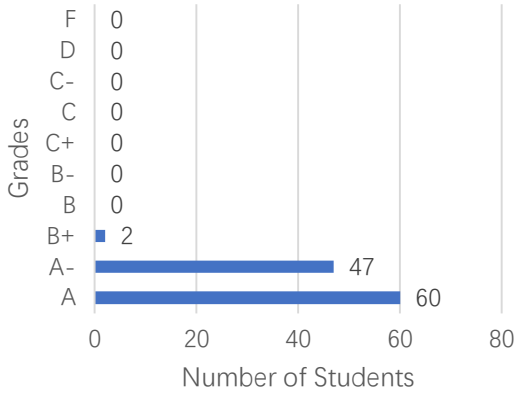
(g) Grade Distribution in Programming Courses from Year-Group 1

Fig. 1. Grade Trends of Students in Programming Courses for Year-Group 1

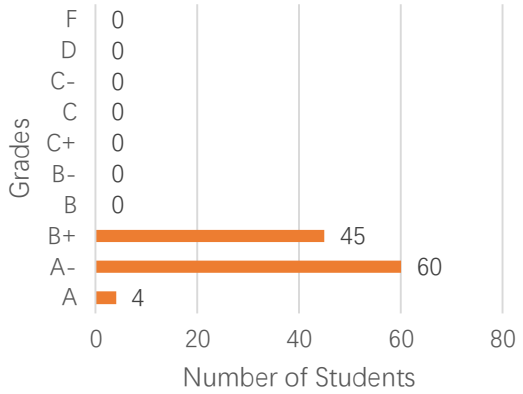
The first cohort of sandwich students was instructed by four lecturers who employed various teaching methods. The students' performance results are summarised in Fig. 1g. In the course Elements of Programming, 46.15% of the students achieved a high pass, 46.15% achieved a low pass, and 7.69% failed the course. In the Programming with C course, all students achieved a high pass. In the Programming with C++ course, 84% of students achieved a high pass, and 15.91% achieved a low pass. Of the 44 students who proceeded to take object-oriented programming courses, 54% achieved a high pass in Java, and the remaining 46% achieved a low passing score. In their final year, 46 students studied the C# programming language; 69.57% achieved a high pass, 28.26% achieved a low pass, and 2.17% failed.

Table 9. Teaching Methodologies (Year-Group 2)

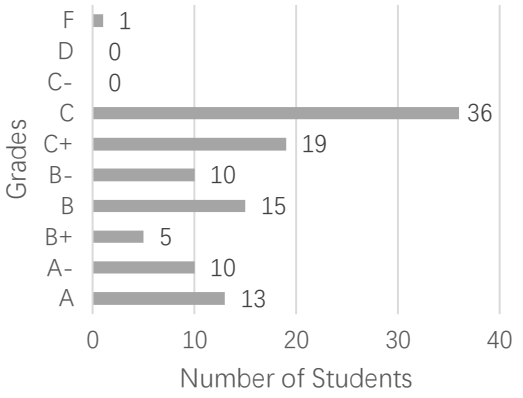
Programming Language	Instructor	Teaching Method	Number of Students	Percentage Passed
Elements of Programming (with Raptor)	A	Teaching by Task and Pair/Group Programming	109	100
Programming with C	A	Teaching by Task and Pair/Group Programming	109	100
Programming with C++	E	Teaching by Task and Pair/Group Programming	109	99.08
Programming with Java	C	Teaching by Task, Peer Tutoring, and Group Programming	107	100
Advanced Programming (C#)	D	Problem-Based Learning, Laboratory Practice, and Peer Tutoring	103	100



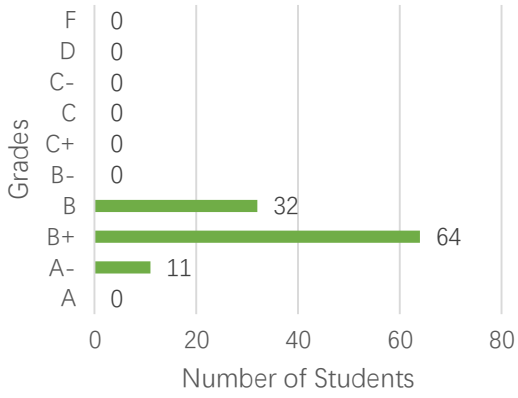
(a) Elements of Programming, Session 1 2014/2015



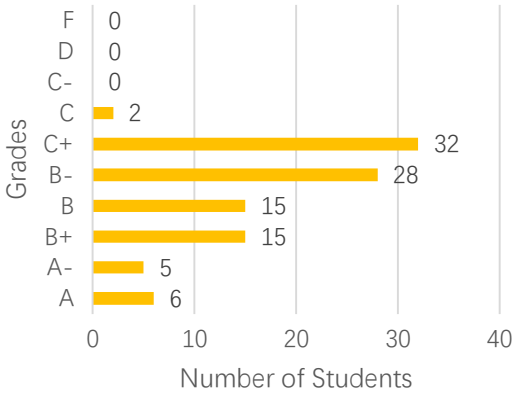
(b) Programming with C, Session 2 2014/2015



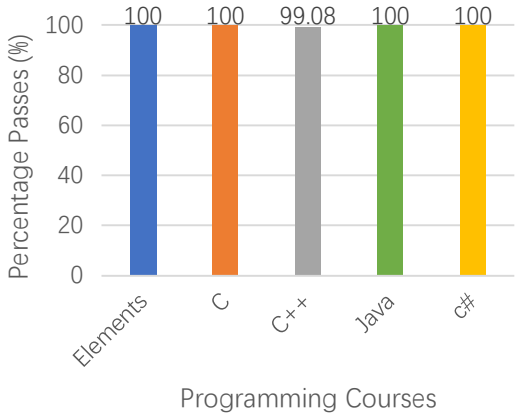
(c) Programming with C++, Session 3 2014/2015



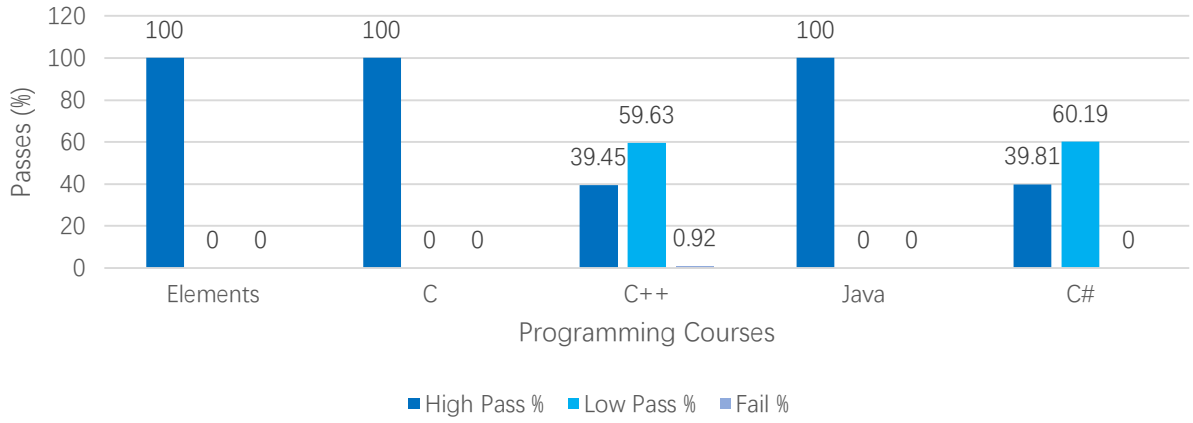
(d) Programming with Java, Session 1 2015/2016



(e) Programming with C#, Session 1 2016/2017



(f) Pass Rates in Programming Courses from Year-Group 2



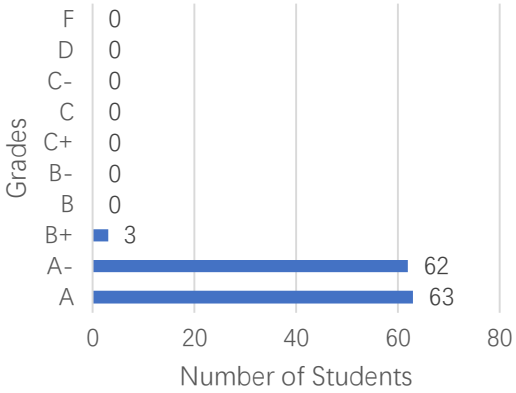
(g) Grade Distribution in Programming Courses from Year-Group 2

Fig. 2. Grade Trends of Students in Programming Courses for Year-Group 2

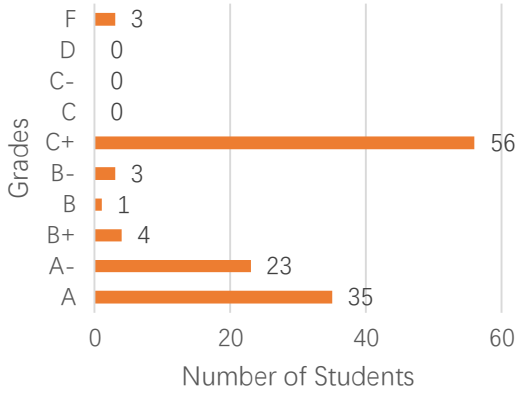
The second cohort of sandwich students was also instructed by four lecturers who employed various teaching methods. Three lecturers who taught the first cohort remained unchanged, while a new lecturer, Lecturer E, was introduced to teach Programming with C++. The average number of students in the second cohort was 107 students. The students' performance results are summarised in Fig. 2g. All students passed the courses Elements of Programming, Programming with C and Programming with Java. In the course Programming with C++, 39.45% of students achieved a high pass, 59.63% achieved a low pass, and 0.92% failed the course. In their final year, 103 students studied the C# programming language; 39.81% achieved a high pass, and 60.19% achieved low pass.

