# Exploring pedagogies, opportunities, and challenges of teaching and learning programming in business school

Abstract

*Purpose*

This study develops an interdisciplinary business and computer science pedagogy for teaching and learning computer programming in business schools at higher education institutions and explores its associated benefits, challenges, and improvement.

*Design/methodology/approach*

Based on a body of theories, an interdisciplinary pedagogy is developed and tested for programming education in a business context. Meanwhile, based on the unified theory of acceptance and use of technology (UTAUT), we used observation study and thematic analysis to explore opportunities, challenges, and future improvements associated with this interdisciplinary pedagogy.

*Findings*

The developed pedagogy includes integrating humanism and construction theory, problem-based learning, cognitive development, active instructional strategies, synergy of individual and group programming tasks, and creating an encouraging and inclusive learning environment. This study shows that business students perceive this novel pedagogy as highly valuable because it enhances their logical thinking and problem-solving abilities while giving them a sense of accomplishment. Although students face challenges in data preprocessing, error handling, and translating theoretical knowledge, they find it useful to review teaching materials, seek peer support, and learn independently through online resources. Further improvements in pedagogy include incorporating collaborative code reviews, using shared documents for troubleshooting, and grouping students based on their prior programming experience.

*Originality*

This is the first interdisciplinary study investigating teaching programming in a business context.

*Practical implications*

This interdisciplinary pedagogy can guide business schools to improve the quality of programming-related modules, enhance students' performance, and prepare them for future careers.

**Keywords:** Programming; Teaching and Learning; Interdisciplinary Pedagogy; Computer Science; Business School; UTAUT.

## 1. Introduction

Computer programming can help business students automate routine tasks to increase productivity and operational efficiency (Nambisan, 2017). Understanding programming helps business students better collaborate with technical teams to ensure business models can be achieved by technical solutions (Alvai et al., 1995). Meanwhile, business students can use computer programming to conduct big-data analytics and inform strategic business decisions (Marr, 2021). Therefore, equipping business students with programming skills is essential and critical for enhancing students' employability and market value.

However, learning programming is always a challenging task. Previous studies found that factors leading to poor learning of programming include inappropriate pedagogy, insufficient instructional materials, cognitive problems, and a lack of critical and computational thinking skills (Chukwunweike et al., 2015; Medeiros et al., 2018; Elcicek and Karal, 2020). Several studies have been devoted to proposing pedagogies, suggesting that interactive lectures, computer-based activities, team collaboration, game-based learning, and problem-based learning can help computer science students perform better in programming-related courses (Buitrago Florez et al., 2017; Arcos et al., 2017). However, there exist distinctive differences in learning objectives and curriculum content between computer science and business students. For business students, the emphasis is on how to apply programming skills to conduct data manipulation and enhance business processes, rather than understanding complex theories of algorithms (Sollosy and Mclnerney, 2022). Therefore, there is a lack of appropriate pedagogy for teaching programming in business schools. This study aims to answer the following 3 research questions:

1. How to design an appropriate pedagogy for teaching programming to business students? Is it possible to integrate both computer science and business pedagogy?
2. How do students rate this pedagogy? What are the associated opportunities and challenges?
3. What should we do to further improve this novel pedagogy?

This study is built around the development and evaluation of an interdisciplinary pedagogy, which integrates both business and computer science pedagogy and is based on a series of theories. This study further employs an empirical approach to test this pedagogy through delivering a programming course to final-year business graduates. This study observes students' behavior during the course to gain insights into how they engage with and respond to this interdisciplinary approach. This study also conducts a thematic analysis of students' reflective diaries to identify recurring themes and patterns that reflect the opportunities, challenges, and areas for future improvements of the proposed pedagogy.

This study makes three distinct contributions. Firstly, previous studies have mainly focused on computer science students, who have different technical backgrounds and curriculum contents (Xu and Babaian, 2021). Technology and business are interrelated and employers are increasing demanding that fresh business graduates possess multiple skills beyond the core business awareness. Computing and technology skills are increasingly on employers' top requirements for new recruits. This study proposes an interdisciplinary business and computer science pedagogy to help students better acquire the emerging technical skills increasingly required in business processes and bridge the gap between business knowledge and technological competence, especially programming given that it can enable them to manipulate, analyse, and visualise data efficiently. Secondly, existing literature focused on developing new tools, software, and learning environments, which require extensive additional time and resources (Buitrago Florez et al., 2017; Arcos et al., 2017). There is a lack of guidance for teachers to translate technical skills to students. This study proposes a step-wise and easy-to-implement pedagogy, which can be adapted to different contexts, skill levels, and learning styles. Thirdly, previous studies have relied on positivist and quantitive paradigm to explore the tranfer of technical skills to students e.g.,questionnaire survey. Whilst these have huge merit, yet there may exist a disconnection between reality and statistics. This study uses observation studies and thematic analysis of students' reflective diaries to gain rich data from real-life experiences. The qualitative data from these sources is rich and descriptive to gain an in-depth understanding of opportunities, challenges and future improvement of the developed pedagogy.

This interdisciplinary pedagogy can contribute to curriculum design in international business schools and provide insightful guidance for teaching and learning programming in higher education institutions in a business context. The identified challenges and solutions can help improve the quality of programming-related courses. This paper is structured as follows: Section 2 presents a comprehensive literature review concerning UTAUT, computer science pedagogy for teaching programming and business pedagogy in business schools. Section 3 illustrates a detailed study design, while Section 4 presents the developed pedagogy. Section 5 summarises the outcome of observation studies and content analysis while Section 6 presents conclusions, practical implications and limitations.