Table 10. Teaching Methodologies (Year-Group 3)

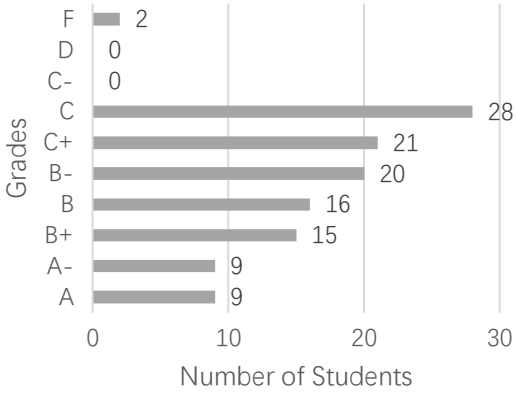
Programming Language	Instructor	Teaching Method	Number of Students	Percentage Passed
Elements of Programming (with Raptor)	A	Teaching by Task and Pair/Group Programming	128	100
Programming with C	A	Teaching by Task and Pair/Group Programming	125	97.6
Programming with C++	E	Teaching by Task, Peer Tutoring, and Group Programming	120	98.33
Programming with Java	C	Teaching by Task, Peer Tutoring, and Group Programming	128	94.53
Advanced Programming (C#)	D	Problem-Based Learning, Laboratory Practice, and Peer Tutoring	122	100



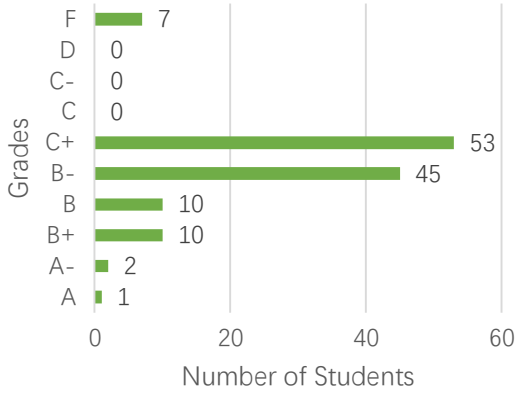
(a) Elements of Programming, Session 1 2015/2016



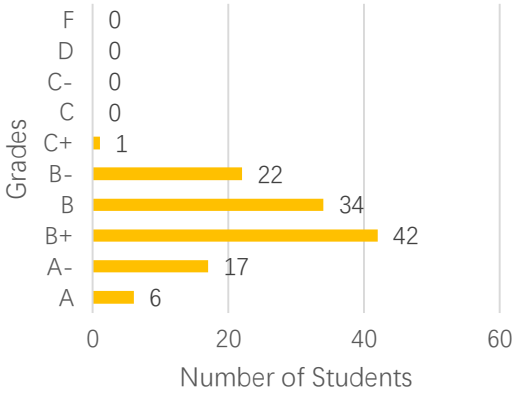
(b) Programming with C, Session 2 2015/2016



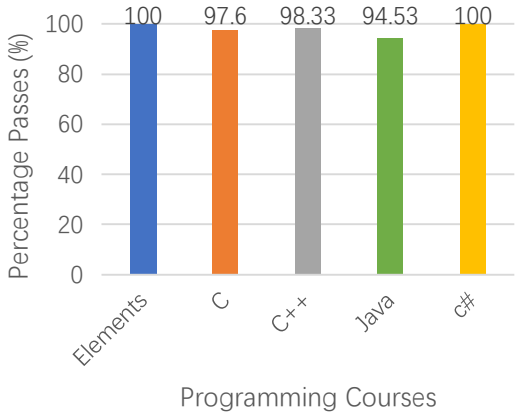
(c) Programming with C++, Session 3 2015/2016



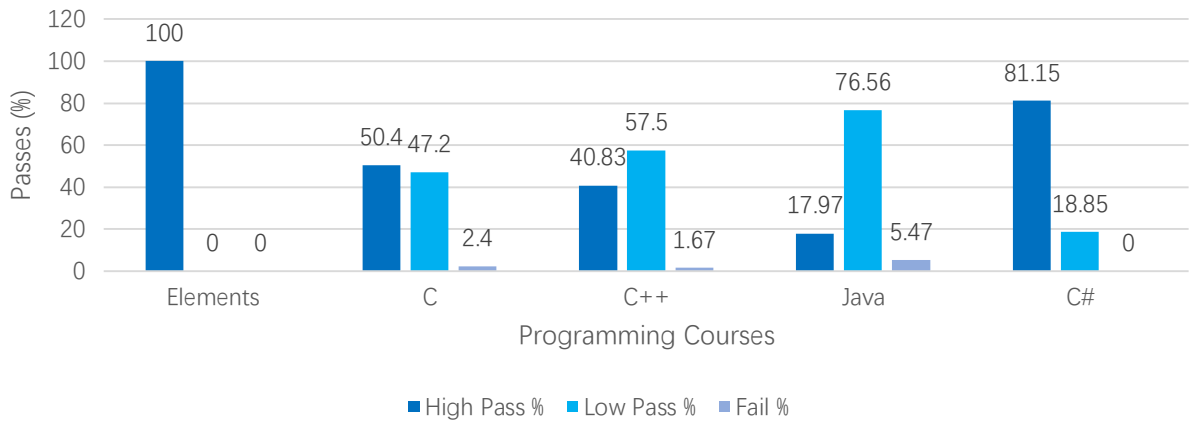
(d) Programming with Java, Session 1 2016/2017



(e) Programming with C#, Session 1 2017/2018



(f) Pass Rates in Programming Courses from Year-Group 3

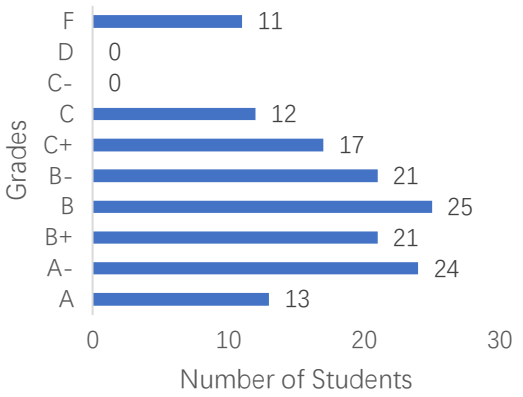


(g) Grade Distribution in Programming Courses from Year-Group 3

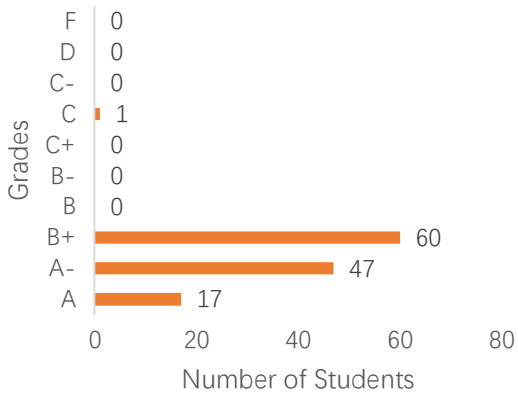
Fig. 3. Grade Trends of Students in Programming Courses for Year-Group 3

Table 11. Teaching Methodologies (Year-Group 4)

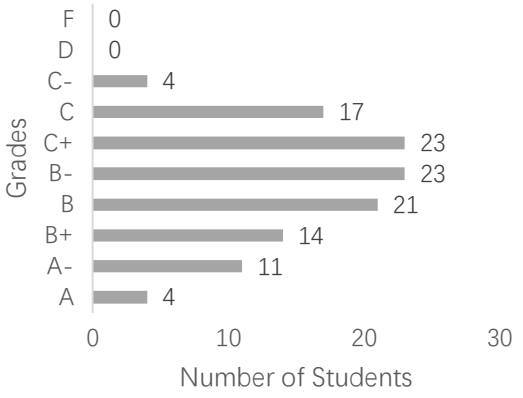
Programming Language	Instructor	Teaching Method	Number of Students	Percentage Passed
Elements of Programming (with Raptor)	A	Teaching by Task and Pair/Group Programming	144	92.36
Programming with C	A	Teaching by Task and Pair/Group Programming	125	100
Programming with C++	F	Puzzle-Based, Problem-Based Learning, Laboratory Practice, Teaching by Task	117	96.58
Programming with Java	D	Problem-Based Learning, Laboratory practice, Pair/Group Programming, and Peer Tutoring	113	100
Advanced Programming (C#)	G	Puzzled-Based Learning, Laboratory Practice, and Teaching by Task	125	81.6