## 2. Literature review

### *2.1 UTAUT*

Venkatesh et al. (2003) pointed out that performance expectancy, effort expectancy, social influence, and facilitating conditions play significant roles as direct determinants of user acceptance and usage behaviour towards a particular technology, which formulates the basics of UTAUT. Previous studies have used UTAUT to explore the acceptance of digital textbooks (Sari and Chou Liu, 2023), e-learning technologies (Qiao et al., 2021), ChatGPT (Strzelecki, 2023), and learning management systems (Raza et al., 2021). As this study aims to explore students' acceptance of interdisciplinary pedagogy and learning programming, UTAUT is used as a foundation of this study. Given teaching programming in a business context, performance expectancy means the degree to which the students trust that learning programming can help them improve their performance in their courses and career opportunities (Yildiz et al., 2020). Effort expectancy refers to how easy student perceives learning programming to be. Social influence indicates how students perceive that future employers require necessary programming skills. Facilitating conditions refers to the extent to which students believe that they have access to a supportive learning environment and available resources.

## *2.2 Computer science pedagogy for teaching programming at higher education*

Previous research focused on investigating challenges and strategies associated with computer science pedagogy for teaching programming to computer science students. Chukwunweike et al. (2015) conducted a questionnaire survey to investigate students' attitudes toward programming courses and explored effective strategies for improving computer science pedagogy. Although students were positive at learning programming, factors that caused poor learning of programming included lack of practice, traditional and inappropriate teaching methods, and insufficient instructional materials. Medeiros et al. (2018) conducted a systematic literature review to identify key skills and background knowledge for novice students to learn programming, as well as the challenges of teaching introductory programming. They found that problem-solving and mathematical ability were essential skills for learning programming, while the main challenge was a lack of scaling and personalised teaching. Elcicek and Karal (2020) used content analysis to examine cognitive problems faced by students and instructors during programming teaching and learning. They found these cognitive problems mainly include syntactic errors, misuse of key concepts, and lack of computational thinking skills. Arcos et al. (2017) conducted a national survey on teaching programming in undergraduate computer science programs in Educator. They summarised that e-learning platforms, pair programming, mobile devices, games, and simulations as commonly used teaching strategies. Buitrago Florez et al. (2017) conducted a literature review to explore the impacts of teaching programming, analytical thinking, and computational thinking. They found that constructivist, sociocultural, and pedagogical approaches are needed to create curricula that are geared toward the development of computational thinking and programming skills in higher education. Other researchers focused on developing effective tools, software, and learning environments for teaching programming to computer science students. This includes blended learning approaches (Đambić et al., 2021), achievement degree analysis approach (Allinjawi et al., 2014; Bati et al., 2014;), pair programming and coding dojo (Estacio et al., 2015), mobile applications (Ataxanovich, 2022), course management systems (Benford et al., 2011), e-learning tools (Li, 2016), flipped classrooms (Pattanaphanchai, 2019), and educational escape rooms (Lopez-Pernas et al., 2019).

## *2.3 Business pedagogy in higher education*

Many scholars have contributed to improving business pedagogy in higher education. For example, Berenyi and Deutsch (2018) explored the proper teaching methods based on the opinions of students with limited core knowledge of business courses. They found that students from various fields (i.e., project management, accounting, finance, marketing, and human resources) had varying preferences towards different teaching methods, including listening to lectures, problem-solving training at seminars, individual and group presentations, case studies, and role-playing. Pedagogy such as lectures, classroom discussions, and reading textbooks are found to be more important than case studies, research projects, guest lectures, presentations and computerised learning assignments (Rodrigues, 2004). Ehirheme and Iyiola (2018) pointed out that technology-based training, computer-aided learning, computer-based training, and computer-based testing are useful tools for delivering enhanced business education. Lourenco and Jones (2006) proposed a business education pedagogy that combines tools such as pre-session case studies, entrepreneurial activities like role-plays, theoretical lectures, feedback sessions, and collaborative group work to enhance learning. Balan and Metcalfe (2012) pointed out that team-based learning, poster planning and presentation sessions are effective methods for engaging business students. However, there are only 2 studies examining teaching information communication technology at business schools. James (2013) focused only on facilities while Roussev (2003) merely used Bloom's hierarchy of cognitive skills for teaching programming.

While there are clear intersections between computer programming and business education, their pedagogical approaches have been developed separately. Business education often focuses on critical thinking, and problem-solving in decision-making, whereas computer programming emphasizes technical proficiency, logical reasoning, and computational thinking. There remains a significant theoretical gap in developing an interdisciplinary pedagogy that effectively integrates programming into business education. This study develops an interdisciplinary pedagogy for teaching programming in the business context and assesses its opportunities, challenges, and potential solutions. Figure 1 below captures the current approaches:

## 3. Study design and research methodology

### *3.1 Background of the case study*

This study first develops an interdisciplinary pedagogy. The interdisciplinary pedagogy was adopted to deliver Financial Technology module in Business School at a university in the southwest of England in 2022-23. The cohort consists of 91 male and 38 female final-year undergraduate students. Most of the students are younger than 25, and from different parts of the world including home students (British), and international students (students from Asian (i.e. Myanmar, Vietnam, Jordan, Malaysia, China, etc.), and African (i.e. Botswana, Cameroon, etc.)). The students were from different educational backgrounds, including Banking and Finance, Accounting and Finance, Economics, Business Management with Accounting and Finance, Accounting and Management, as well as Business Management and Economics. The main objective of this module is to teach students the concepts of blockchain, big data, and machine learning. These students were randomly allocated into 5 tutorial groups. The assessment on the module was an individual Python programming task (i.e. a Python file) for either a simple inventory management system or a simple machine learning-based stock price prediction model, along with a 400-word reflective diary. It was expected that the inventory management system would take inputs from users, keep track of items (e.g., name, unit, and value), and produce the summary of total value. The simple machine learning model asks students to use a prediction model (e.g., linear regression, polynomial regression, decision tree, logistic regression, support vector classifier, or artificial neural network) to forecast stock trends using the given dataset. In the reflective diary, students were asked to reflect on learning programming (i.e., What they learned? Which three topics did they find the most interesting and valuable?), completing programming tasks (i.e., Which task did they choose? How effectively were they able to apply their skills? How did they overcome any challenges? What did they find particularly enjoyable or satisfying?) and general reflection about the programming learning on the module (i.e., Their own experience of learning programming throughout the tutorials? How did they manage to complete programming assignment? What would they add or change in the module?).

## *3.2 Thematic analysis*

Thematic analysis and inductive analysis are performed on 129 students' reflective diaries to explore their perceptions and experiences with learning programming (Liu, 2016, Clarke and Braun, 2017). Before undertaking the thematic analysis, we defined key themes to reflect research questions (Crossman and Bordia, 2021). These themes include the value students place on this pedagogy, the opportunities they perceive, the challenges they encounter, and the improvement solutions they suggest. The reflective diaries were systematically reviewed to identify emergent codes and specific patterns. Each emergent code was then carefully examined and categorised under corresponding key themes. This process ensures that the codes align well with the research objective while capturing the nuances of students' reflections. Sub-themes were then generated inductively from emergent codes to provide a deeper understanding of the data (Camilleri, 2024). Suitable constructs of UTAUT are then identified for each sub-theme and emergent code.

## *3.3 Observation study*

Observation study is a powerful tool in educational settings because it allows researchers to gather data from a natural environment and provides real-time insights into how students behave and engage (Cohen et al., 2002; Tondeur et al., 2017). Observation studies are conducted during one of the author’s teaching practice throughout the semester. The researcher acted as the observer and systematically observed classroom engagement levels and students' behaviours. The researcher monitored how students engaged with the learning materials and assignments, how they collaborated with peers, their interactions during the class, and the quality of their assignments. The researcher also observed how students' programming skills develop over time through project-based tasks and kept a note on these observations.

## 4. An interdisciplinary computer science and business pedagogy

The proposed interdisciplinary pedagogy is based on the integration of both computer science and business pedagogy, along with a series of contemporary theories including humanism, scaffolding, constructivism, cognitive load, spiral learning, backward design, problem-based learning, active learning, and psychological safety. The pedagogy design aims at meeting the performance expectancy, effort expectancy, social influence, perceived learning opportunities, and facilitating conditions set by the UTAUT theory. The conceptual framework of this interdisciplinary pedagogy is illustrated in Fig. 1.

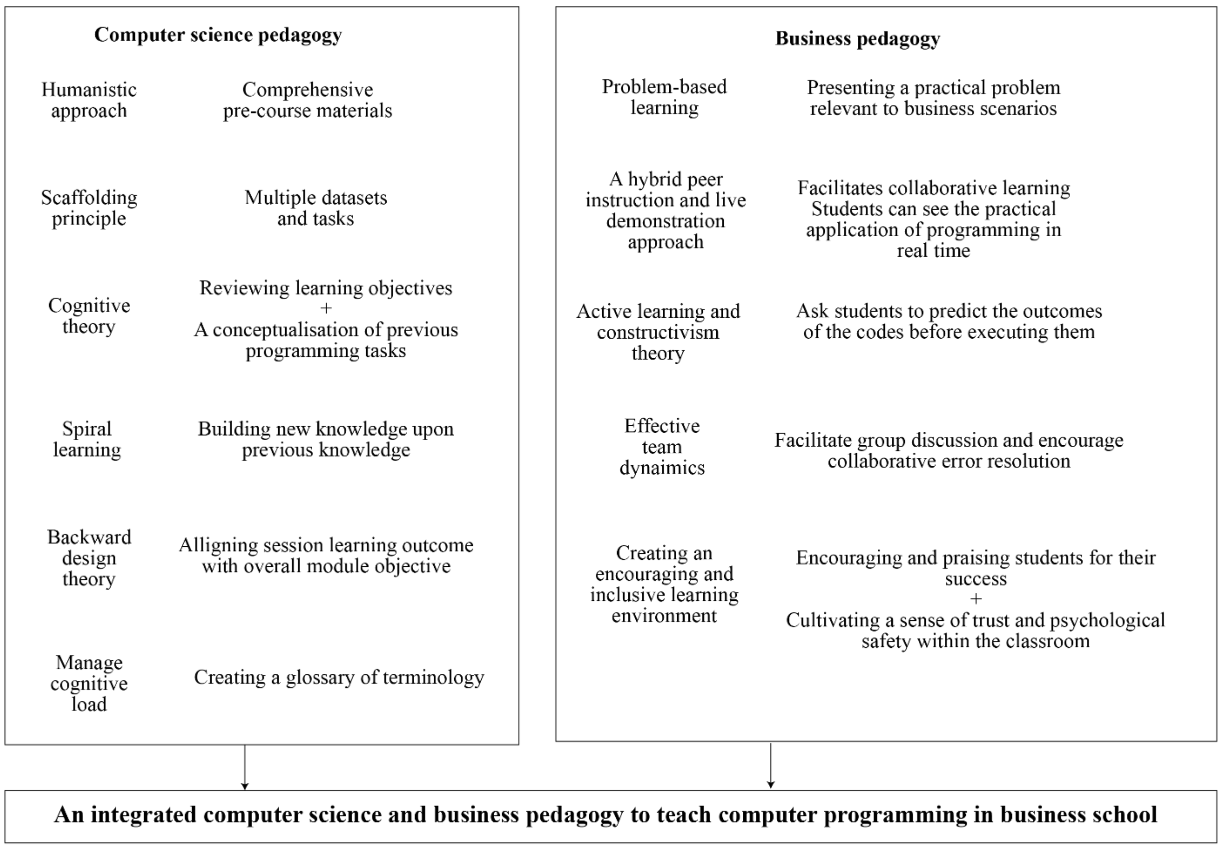
Computer science pedagogy
Humanistic approach
Comprehensive pre-course materials
Scaffolding principle
Multiple datasets and tasks
Cognitive theory
Reviewing learning objectives + A conceptualisation of previous programming tasks
Spiral learning
Building new knowledge upon previous knowledge
Backward design theory
Alligning session learning outcome with overall module objective
Manage cognitive load
Creating a glossary of terminology
Business pedagogy
Problem-based learning
Presenting a practical problem relevant to business scenarios
A hybrid peer instruction and live demonstration approach
Facilitates collaborative learning Students can see the practical application of programming in real time
Active learning and constructivism theory
Ask students to predict the outcomes of the codes before executing them
Effective team dynaimics
Facilitate group discussion and encourage collaborative error resolution
Creating an encouraging and inclusive learning environment
Encouraging and praising students for their success + Cultivating a sense of trust and psychological safety within the classroom
An integrated computer science and business pedagogy to teach computer programming in business school