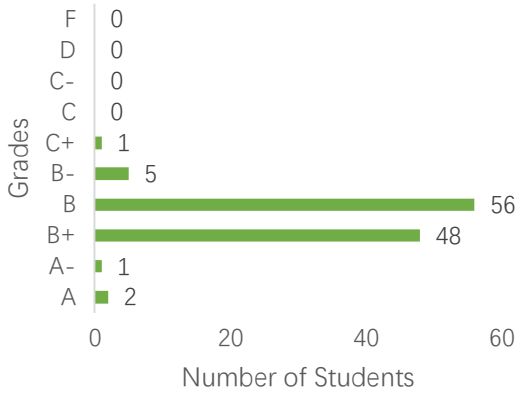
(a) Elements of Programming, Session 1 2016/2017



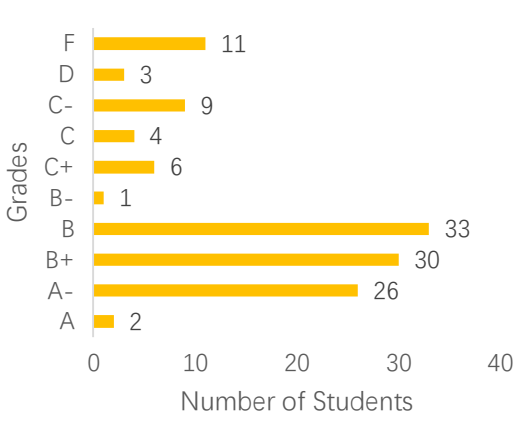
(b) Programming with C, Session 2 2016/2017



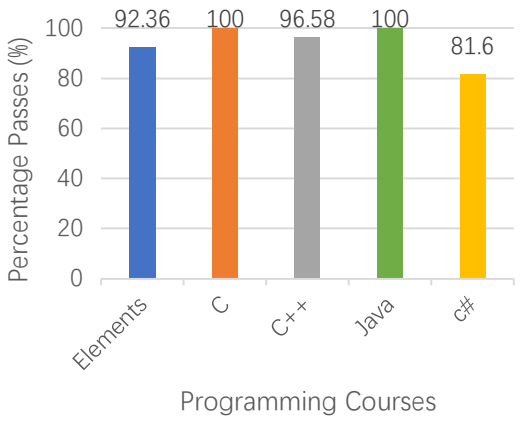
(c) Programming with C++, Session 3 2016/2017



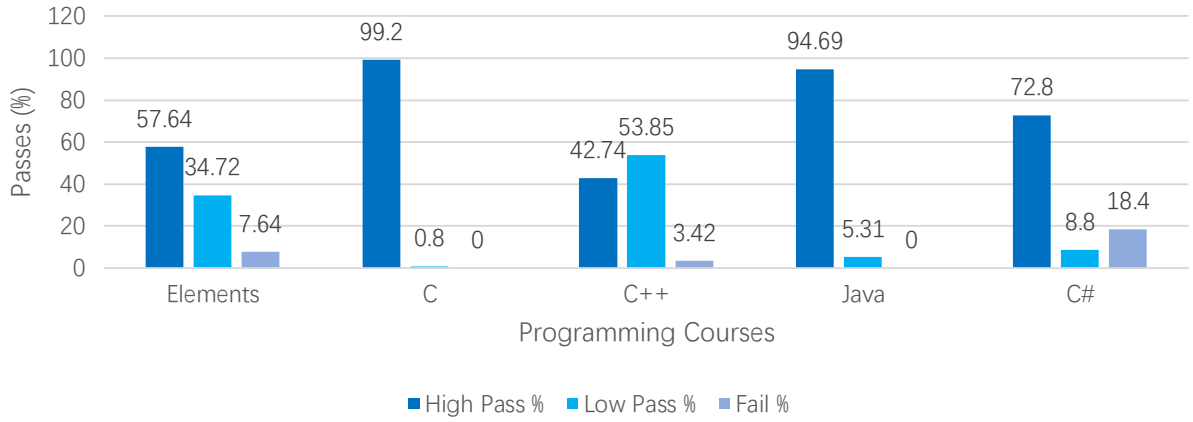
(d) Programming with Java, Session 1 2017/2018



(d) Programming with C-Sharp, Session 1 2018/2019



(f) Pass Rates in Programming Courses from Year-Group 4



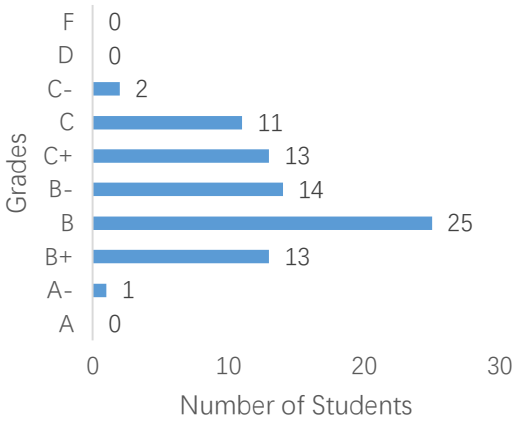
(g) Grade Distribution in Programming Courses from Year-Group 4

Fig. 4. Grade Trends of Students in Programming Courses for Year-Group 4

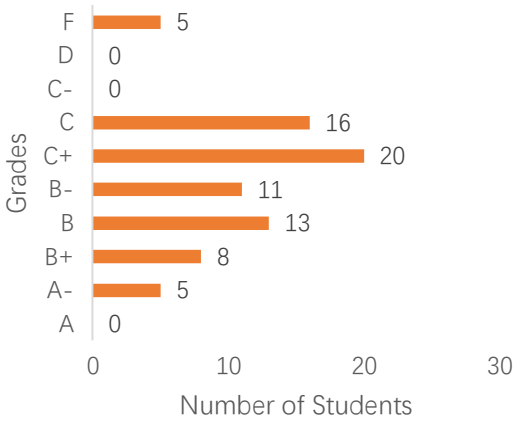
Four lecturers instructed the fourth cohort of the sandwich students. Lecturer A remained the instructor for the courses Elements of Programming and Programming with C. A new lecturer, Lecturer F, was introduced to teach C++, Lecturer D replaced Lecturer C in teaching Java, and Lecturer G, a new lecturer, taught C#. The average number of students in the fourth cohort was 125 students. The students' performance results are summarised in Fig. 4g. In the course Elements of Programming, 57.64% of students achieved a high pass, 34.72% achieved a low pass, and 7.64% failed the course. In the Programming with C course, 99.2% of students achieved a high pass, while 0.8% achieved a low pass. In the course Programming with C++, 42.74% of students achieved a high pass, 53.85% achieved a low pass, and 3.42% failed the course. In the Programming with Java course, 94.69% of students achieved a high pass, while 5.31% achieved a low pass. Finally, of the 125 students who studied the C# programming language, 72.8% achieved a high pass, 8.8% achieved a low pass, and 18.4% failed.