**Fig. 1.** Conceptual framework of the proposed interdisciplinary pedagogy. *(Source: Authors' own creation/work)*

### *4.1 Integrating humanism and constructivism theory in computer science pedagogy*

In computer science pedagogy, particularly within technology-intensive modules, integrating humanistic approaches is essential to maintaining a student-centered and value-driven educational experience (Hamdan et al., 2014). Humanistic approaches focus on the whole student and their individual needs and personal growth. To facilitate this, we provide comprehensive pre-course materials, including PowerPoint slides, example programming files, datasets, and a reference list of YouTube tutorials and journal articles. Aligning with scaffolding principles (Van, 2002), these resources are designed to help students preview session objectives and content. Recognising that new concepts and theories can often seem abstract without practical application, we incorporate multiple datasets and tasks and require students to apply learned concepts to real-world problems. This approach aligns with constructivist theory and emphasises the importance of active learning and practical problem-solving to reinforce understanding.

## *4.2 Enhancing cognitive development in computer science pedagogy for business students*

Many business students struggle to develop cognitive structures to learn programming concepts (Olds et al., 2007). To address this challenge, we employed cognitive load theory (Sweller, 1994), reviewed learning objectives at the beginning of each tutorial, and provided a conceptualisation and recap of previous programming tasks. This approach helps to manage cognitive load and reduce cognitive overload. To further support learning, we applied spiral learning (Harden, 1999) and strategically built new content upon previously covered knowledge. This iterative approach allows students to deepen their understanding over time and construct knowledge gradually by integrating new ideas with what they have already learned (Brod et al., 2013; Liu et al., 2017; Shing and Brod, 2016). Additionally, we used backward design theory (Wiggins and McTighe, 2005) and aligned the learning outcomes of each session with the overall module objectives and assessment. This helps students see the relevance of each session within the broader curriculum, thereby enhancing their motivation to engage (Deci and Ryan, 2012). After each session, we asked students to create a glossary of terminology, summarising the definitions and purposes of programming commands. This practice not only aids in managing cognitive load but also boosts self-efficacy (Bandura, 1982).

## *4.3 Integrating problem-based learning and active instructional strategies in business pedagogy*

To integrate problem-based learning in business pedagogy, we begin by presenting a practical problem relevant to business scenarios and asking students to think of potential solutions (Haas and Furman, 2008). Because learning is better promoted when learners are engaged in real-world tasks (Merrill and Gibert, 2008), it is important to encourage students to engage in active problem-solving and apply theoretical knowledge to real-world business cases. We utilise a hybrid peer instruction and live demonstration approach to teaching programming concepts (Porter et al., 2011, 2013). This approach not only facilitates collaborative learning, which is key in business education (Leidner and Jarvenpaa, 1995), but also ensures that students can see the practical application of programming in real-time. We provide starter codes as a foundation, then extend these with live demonstration sessions that are directly related to the session's learning outcomes. To assess students' understanding, we ask them to predict the outcomes of the codes before executing codes. This strategy is based on active learning and constructivism theory (Miller et al., 2013), and helps students actively engage with the material, interrogate their own understanding, and identify misconceptions (Soeharto et al., 2019), which is essential in business education.

### *4.4 Enhancing competency through the synergy of individual and group programming tasks*

Integrating principles from both computer science and business pedagogy through intensive practice in individual and group programming tasks can significantly enhance students' programming competencies. According to computer science pedagogy, engaging in practical problem-solving is crucial for skill development (Ivanović and Aleksić, 2016). Furthermore, the application of social constructivism supports business pedagogy with an emphasis on fostering dynamic classroom interactions and stimulating innovative ideas (Yazedjian and Kolkhorst, 2007). To manage cognitive overload and maximise learning performance, we implement follow-up individual tasks, facilitate group discussions, and encourage collaborative error resolution (Crouch and Mazur, 2001). This pedagogy not only enhances social influence and perceived learning opportunities but also promotes student cohesion and trust within smaller groups. Positive interactions in these small groups align with business pedagogy’s focus on effective team dynamics (Williams et al., 2006).

### *4.5 Creating an encouraging and inclusive learning environment*

To foster an engaging and inclusive learning environment, we implemented strategies aligned with business pedagogy principles. By actively encouraging and praising students for their successful completion of tasks, we aim to boost their engagement and performance to ensure that they feel included and respected (Reyes et al., 2012). Recognising that some students might fear making mistakes and avoid participation, we address these psychological barriers by cultivating a sense of trust and psychological safety within the classroom (Erdogan, 2016). Additionally, we acknowledge students' hard work and their contributions of inspiration and energy, particularly after challenging sessions. This approach not only supports the development of a positive learning atmosphere but also enhances the facilitating conditions for effective programming education.

## 5. Research outcome from observation studies and thematic analysis on reflective diaries

The key outcomes from the thematic analysis are summarised in Table 1, along with the UTAUT theory on which it is based.

Table 1. The research outcome of thematic analysis. *(Source: Authors' own creation/work)*

| **Pre-defined key themes** | **Generated sub-theme from emergent codes** | **Emergent codes from reflective diaries** | **Relevance with UTAUT theory** |
|---|---|---|---|
| Value perception of learning programming | Apply skills in completing tasks to gain hands-on experience | • *Data analytic skills.*<br>• *Precise logic and meticulous content.*<br>• *Firsthand experience in applying theoretical concepts.*<br>• *Valuable hands-on experience in software design, user input handling, and error management.*<br>• *Identifying the problem and finding a solution.*<br>• *Bridge the gap between academic knowledge and its practical application.* | Performance expectancy |
| | Feel rewarded and proud by debugging errors and completing tasks | • *The most enjoyable aspect was seeing the system work seamlessly.*<br>• *Operating it successfully was fulfilling.*<br>• *It was also satisfying to have overcome my challenges and produce good concise code.*<br>• *Seeing the error messages disappear gave me a sense of accomplishment.*<br>• *Watching each piece of code come together to create a cohesive programme was the most enjoyable.*<br>• *Satisfying with the visual representation of linear regression* | Self-efficacy |
| | Interactive and active learning approach | • *Smoother iteration over code blocks, making the debugging and visualisation process more intuitive.*<br>• *Facilitated a deeper understanding of the programming concepts covered.*<br>• *Satisfying to see the program become a visual entity rather than just a block of code.*<br>• *Turning theoretical machine-learning concepts into something practical in the form of code that can be run.* | Effort expectancy<br>Social influence<br>Facilitating conditions |
| Challenges in learning programming | Data preprocessing | • *The commas initially present in the data made it incompatible with Python.*<br>• *Deciding which data to utilise and verifying that everything was properly configured*<br>• *Ensuring data quality and preparing it* | Effort expectancy |
| | Error handling | • *Forgetting to put a semicolon somewhere or not properly indenting the lines.*<br>• *Finding and correcting logical errors or unexpected behaviours was especially difficult.*<br>• *Missing spaces required for syntax.*<br>• *A very tedious task.* | Effort expectancy |
| | Translate theoretical | • *Applying the theory to real data and building effective models requires experimentation and adjustment* | Social influence |

| | | | |
|---|---|---|---|
| | knowledge into practical codes | | |
| Solutions to overcome programming challenges | Reviewing teaching materials | • *Using the given example and reviewing the principles I had learned.*<br>• *Reviewing my previous work step by step to fully understand how to apply each input.*<br>• *Looking over my previous work in the practical sessions.*<br>• *Searched again for the PowerPoint used in class.* | Effort expectancy<br>Facilitating conditions |
| | Reviewing code documentation and online resources | • *Using websites that teach coding.*<br>• *Consulted documentation and online resources.*<br>• *A series of YouTube videos.*<br>• *Information from the internet.* | Facilitating conditions |
| | Keeping a detailed note of each programming code and function | • *I would take more detailed notes in the early weeks.*<br>• *Summarising and generalising in terms of skill application.* | Effort expectancy<br>Facilitating conditions |
| | Continuous practices and proactive learning | • *Complete the tutorial tasks at home on my own time.*<br>• *Consistent practice and a proactive approach to problem-solving.*<br>• *Prepare for the course before the seminar by searching for hard words.* | Facilitating conditions |
| | Project planning and task segmentation | • *Making a plan beforehand helped me ensure that I had included all the parts.*<br>• *Breaking down the problem into smaller tasks and using my coding knowledge.* | Facilitating conditions |
| Future improvements in pedagogy design | Collaborative exercises and group work. | • *More opportunities for us to work in groups and review each other's code.*<br>• *Incorporating more collaborative exercises could be beneficial.* | Facilitating conditions<br>Social influence |
| | More materials for independent deep learning | • *More additional materials could be provided on the blackboard that would allow for some more independent development.*<br>• *Providing more challenging exercises can further enhance problem-solving skills.*<br>• *A list of pre-stated common errors could help rectify some of these issues early and easily.* | Effort expectancy<br>Facilitating conditions |
| | Personalised tutorial sessions | • *The pace of learning was a little overwhelming.*<br>• *The pace of the tutorials was rather slow for me.* | Effort expectancy |

## *5.1 Value perception of learning programming*

### 5.1.1 Apply skills in completing tasks to gain hands-on experience.

Through developing real-life projects like inventory management systems or machine learning prediction models, students could translate theoretical concepts, such as data structures, user input handling, and algorithm operations into practical programming applications. This can enable them to enhance their problem-solving, logical reasoning, and data analytics skills.