Table 12. Teaching Methodologies (Year-Group 5)

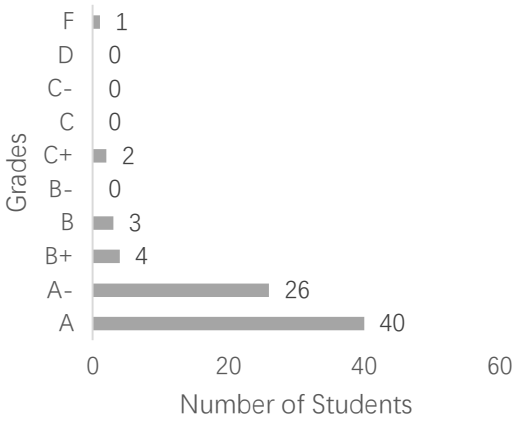
Programming Language	Instructor	Teaching Method	Number of Students	Percentage Passed
Elements of Programming (with Raptor)	F	Puzzle-Based Learning, Pair/Group Programming, and Laboratory Practice	79	97.47
Programming with C	F	Puzzle-Based Learning, Pair/Group Programming, and Laboratory Practice	78	93.59
Programming with C++	H	Pair/Group Programming, Teaching by a Task, and Laboratory Practice	76	98.68
Programming with Java	D	Problem-Based Learning, Laboratory Practice, Pair/Group Programming, and Peer Tutoring	78	98.72
Advanced Programming (C#)	G	Puzzled-Based Learning, Laboratory Practice, and Teaching by Task	75	97.33



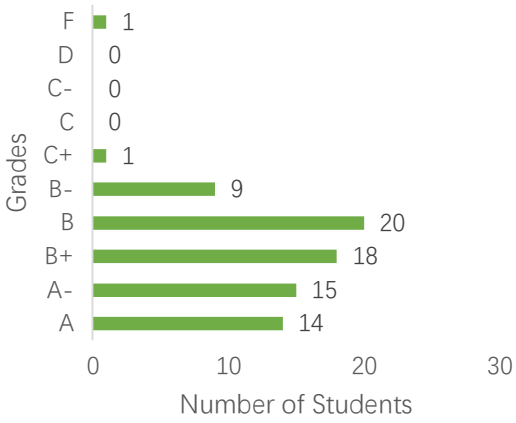
(a) Elements of Programming, Session 1 2017/2018



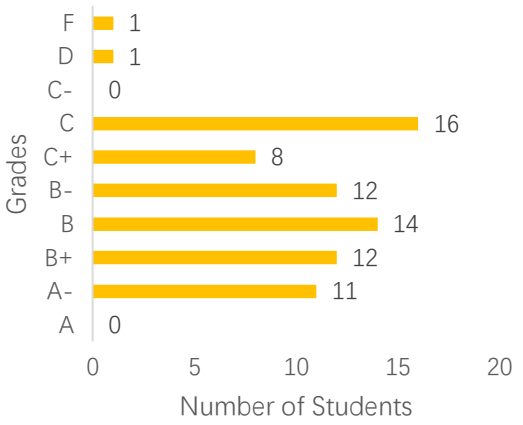
(b) Programming with C, Session 2 2017/2018



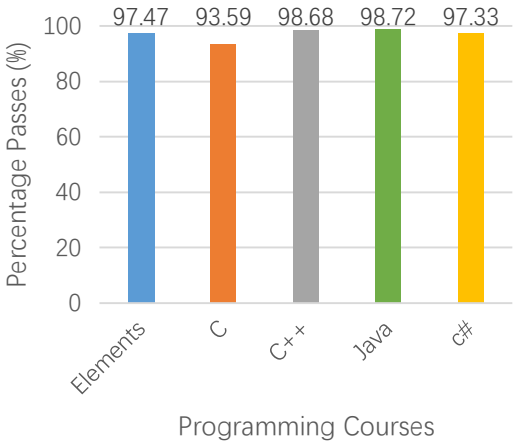
(c) Programming with C++, Session 3 2017/2018



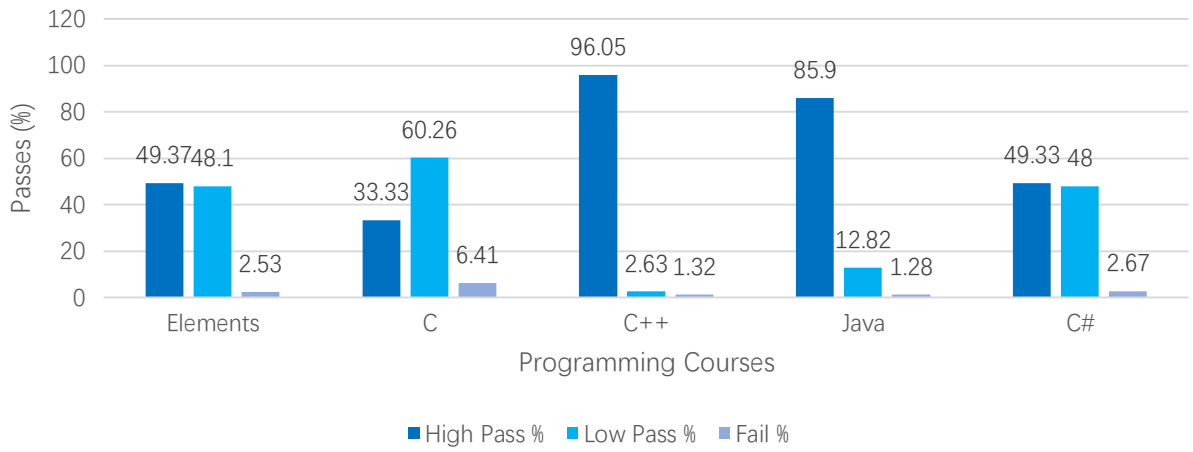
(d) Programming with Java, Session 1 2018/2019



(e) Programming with C-Sharp, Session 1 2019/2020



(f) Pass Rates in Programming Courses from Year-Group 5



(g) Grade Distribution in Programming Courses from Year-Group 5

Fig. 5. Grade Trends of Students in Programming Courses for Year-Group 5

Four lecturers instructed the fifth cohort of the sandwich students. Lecturer F took over from Lecturer A to teach the courses Elements of Programming and Programming with C. A new lecturer, Lecturer H, was introduced to teach C++, whereas Lecturers D and G maintained their courses.

Table 13. Grade Statistics for each Teaching Method across all Year Groups

Teaching Method	ID	Trials	Mean High Pass Rates (%)	Mean Low Pass Rates (%)	Mean Pass Rates (%)	Mean Fail Rates (%)
Teaching by Task and Pair/Group Programming	M1	9	77.071	20.856	97.928	2.072
Puzzle-Based, Problem-Based Learning, Laboratory Practice, Teaching by Task	M2	2	63.415	34.88	98.29	1.71
Teaching by Task, Peer Tutoring, and Group Programming	M3	4	53.2	45.015	98.215	1.785
Problem-Based Learning, Laboratory practice, and Peer Tutoring	M4	3	63.51	35.767	99.277	0.723
Puzzle-based Learning, Pair/Group Programming, and Laboratory Practice	M5	2	41.35	54.18	95.53	4.47
Puzzled-Based Learning, Laboratory Practice, and Teaching by Task	M6	2	61.065	28.4	89.465	10.535
Pair/Group Programming, Teaching by a Task, and Laboratory Practice	M7	1	96.05	2.63	98.68	1.32
Problem-Based Learning, Laboratory Practice, Pair/Group Programming, and Peer Tutoring	M8	2	90.29	9.065	99.36	0.64

The average number of students in the fifth cohort was 77 students. The students' performance results are summarised in Fig. 5g. In the course Elements of Programming, 49.37% of students achieved a high pass, 48.1% achieved a low pass, and 2.53% failed the course. In the Programming with C course, 33.33% achieved a high pass, 60.26% achieved a low pass, and 6.41% failed. In the course Programming with C++, 96.05% achieved a high pass, 2.63% achieved a low pass, and 1.32% failed the course. In Programming with Java, 85.9% achieved a high pass, 12.82% achieved a low pass, and 1.28% failed. Finally, of the 75 students who studied the C# programming language, 49.33% achieved a high pass, 48% achieved a low pass, and 2.67% failed.