One student highlighted problem-solving and data analytics skills by stating:

*"I applied my programming and data analytic skills effectively to complete this task. I utilized Python, data structures (i.e., dictionaries), user input handling, and basic arithmetic operations to implement the inventory management system."*

Another student noted the logical skills in programming:

*"I discovered that programming necessitates precise logic and meticulous content verification, such as handling user input and ensuring data integrity."*

Students observed that programming tasks significantly reinforced their theoretical concepts and provided practical experience in software design and error management. As one student reflected:

*"Learning programming throughout the tutorials and working on the Jupyter Notebook task provided a firsthand experience in applying theoretical concepts. Managing to complete the assignment reinforced my understanding of data structures, file handling, and user input validation."*

Additionally, another student emphasised the value of hands-on experience in software development:

*"The Inventory Management System project provided valuable hands-on experience in software design, user input handling, and error management. It showcased the significance of well-organized, readable code and the iterative nature of software development, where continuous improvement is essential for building reliable and maintainable systems. Moving forward, applying these lessons to more complex projects will undoubtedly contribute to the growth and proficiency of my programming skills."*

These reflections illustrate how engaging with programming tasks allows students to bridge the gap between theoretical knowledge and practical application, thereby deepening their understanding and enhancing their technical skills. From the observation study, we noticed that it took time for students to transform their theoretical knowledge into programming codes. They thought about the logic behind the algorithms and referred to lecture notes and teaching materials for useful functions and coding commands.

### 5.1.2 Feel rewarding and proud by debugging errors and completing tasks.

At the end of the semester, 80% of the 129 students successfully completed and submitted a satisfactory programming file. 10% of students reported that, despite their efforts, they were unable to fully complete the programming task. However, more than 90% of students expressed a strong sense of accomplishment and pride, particularly in overcoming programming errors, seeing their code run smoothly, and achieving functional project outcomes. They found the coding tasks enjoyable, reflecting on their journey from having no programming experience to successfully executing a cohesive programming project and improving data handling efficiency.

One student articulated this sense of achievement:

*"While I found this difficult as it was my first time coding, I found myself enjoying the task. I was not able to accomplish a fully functioning prediction model; however, I was able to overcome many challenges to get the program to where it is now. Slowly, one by one, seeing the error messages disappear gave me a sense of accomplishment."*

Another student shared their enjoyment in seeing their work come together:

*"The most enjoyable part of this assignment was watching each piece of code come together to create a cohesive program. Additionally, once I had formed the basic code, adding in more complex features such as search and updating the inventory was challenging but rewarding."*

The observational study further revealed that students exhibited visible signs of satisfaction, such as smiling and praising each other, upon completing challenging programming tasks during tutorials. Additionally, students often shared their completed work with their peers, reflecting a sense of pride and enthusiasm for their accomplishments.

### 5.1.3 Interactive and active learning approach.

Kaggle Notebook was utilized as the platform for teaching and learning programming due to its interactive features, which offer clear and immediate feedback on errors and provide visual representations of programming results. Students noted that this interactivity significantly enhances the debugging process by transforming the program from a mere block of code into a visual entity, thereby making the process more intuitive. One student remarked:

*"The interactive nature of Kaggle Notebook facilitated a smoother iteration over code blocks, making the debugging and visualisation process more intuitive."*

Another student expressed their appreciation for the visual feedback:

*"This is due to how satisfying it was to see the program become a visual entity rather than just a block of code. I have thoroughly enjoyed learning the basics of Python and think it's been a useful part of the course."*

The observational study further revealed that students often searched for keywords related to error messages to find solutions from online resources. Many found this approach effective in addressing programming errors. Additionally, the immediate output provided by Kaggle Notebook allowed students to engage in trial-and-error practice, thereby reinforcing their knowledge.

## *5.2 Challenges met in learning programming.*

### 5.2.1 Data preprocessing

Some students found data preprocessing to be particularly challenging. Difficulties encountered included issues with data compatibility with Python, verifying data configurations, and ensuring data quality. One student described their experience with data cleaning:

*"One challenge I faced in completing the task was cleaning the data. The commas initially present in the data made it incompatible with Python, so to complete the task I had to remove them."*

Another student highlighted the complexities involved in preparing data for analysis:

*"Ensuring data quality and preparing it for the linear regression model was more intricate than I initially thought."*

The observational study also suggests that most students repeatedly encounter errors related to data types (e.g., arrays and data frames), missing values, and incorrect data formats. We also noticed that few students rely on inefficient methods or manual inspection of data, instead of more advanced techniques and functions.

### 5.2.2 Error handling

Students reported frequent challenges with both careless errors and technical errors during their programming tasks. Careless errors included issues such as forgetting a semicolon, improper indentation, missing spaces for syntax, and neglecting to use pre-defined variable names. Technical errors involved problems like undefined functions and variable names. One student shared their frustrations with these errors:

*"I truthfully found it quite challenging. I am not very good at coding, so I did my best, but I feel I could have made a much better effort had I given it more time. I found it very frustrating as I kept running into issues of forgetting to put a semicolon somewhere or not properly indenting the lines, which would mess up the code. I will likely not do any coding again in the future as I found it really tedious, and I commend anyone good at it."*

Another student highlighted issues with syntax and logic:

*"I am quite careless and will often miss spaces required for syntax. An additional problem I had was my first working code failed to show an accurate inventory value as it didn't multiply the value of an item by its volume, and so I had to rewrite the total_value line."*

Additionally, some students found the error-handling process particularly tedious and time-consuming. As one student reflected:

*"The process was very humbling, as there would be a constant fix or repairment needed within my code, making it a very tedious task."*

Observations revealed that students often misused pre-defined function names. For instance, they would use "arrange" instead of the intended function name due to its general usage as an English word, despite initial instructions on the correct terminology.

#### 5.2.3 Translate theoretical knowledge into practical codes

Translating theoretical knowledge into programming code proved to be a time-consuming and logic-intensive task for students. This process requires careful consideration of how to utilise libraries and functions to process real-world datasets effectively. One student articulated this challenge:

*"One of the challenges was translating theoretical knowledge into practical code. While the principles of linear regression are relatively straightforward, applying the theory to real data and building effective models requires experimentation and adjustment."*

Observational studies revealed that students were good at explaining the underlying concepts and workflow when provided with a case study example. However, they struggled with selecting and applying the appropriate functions to implement their solutions. This discrepancy highlights the gap between understanding theoretical concepts and applying them effectively in programming tasks.

### *5.3 Solutions to overcome challenges*

#### 5.3.1 Reviewing teaching materials and previous works

Students found that example Python files, PowerPoint slides, associated reading lists, and their own previous work are very beneficial. These resources were particularly useful for overcoming challenges and reinforcing their understanding of coding concepts. One student explained how they utilised her previous work:

*"I overcame my challenge in this case just not being too confident with coding by reviewing my previous work step by step to fully understand how to apply each input."*

Another student noted the relevance of earlier tutorials to their current assignment:

*"In the completion of this assignment, first of all, I found the knowledge we learned in the fifth week when we were at a coffee shop, and this time it was the use of inventory management. I think there are similarities between the two, so I can add the part of inventory management based on the original coffee shop and then modify it."*

Observation studies revealed that students frequently reviewed PowerPoint slides and lecture notes from earlier tutorials and revisited those learned functions when we are doing a real case study in the final week. This practice helps them reinforce their understanding of functions and ability of translating theorietical understanding into practical codes.

### 5.3.2 Reviewing documentation and online resources

Given that many Python projects are open-source, students have access to a wealth of online tutorials and resources to address specific errors. They found that utilising documentation and online resources is a valuable approach to problem-solving in programming. One student described their reliance on these resources:

*"Throughout the development process, I often had to rely on research and problem-solving. I frequently consulted documentation and online resources. Also, watching a series of YouTube videos and reading up on how to perform certain tasks and then implementing this was tricky."*

Another student emphasised the role of trial and error and online information in overcoming challenges:

*"A lot of reading and experimenting went into finding a solution that worked for me to overcome issues. I got through mainly by trial and error and information from the internet. It took numerous attempts, but the satisfaction appears once the program works."*

These reflections underscore the importance of external resources in navigating programming challenges and achieving successful outcomes. Observation studies further validated that some students frequently browse documentation or search keywords online when they get stuck in programming tasks.

### 5.3.3 Keeping a detailed note of each programming code and function

Students expressed that having a glossary of terminology and maintaining detailed notes would be beneficial for summarising the definitions and purposes of each programming code. This approach could help manage cognitive load and enhance their learning experience. As one student noted:

"*If I had to do it again, I would take more detailed notes in the early weeks, as these were the skills I struggled with while programming. I presume as the skills were not fresh in my mind. I found this task relatively challenging but overall enjoyed the process."*.

This reflects the value of early and detailed note-taking in mitigating challenges related to skill retention and application. Another student mentioned:

"*I am good at summarising and generalising in terms of skill application. This provided me with a clear direction to complete my final selection, and I could find the knowledge I wanted to apply from my previous summaries*."

This highlights the effectiveness of summarisation and note-taking in providing clarity and direction during complex tasks. The observation study also indicates that students who kept detailed notes and a glossary of terminology tended to grasp new concepts and functions more efficiently.

### 5.3.4 Continuous practices and proactive learning

Continuous practice and proactive learning are essential for students, particularly those new to programming, to become familiar with the programming environment and compiler. As noted by Rahman et al. (2018), novice learners benefit significantly from engaging in regular practice to become accustomed to programming syntax, structure, and style. One student highlighted the importance of dedicated extra time:

*"Dedicating extra time outside lectures is crucial to enhancing one's knowledge and skills. I found this proactive approach pivotal in fully comprehending and excelling in the subject matter."*

Another student appreciated the gradual progression:

*"The approach taken by tutorials provided a gradual progression from basic concepts to more complex tasks, making it easier for the average beginner programmer to get on board. Being able to complete the programming tasks required consistent practice and a proactive approach to problem-solving."*

Observations indicated that some students preferred to think ahead and explore alternative solutions to programming tasks, reflecting their commitment to deepening their understanding and problem-solving abilities.