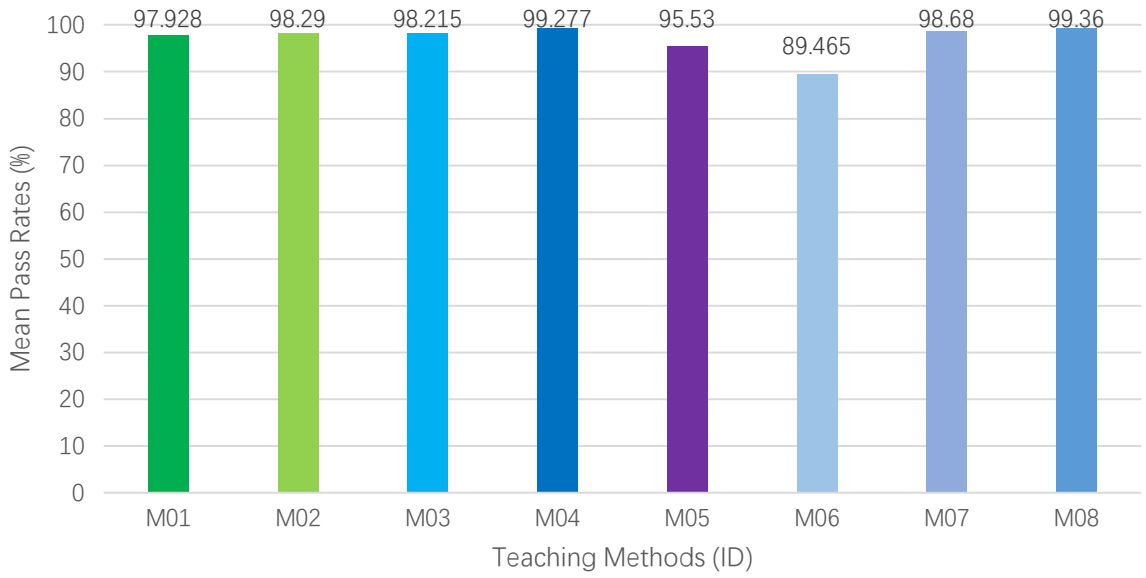


Fig. 6. Pass Rates for the Teaching Methods across all the Year Groups

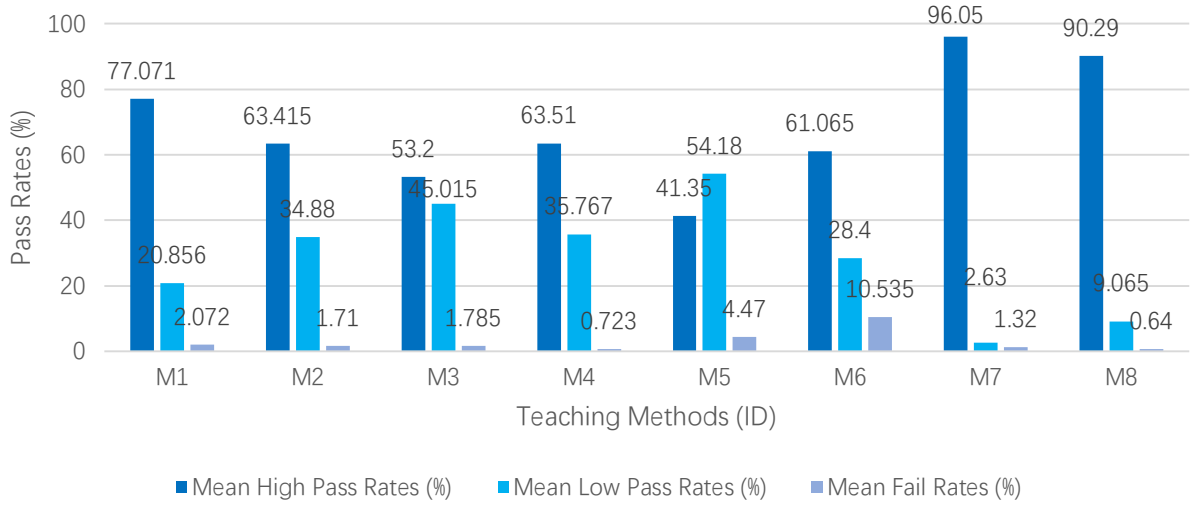


Fig. 7. Grade Distribution for the Teaching Methods

Table 14. Student Performance with Teaching Methods in Programming Courses Across Year Groups. TM is the IDs of the teaching methods used, as shown in Table 13, and HP and LP are the approximate high-pass and low-pass percentages, respectively.

Year Grp.	Programming Courses														
	Elements			C			C++			Java			C#		
	TM	HP	LP	TM	HP	LP	TM	HP	LP	TM	HP	LP	TM	HP	LP
1	M1	46.2	46.2	M1	100	0	M2	84.1	15.9	M3	54	46	M4	69.6	28.3
2	M1	100	0	M1	100	0	M1	39.5	59.6	M3	100	0	M4	39.8	60.2
3	M1	100	0	M1	50.4	47.2	M3	40.8	57.5	M3	18	76.6	M4	81.2	18.6
4	M1	57.6	34.7	M1	99.2	0.8	M2	42.7	53.9	M8	94.7	5.3	M6	72.8	8.8
5	M5	49.4	48.1	M5	33.3	60.3	M7	96.1	2.6	M8	85.9	12.8	M6	49.3	48

The data obtained from the experiment offer an intriguing perspective on the correlation between student performance in programming courses and the instructional methods employed. Although the selection of teaching methods was not intentionally randomised, the observed variations allowed for valuable conclusions to be drawn.

Table 14 shows the pass rates for each programming course and the corresponding teaching methods used for each year group. In the Elements of Programming (with Raptor) course, two instructors, Lecturer A and Lecturer F, were assigned to teach different year groups. Lecturer A taught year groups 1 to 4, utilising the teaching method, Teaching by Task and Pair/Group Programming (M1), which resulted in an average high pass rate of approximately 75.95% and an average low-pass rate of approximately 20.23%. Conversely, Lecturer F instructed year-group 5 in the same course

using the teaching methods of PBL, pair/group programming, and laboratory practice (M5), achieving an average high pass rate of approximately 49.4% and an average low pass rate of approximately 48.1%.

Similarly, in the Programming with C course, the same instructors as in the Elements of Programming (with Raptor) course, Lecturer A and Lecturer F, were responsible for teaching year-groups 1–4 and year-group 5, respectively. Both instructors continued to employ their previous methods of teaching. Lecturer A's utilisation of the Teaching by Task and Pair/Group Programming (M1) method resulted in an average high pass rate of approximately 87.4% and an average low pass rate of approximately 12%. In contrast, Lecturer F implemented the Puzzle-based Learning, Pair/Group Programming, and Laboratory Practice (M5) method, leading to an average high pass rate of approximately 33.3% and an average low pass rate of approximately 60.3%.

Four instructors taught all five-year groups for the Programming with C++ course: Lecturers B, E, F, and H. This course exhibited the most significant variation in the teaching methods employed. Lecturers B and F taught year groups 1 and 4, respectively, utilising the Puzzle-Based Problem-Based Learning Laboratory Practice Teaching by Task (M2) method, resulting in average high and low pass rates of approximately 63.4% and 34.9%, respectively. Lecturer E instructed year groups 2 and 3 using the Teaching by Task and Pair/Group Programming (M1) and Teaching by Task, Peer Tutoring, and Group Programming (M3) methods, respectively. For year group 2, the pass rates were approximately 39.5% (high) and 59.6% (low), whereas for year group 3, the pass rates were approximately 40.8% (high) and 57.5% (low). Lastly, Lecturer H, who taught year-group 5, employed the teaching method (M7), resulting in high and low pass rates of 96.1% and 2.6%, respectively.

Two different instructors were involved in the Programming with Java course. Lecturer C taught year-groups 1 to 3 in Java, favouring the Teaching by Task, Peer Tutoring, and Group Programming (M3) method. Lecturer D handled year groups 4 and 5 in Java, employing the Problem-Based Learning, Laboratory Practice, Pair/Group Programming, and Peer Tutoring (M8) method. Lecturer C achieved high and low pass rates of approximately 57.3% and 40.9%, respectively, while Lecturer D recorded high and low pass rates of approximately 90.3% and 9.1%, respectively.

Finally, in the Advanced Programming (C#) course, two different instructors, Lecturers D and G, taught the course. Lecturer D taught year groups 1 to 3, utilising the Problem-Based Learning, Laboratory Practice, and Peer Tutoring (M4) method, resulting in high and low pass rates of approximately 63.5% and 35.7%, respectively. Lecturer G handled year groups 4 and 5, favouring the Puzzle-Based Learning, Laboratory Practice, and Teaching by Task (M6) method, which yielded high and low pass rates of approximately 61.1% and 28.4%, respectively.

Each lecturer utilised a combination of teaching approaches to instruct their classes. All combinations contained at least two individual teaching approaches: Teaching by Task, Pair/Group Programming, Puzzle-Based Learning, Problem-Based Learning, Laboratory Practice, and Peer Tutoring. As shown in Fig. 6a, all the teaching method combinations performed relatively well in improving student performance.

Puzzle-Based Learning, Laboratory Practice, and Teaching by Task (M6) produced the lowest average student pass rate (89.465%). However, some teaching method combinations were more effective in producing high pass rates, with some instances of all students in a course receiving high passes, as shown in Table 15. The combinations included Teaching by Task and Pair/Group Programming (M1), Pair/Group Programming, Teaching by Task, and Laboratory Practice (M7), and Problem-Based Learning, Laboratory Practice, Pair/Group Programming, and Peer Tutoring (M8). These three combinations also exhibited the best performance (Fig. 6b).

Table 15. Percentage Pass Rates by Teaching Method Ranks

Teaching Methods	Percentage Passes	Ranks
M6	81.6	1
M1	92.31	2
M1	92.36	3
M5	93.59	4
M3	94.53	5
M2	96.58	6
M6	97.33	7
M5	97.47	8
M1	97.6	9
M4	97.83	10
M3	98.33	11
M7	98.68	12
M8	98.72	13
M1	99.08	14
M1	100	20
M1	100	20
M1	100	20
M1	100	20
M1	100	20
M2	100	20
M3	100	20
M3	100	20
M4	100	20
M4	100	20
M8	100	20

We employed the Kruskal-Wallis test to evaluate the significance of our experimental results. This non-parametric test is equivalent to a one-way analysis of variance (ANOVA) and compares the ranks of two or more groups. It is particularly useful when the sample size is small and the data do not meet the assumptions of a one-way ANOVA, such as equal probabilities of containing the highest value in each group. The test determines whether the differences between the rank sums of each group reflect a significant difference between the groups or are due to random noise using the chi-square statistic as an approximation. Table 15 shows the ranks of the pass rates for students for the different teaching methods.

We selected a significance level (α) of 0.05. The Kruskal-Wallis test results indicated no significant difference in academic performance among the eight teaching methods ($\chi 2(7) = 6.92$, $p = 0.437$). The mean rank scores for each method were M1 = 14.22, M2 = 13, M3 = 14, M4 = 16.67, M5 = 6, M6 = 4, M7 = 12, and M8 = 16.5.

The Kruskal-Wallis test did not reveal any significant intergroup differences. However, several factors may have influenced these results. Limitations in the experimental design may have contributed to the failure to detect significance in this study. Additionally, lecturer competency, student intelligence, prior programming experience, and student discipline culture may have affected students course performance. Therefore, the results of this study should be considered preliminary and inconclusive until further validation by a standardised randomised controlled trial.

6.3 Teaching Method Preferences

Although the grade trend analysis of the experimental data was preliminary and did not allow for causal inferences to be drawn in the trends of observed student performances, a thoughtful selection of a teaching approach or group of teaching approaches from the experiment can be made that is highly likely to foster high student performance in programming courses. From the grade trend analysis in the previous section, three combinations of teaching methods were found to accompany high percentages of students with high passes and very low failed student percentages, suggesting the potential for facilitating high comprehension of course content. These are Teaching by Task and Pair/Group Programming (M1) with mean high, low, and failed pass rates of approximately 77.1%, 20.9%, and 2.1%, respectively, from nine test cases; Pair/Group Programming, Teaching by Task, and Laboratory Practice (M7) with mean high, low, and failed pass rates of approximately 96.1%, 2.6%, and 1.3%, respectively, from a single test case; and Problem-Based Learning, Laboratory Practice, Pair/Group Programming, and Peer Tutoring (M8) with mean high, low, and failed pass rates of approximately 90.3%, 9.1%, and 0.6%, respectively, from two test cases.

The first two combinations shared teaching methods: Teaching by Task and Pair/Group Programming, while only differentiated by Laboratory Practice in the second combination. Pair/Group Programming was also part of the third combination, which shared Laboratory Practice with the second combination and included two unique methods: Peer Tutoring and Problem-Based Learning. The first combination (Teaching by Task and Pair/Group Programming (M1)) was nearly identical to the second combination (Pair/Group Programming, Teaching by Task, and Laboratory Practice (M7)). Considering that the former had been tested nine times compared to the single instance of the latter, it can be concluded that the former is objectively more effective than the latter, despite the latter's average passes being significantly more impressive than the former's. Similarly, the third combination lacked sufficient data to support any claim that it was more effective than its counterparts. Therefore, it can be concluded that the teaching method combination of Teaching by Task and Pair/Group Programming (M1) is the most effective method for instructing programming courses in the Sandwich Education System (SES), as indicated by the results of the experiments.

To further enhance the grade trend analysis, we conducted a cohort-based longitudinal evaluation. By analysing each year group's performance from their sophomore (entry-level) to senior-level programming courses, we identified and interpreted performance trajectories, highlighting patterns such as consistent improvement, fluctuations, or declines over the study period. This cohort-level analysis partially addresses the limitation of individual student tracking, providing more granular insights into the long-term impact of instructional methods on programming performance in the SES context.

7. Threats to Validity

This section presents and discusses the threats to the validity of the quantitative studies designed to ascertain students' viewpoints in the sandwich mode and evaluate experimental outcomes to determine potential correlations between students' performance in programming courses and the teaching methods employed. While the findings offer valuable insights, it is important to acknowledge potential threats to the research's validity, as they might have an undue influence or skew the collected data. By critically examining these threats, the reliability and generalisability of the results can be evaluated.

The questionnaires used in our survey were designed to be unambiguous. However, their reliability is related to how the data is collected and how other researchers could replicate the analysis performed. This work can be replicated quickly because the report explicitly states how the results were obtained. However, potential threats to validity could have affected the survey results. One such threat is the sampling bias. The study's participants were exclusively drawn from the Sandwich Education System (SES) Bachelor of Education with Information Technology (B.Ed. IT) program at University X. Consequently, the findings may not be generalisable to a broader population of students or educational

institutions. The unique characteristics of the SES program and its specific student body limit the external validity of this study.

Another potential threat to validity arises from the self-selection bias. The relatively low response rate, with only 218 of the 357 students participating in the survey, introduces the possibility that the subset of students who chose to respond may differ significantly from those who did not participate. These differences in characteristics, attitudes, or motivations could lead to biased results and limit the generalisability of the findings beyond the participating students. Social desirability bias is also a potential concern in self-report surveys such as this one. Participants may have been motivated to provide responses that were perceived as socially desirable or aligned with the researchers' expectations. This bias can influence the accuracy and honesty of participants' responses, potentially leading to an overrepresentation of socially desirable views and understating the prevalence of less socially acceptable perspectives.

Moreover, the survey relied on participants' recollection of past experiences, preferences, and attitudes. Recall bias is a potential threat to validity, as participants may not accurately remember or report their experiences, potentially leading to inaccuracies in the data. Distortions in participants' responses may be introduced by memory biases, such as selective recall or forgetting specific details, thereby limiting the reliability and validity of the findings. Similarly, surveying at a single point threatens the validity of the results. Participants' views and experiences may vary, and a single data collection point may not capture this variability. Longitudinal or repeated measures could provide a more comprehensive understanding of participants' perspectives and allow for the identification of temporal changes in attitudes and experiences.

The survey focused on specific aspects such as demographics, motivations, teaching approaches, learning styles and satisfaction. While these aspects provide valuable insights, other relevant factors that could influence students' experiences and perceptions may not have been included in this survey. The limited scope of the questions may restrict the understanding of the depth and breadth of the research topic.

In future research, we hope to mitigate these threats and improve the validity of research results by extending survey coverage to multiple SES institutions to ensure a representative sample; using validated measurement instruments to reduce biases; conducting longitudinal studies to capture changes over time; and employing diverse data collection methods (e.g. interviews, observations) to gain a comprehensive understanding of students' experiences.

Selection bias potentially threatens the validity of research experiments. Our experiment had to be a quasi-experiment to eliminate any potential disruption to students' academic workflow. We had very little control over the independent variables and could only observe the naturally occurring differences in the teaching methodologies employed by different instructors. As such, the assignment of teaching methods to instructors could not be randomised, leading to biases in the observed trends. Variations in instructors' experience, skill, or teaching style could have influenced student performance, confounding the observed results. Future studies should consider randomising teaching method allocation to mitigate this threat.

Another potential threat is the maturation effect. Over time, students may naturally improve their programming skills, irrespective of the teaching methods employed. This inherent maturation could contribute to changes in student performance, making it challenging to attribute outcomes solely to the instructional approaches. Longer study durations or longitudinal designs would allow for a better understanding and control of these effects.

The testing effect poses a threat to validity. With repeated exposure to teaching methods and assessments, students may become more familiar with the format and content, potentially improving their performance in subsequent trials. The Hawthorne effect, whereby participants alter their behaviour because they are part of an experiment, could further influence student performance. Measures to mitigate these effects include counterbalancing teaching method orders and implementing control groups. The history effect is another potential threat to the validity of the study. External events, changes in curriculum, or access to additional resources may have influenced student performance independently of the teaching methods being investigated. Controlling for these external factors or conducting the study across multiple institutions would strengthen the validity of the findings.

Moreover, the limited sample size employed in this study could compromise the generalisability and statistical power of the results. With a small sample size, the findings may not represent the broader student population or achieve sufficient statistical significance. Replication studies with larger sample sizes would enhance the reliability of the results.

Finally, the lack of control over certain variables introduces another potential threat. Factors such as instructor competence, students' prior programming experience, and students' discipline culture were not explicitly addressed in this study. These variables could have influenced student performance and should be considered in future research using controlled experimental designs or statistical control techniques.

Employing randomised controlled trials with more extensive and diverse samples is recommended to strengthen the validity of future research in this area. Additionally, utilising rigorous experimental designs, controlling for relevant variables, and addressing potential threats to validity will enhance the scientific rigour and objectivity of the research. By addressing these threats, researchers can refine their methodologies and contribute to a more robust understanding of the effectiveness of instructional approaches in programming education.

8. Conclusion, Future Research Directions and Recommendations for SES Educators

8.1 Conclusion

This study presents a comprehensive investigation of students' performance in programming courses within Ghana's Sandwich Education System (SES). The study employed a mixed-methods approach, using surveys, observations, and quasi-experiments to collect and analyse data from students enrolled in the Bachelor of Education programme in Information Technology (B.Ed. IT) at University X. This study aimed to determine whether a significant relationship existed between students' pass and failure rates in programming courses and the teaching methodologies employed by their instructors within the SES. The study also explored students' views, attitudes, perceptions, and beliefs regarding various aspects of programming education within SES.

The findings of this study did not demonstrate a significant difference in the academic performance of students taught using different teaching methods in SES. This qualitative exploration strengthened our mixed-methods approach by highlighting critical insights into students' experiences and perceptions. However, it is essential to acknowledge the limitations of this study, which necessitate considering the findings as insightful observations rather than conclusive results. The study faced several challenges, such as focusing on a single institution, relying on self-reported data, and using a quasi-experimental design that limited the control over independent variables. These challenges may have introduced biases or confounding factors into the data, affecting the validity and reliability of our results.

The study also revealed that students faced various difficulties in learning programming concepts, such as conceptual complexity, a lack of feedback, and insufficient practice time. Moreover, the study identified that students had diverse learning styles, preferences, and motivations for choosing the sandwich mode of study. These factors may have influenced students' engagement and performance in programming courses. The study also found that students utilised online resources, such as video tutorials, to supplement their learning and overcome some challenges posed by the sandwich mode. This study has significant implications for educators and policymakers in Ghana and other countries with similar educational systems. The study provides valuable insights into effective pedagogical approaches that can be used to enhance student outcomes in programming courses. Furthermore, this study emphasises the importance of providing students with diverse learning opportunities and support, such as access to online resources and collaborative learning. The implementation of cohort-based longitudinal tracking allowed us to partially overcome the limitations associated with the lack of individual-level longitudinal data. This methodological enhancement provides insights into the stability and evolution of student performance, thereby strengthening the reliability of our findings.

8.2 Future Research Directions

The study also suggests several directions for future research to address its limitations and explore additional factors that may influence student performance within the SES. Future research should:

i. focus on a more rigorous experimental design, such as a randomised controlled trial, should be used to examine the causal effects of different teaching methods on student performance. Given the inherent limitations of the quasi-experimental approach used in this study, future research should employ a randomised controlled trial (RCT) or other rigorous experimental designs to effectively isolate variables and establish causal relationships between instructional methods and student performance outcomes in programming courses. Rigorous methodologies would provide substantial evidence regarding the effectiveness of teaching interventions.
ii. explicitly incorporated a clearly defined control group, such as students enrolled in traditional or distance learning programs, to facilitate a rigorous comparative analysis. Such an approach would significantly enhance the ability to attribute observed differences in student performance directly to the SES model's specific characteristics. Indeed, we recognise that inconsistent application of pedagogical approaches can threaten reliability and internal validity.
iii. include a larger and more representative sample of students from various institutions and programs within SES to increase the generalisability of the findings.
iv. investigate other aspects of programming education within SES, such as student satisfaction, retention rates, and career outcomes. Such endeavours would enable more robust causal inferences to be drawn and provide more comprehensive evidence for improving programming education within the SES. Future research should replicate this study across multiple institutions that employ the Sandwich Education System (SES) to establish more generalisable insights and enhance the external validity of the findings. Conducting comparative analyses involving various universities and colleges could provide robust evidence of the influence of teaching methodologies on students' programming performance in diverse educational contexts.
v. complement self-reported survey data with objective performance measures or qualitative methods, such as structured interviews or direct observations, to validate the findings and reduce potential response bias related to sensitive self-assessments of programming difficulties.

vi. provide a comprehensive and standardised protocol detailing how each instructional method should be applied across all student groups. Systematic monitoring and documentation of teaching method fidelity would significantly enhance the reliability and reproducibility of the findings.
vii. examine how different teaching methodologies interact with students' diverse learning styles and varying levels of prior programming knowledge. Conducting such analyses could enhance our understanding of personalised educational strategies, thereby improving the effectiveness of programming instruction within the Sandwich Education System and similar educational contexts
viii. integrate qualitative data collection and thematic analysis into similar studies will ensure a balanced understanding of quantitative results through the depth and context provided by qualitative analysis.

8.3 Recommendations for Educators in SES Programming Courses

Based on the findings of this study, we propose the following strategies to enhance programming education within the SES context.

i. *Modular Instructional Design:* Decompose complex programming concepts into manageable modules, each accompanied by real-world examples and practical tasks to reinforce learning.
ii. *Blended Learning Approaches:* Offer video tutorials, simulations, and interactive coding platforms for pre-session preparation and post-class reinforcement, which are particularly beneficial given the compressed schedule of the SES. Peer
iii. *Mentoring and Group Projects:* Implement structured peer-tutoring systems where advanced students or alumni mentor current learners, and encourage small group-based coding projects to enhance collaboration and mitigate isolation.
iv. *Formative Assessments and Immediate Feedback:* Integrate low-stakes, in-class coding quizzes with real-time feedback to enable students to monitor their progress and rectify misunderstandings promptly.
v. *Personalised Remediation Sessions:* Identify struggling students early through diagnostic assessments and provide tailored support, either through tutorial hours or targeted workshops.
vi. *Gamified Learning Elements:* Introduce gamification elements (e.g. leaderboards, badges, coding challenges) to foster engagement, particularly among students with limited prior exposure to programming. Use of Visual and
vii. *Block-Based Programming Tools:* Initiate instruction with visual languages such as Scratch or Raptor for students with no prior exposure to help them grasp fundamental logic before transitioning to text-based languages.

References

[1] Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. ACM SIGCSE Bulletin, 39 (2), 32–36.
[2] Yunlong Lu, Na Meng, Wenxin Li. 2021. FAPR: Fast and Accurate Program Repair for Introductory Programming Courses. arXiv preprint arXiv:2107.06550 [cs.SE].
[3] Hawi, N. (2010). Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course. Computers & Education, 54 (4), 1127–1136.
[4] José A. Q. Figueiredo and Francisco José García-Peñalvo. 2021. Teaching and Learning Strategies for Introductory Programming in University Courses. In Ninth International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM'21) (TEEM'21), October 26–29, 2021, Barcelona, Spain. DOI: https://doi.org/10.1145/3486011.3486540.
[5] Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. In Acm sigcse bulletin (Vol. 37, pp. 14–18).
[6] Horton, D., Craig, M., Campbell, J., Gries, P., & Zingaro, D. (2014). Comparing outcomes in inverted and traditional cs1. In Proceedings of the 2014 conference on innovation & technology in computer science education (pp. 261–266).
[7] Bennedsen, J. (2008). Teaching and learning introductory programming: a model-based approach.
[8] Bosch, N., & D'Mello, S. (2013). Sequential patterns of affective states of novice programmers. In The first workshop on ai-supported education for computer science (aiedcs 2013) (pp. 1–10).
[9] Esteves, M., Fonseca, B., Morgado, L., & Martins, P. (2011). Improving teaching and learning of computer programming through the use of the second life virtual world. British Journal of Educational Technology, 42 (4), 624–637.
[10] Sarpong, K. A.-M., Arthur, J. K., & Amoako, P. Y. O. (2013). Causes of failure of students in computer programming courses: The teacher-learner perspective. International Journal of Computer Applications, 77 (12).
[11] Chibaya, C. (2019). A metaphor-based approach for introducing programming concepts. In 2019 international multidisciplinary information technology and engineering conference (imitec) (pp. 1–8).
[12] Dasuki, S., & Quaye, A. (2016). Undergraduate students? failure in programming courses in institutions of higher education in developing countries: A nigerian perspective. The Electronic Journal of Information Systems in Developing Countries, 76 (1), 1–18.
[13] Lishinski, A., Yadav, A., Enbody, R., & Good, J. (2016). The influence of problem solving abilities on students' performance on different assessment tasks in cs1. In Proceedings of the 47th acm technical symposium on computing science education (pp. 329–334).

[14] Lishinski, A., Yadav, A., Good, J., & Enbody, R. (2016). Learning to program: Gender differences and interactive effects of students' motivation, goals, and self-efficacy on performance. In Proceedings of the 2016 acm conference on international computing education research (pp. 211–220).
[15] Rafique, W., Dou, W., Hussain, K., & Ahmed, K. (2020). Factors influencing programming expertise in a web-based e-learning paradigm. Online Learning, 24 (1).
[16] Kori, K., Pedaste, M., Leijen, A., & T˜onisson, E. (2016). The role of programming experience ¨ in ict students' learning motivation and academic achievement. International Journal of Information and Education Technology, 6 (5), 331.
[17] Bennedsen, J., & Caspersen, M. E. (2019). Failure rates in introductory programming: 12 years later. ACM Inroads, 10 (2), 30–36.
[18] Watson, C., & Li, F. W. (2014). Failure rates in introductory programming revisited. In Proceedings of the 2014 conference on innovation & technology in computer science education (pp. 39–44).
[19] Mustapha, A., Samsudin, N. A., Arbaiy, N., Mohammed, R., & Hamid, I. R. (2016). Generic assessment rubrics for computer programming courses. Turkish Online Journal of Educational Technology-TOJET, 15 (1), 53–68.
[20] Jakchaikul, V., & Kansrirat, T. (2015). Teaching management to resolve student problems in computer programming.
[21] Watson, C., Li, F. W., & Godwin, J. L. (2013). Predicting performance in an introductory programming course by logging and analyzing student programming behavior. In Advanced learning technologies (icalt), 2013 ieee 13th international conference on (pp. 319–323).
[22] Razakhovich, S. S., & Razakhovna, S. B. (2020). Development of students' programming abilities with the means of non-programming disciplines and activities. In Handbook of research on diverse teaching strategies for the technology-rich classroom (pp. 89–97). IGI Global.
[23] Martins, V. F., de Almeida Souza Concilio, I., & de Paiva Guimar˜aes, M. (2018). Problem based learning associated to the development of games for programming teaching. Computer Applications in Engineering Education, 26 (5), 1577–1589.
[24] Matthew, F. T., Adepoju, A. I., Ayodele, O., Olumide, O., Olatayo, O., Adebimpe, E., . . . Funmilola, E. (2018). Development of mobile-interfaced machine learning-based predictive models for improving students? performance in programming courses. Development, 9 (5).
[25] Yildiz Durak, H. (2018). Digital story design activities used for teaching programming effect on learning of programming concepts, programming self-efficacy, and participation and analysis of student experiences. Journal of Computer Assisted Learning, 34 (6), 740– 752.
[26] Ismail, M. N., Ngah, N. A., & Umar, I. N. (2010). Instructional strategy in the teaching of computer programming: a need assessment analyses. TOJET: The Turkish Online Journal of Educational Technology, 9 (2).
[27] Kaplan, R. (2015). Using problem-based learning in a cs1 course-tales from the trenches. In Proceedings of the international conference on frontiers in education: Computer science and computer engineering (fecs) (p. 86).
[28] Baloyi, L. L., Ojo, S. O., & Van Wyk, E. A. (2017). Design and development of an interactive multimedia simulation for augmenting the teaching and learning of programming concepts. International Association for Development of the Information Society.
[29] Quaye, A. M., & Dasuki, S. I. (2017). A computational approach to learning programming using visual programming in a developing country university. In Emerging research, practice, and policy on computational thinking (pp. 121–134). Springer.
[30] Tritrakan, K., Kidrakarn, P., & Asanok, M. (2016). The use of engineering design concept for computer programming course: A model of blended learning environment. Educational Research and Reviews, 11 (18), 1757–1765.
[31] Chua, F. F., & Liew, K. L. (2016). A sns-integrated collaborative learning system to support programming language learning. Journal of Theoretical and Applied Information Technology, 88 (3), 456–463.
[32] Bati, T., Gelderblom, H., & van Biljon, J. (2015). Blended learning of programming in large classes: A reflection of student? experience from an ethiopian university. Transform2015. net.
[33] Alajmi, F., & Alkhatib, A. A. (2015). Enhanced teaching model (etm) for teaching programming languages. International Journal of Computer Applications, 121 (20).
[34] Vihavainen, A., Airaksinen, J., & Watson, C. (2014). A systematic review of approaches for teaching introductory programming and their influence on success. In Proceedings of the tenth annual conference on international computing education research (pp. 19–26).
[35] Butler, M., Morgan, M., et al. (2007). Learning challenges faced by novice programming students studying high level and low feedback concepts. Proceedings ascilite Singapore (99-107).
[36] Kasto, N. (2016). Learning to program: The development of knowledge in novice programmers (Unpublished doctoral dissertation). Auckland University of Technology.
[37] Mccall, D. (2016). Novice programmer errors-analysis and diagnostics (Unpublished doctoral dissertation). University of Kent.
[38] Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. Computer science education, 13 (2), 137–172.
[39] Hoda, R., & Andreae, P. (2014). It's not them, it's us! why computer science fails to impress many first years. In Proceedings of the sixteenth australasian computing education conference-volume 148 (pp. 159–162).
[40] Jayaprakash, S., Balamurugan, E., & Chandar, V. (2015). Predicting students? academic performance using na ̈ıve bayes algorithm. In 8th annual international applied research conference.
[41] Shuhidan, S., Hamilton, M., & D'Souza, D. (2009). A taxonomic study of novice programming summative assessment. In Proceedings of the eleventh australasian conference on computing education-volume 95 (pp. 147–156).
[42] Cˆandida, L., Silva Carmo, M. J. M., & Mendes, A. J. (2007). The impact of learning styles in introductory programming learning.
[43] Seyal, A. H., Mey, Y. S., Matusin, M. H., Siau, N. H., & Rahman, A. A. (2015). Understanding students learning style and their performance in computer programming course: Evidence from bruneian technical institution of higher learning. International Journal of Computer Theory and Engineering, 7 (3), 241.
[44] Felder, R. M. (1996). Matters of style. ASEE prism, 6 (4), 18–23.

[45] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., . . . Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. ACM SIGCSE Bulletin, 33 (4), 125–18.
[46] Norwawi, N. M., Abdusalam, S. F., Hibadullah, C. F., & Shuaibu, B. M. (2009). Classification of students' performance in computer programming course according to learning style. In Data mining and optimization, 2009. dmo'09. 2nd conference on (pp. 37–41).
[47] Çakıroğlu, Û. (2014). Analyzing the effect of learning styles and study habits of distance ¨learners on learning performances: A case of an introductory programming course. The International Review of Research in Open and Distributed Learning, 15 (4).
[48] Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming: What works? Communications of the ACM, 56 (8), 34–36.
[49] Sax, L. J., Lehman, K. J., Jacobs, J. A., Kanny, M. A., Lim, G., Monje-Paulson, L., & Zimmerman, H. B. (2017). Anatomy of an enduring gender gap: The evolution of womens participation in computer science. The Journal of Higher Education, 88 (2), 258-293. Retrieved from https://doi.org/10.1080/00221546.2016.1257306.
[50] Schindler, C., & Müller, M. (2019). Gender gap? a snapshot of a bachelor computer science course at graz university of technology. In Proceedings of the 13th european conference on software architecture-volume 2 (pp. 100–104).
[51] Mhashi, M. M., & Alakeel, A. (2013). Difficulties facing students in learning computer programming skills at tabuk university. In Proceedings of the 12th international conference on education and educational technology (edu?13), iwate, japan (pp. 15–24).
[52] Amoako, P. Y. O., Sarpong, K. A.-m., Arthur, J. K., & Adjetey, C. (2013). Performance of students in computer programming: Background, field of study and learning approach paradigm. International Journal of Computer Applications, 77 (12).

Author's Profiles

Kofi Sarpong Adu-Manu is an Associate Professor of Wireless Communication Networks at the Department of Computer Science, University of Ghana. He has over 50 publications in reputable journals, with 1500+ citations. His research interests include Wireless Sensor Networks (WSNs), Artificial Intelligence (AI), Cyber Security (CS), Internet of Things (IoTs), Emerging Technologies (ETs), Computer Science Education (CSE), and Educational Technologies (EdTech). He has consulted for UNICEF, UNESCO, the Commonwealth of Learning, and the World Bank. He serves as the lead Judge of the National Stemnnovation Contest and supports CENDLOS under the Ministry of Education. He serves on University of Ghana committees, is a Programme Assessor for the Ghana Tertiary Education Commission (GTEC), and founded Ladies in Tech Ghana to empower females in computer science.

Charles Adjetey is an adept problem-solver with strong communication and leadership skills. He excels in understanding students and achieving optimal learning outcomes. He has taught at Ashesi University, Lancaster University (Ghana Campus), and Valley View University, progressing from a teaching assistant to a lecturer. Charles has taught various undergraduate courses in Computer Science, Information Technology, and Education programs, specialising in Programming, System Analysis, and Operating Systems. He considers teaching a privilege and responsibility that shapes the future generations. His teaching approach focuses on engaging students, developing critical thinking skills, and respecting diverse perspectives. Charles holds an MPhil in Computer Science from Kwame Nkrumah University, a BSc from Valley View University, and a PGCAP from Lancaster University, UK.

John Kingsley Arthur is a Senior Lecturer of Computer Science at Valley View University, Ghana, and currently pursuing a PhD in Computer Applied Technology at Jiangsu University, China. With over a decade of experience in teaching, research, and academic leadership, his scholarly contributions span artificial intelligence, recommender systems, blockchain technology, and cybersecurity. He has published extensively in high-impact journals such as IEEE Access, Applied Sciences, and Engineering Applications of Artificial Intelligence. He serves as a reviewer for top-tier journals, including ACM Computing Surveys and Computers & Security.

How to cite this paper: Kofi Sarpong Adu-Manu, Charles Adjetey, John Kingsley Arthur, "Exploring the Performance of Students in Programming Courses: A Study of Ghana's Sandwich Computing Education", International Journal of Education and Management Engineering (IJEME), Vol.16, No.1, pp. 34-62, 2026. DOI:10.5815/ijeme.2026.01.03