### 5.3.5 Project planning and task segmentation

Before beginning coding, it is crucial for students to thoroughly understand project requirements to develop a comprehensive project plan. By deconstructing the project into smaller, manageable steps, students can effectively connect project requirements with relevant programming knowledge. One student noted the value of this approach:

*"I found completing a series of small programming tasks during the tutorials really helped me solidify my understanding of that specific skill and help was available when needed. This was vital for the completion of the assessed programming task as I was able to walk away from the tutorials feeling comfortable that I could implement that piece of programming into my own work."*

Another student emphasised the effectiveness of this strategy:

*"I was fairly effective when applying my skills, especially when breaking down the problem into smaller tasks and using my coding knowledge."*

These reflections underscore the significance of breaking down complex projects into manageable components and utilising structured learning approaches to enhance programming proficiency. The observation study also indicates that students achieve higher efficiency in completing big projects if they use a diagram or flowchart to segment tasks.

## *5.4 Future improvements in pedagogy design*

### 5.4.1 Collaborative exercises and group work

Social interaction in programming education can significantly enhance students' cognitive strategies and foster critical thinking (Ideris et al., 2018; Othman et al., 2015). Encouraging collaborative exercises and group work, such as peer reviews of code and shared documents for common mistakes, shortcuts, and useful programming commands, can be particularly beneficial. One student observed:

*"Incorporating more collaborative exercises could be beneficial as I learn well from talking on the subject. It could also be helpful for classmates to discuss mistakes or successes and shortcuts they found while programming because it will advance learning and could speed up knowledge intake as well."*

Another student added:

*"My suggestion for this study is that adding more programming assignments, specifically group assignments, will make it more interesting. Students would be able to learn from one another while developing collaboration and teamwork skills, which are crucial for their future careers."*

These reflections highlight the value of integrating social learning strategies in programming education to enhance both individual and group learning experiences.

### 5.4.2 More materials for independent deep learning

Teachers must recognise that learners begin from varying levels of understanding and progress at different rates. Consequently, some students may achieve more during a project period than others, necessitating tailored support and the provision of additional, more advanced tasks. Students with a strong interest in programming often engage in extensive practice to deepen their knowledge (Hall et al., 2004). By exploring advanced programming concepts and utilising supplementary materials and literature, students can further enhance their problem-solving skills (Turner & Baskerville, 2013). Student feedback underscores the importance of challenging exercises:

*"Providing more challenging exercises or real-world scenarios to apply programming skills could improve problem-solving capabilities."*

Another student highlighted the importance of lists of common errors:

*"Having a file on Blackboard with a list of common errors and demonstrating how to handle these issues could help students troubleshoot effectively. This would make the learning experience more engaging and efficient."*

This highlights the need for more learning materials, especially challenging exercises and reference lists of common errors.

5.4.3 Personalised tutorial sessions for students with different levels of programming experiences.

Students with no prior programming experience found the tutorial pace to be somewhat rushed and overwhelming. One student remarked:

*"The pace of learning was a little overwhelming. Learning programming felt like picking up a new language, and the speed made it confusing at times. It would be beneficial to adjust the pace so that it's less overwhelming for beginners, while still keeping advanced learners engaged."*

Conversely, students with previous programming experience felt that the pace was too slow and suggested a more accelerated approach. As one student noted:

*"I found the pace of the tutorials to be rather slow for me, though I understand the importance of ensuring everyone comprehends the material before moving on."*

To address these differing needs, it may be useful to offer separate tutorials tailored to students' experience levels. For those with previous programming knowledge, tutorials could focus on developing critical and logical thinking skills by encouraging the exploration of different solutions and optimising approaches to tasks. On the other hand, beginners would benefit from tutorials designed to familiarize them with programming environments and basic functions, ensuring a solid foundation before moving on to more complex concepts.

## 6 Discussion and conclusion

Firstly, this study addresses the theoretical gap in teaching programming in business schools by proposing an interdisciplinary pedagogy. This interdisciplinary pedagogy can promote computer science skills in a business context by integrating humanism and constructivism theory and enhancing cognitive development in computer science pedagogy, integrating problem-based learning and active instructional strategies in business pedagogy, enhancing competency through the synergy of individual and group programming tasks, as well as creating an encouraging and inclusive learning environment.

Secondly, observation study and thematic analysis are conducted to reveal insights related to the value perception of the novel pedagogy, opportunities, challenges, and solutions for future improvement. This study highlights that students perceive significant value in this interdisciplinary pedagogy as it enhances students' capability to solve real-world problems and helps them gain hands-on experiences. Students feel accomplished through debugging errors and applying their theoretical knowledge in real-world projects, demonstrating the effectiveness of the proposed pedagogy in increasing engagement and perceived value (Corney, 2010). Meanwhile, students faced several challenges, including preprocessing raw data, handling errors, and translating theoretical concepts into practical codes. These challenges are consistent with findings in the literature, which identify similar obstacles in programming education (Najafabadi et al., 2015; Oweida et al., 2020). The proposed solutions, such as reviewing teaching materials, utilising online resources, maintaining detailed notes, and engaging in continuous practice, are supported by previous research that highlights the importance of iterative learning and proactive problem-solving strategies (Rahman et al., 2018; Ideris et al., 2018).

This study is based on business students from different countries, especially Great Britain, Asia, and Africa. Thus, the proposed interdisciplinary pedagogy offers valuable implications for international business education, especially in terms of computer programming. By addressing the diverse needs of students with varying levels of programming experiences and incorporating collaborative exercises for real-world problems, this interdisciplinary pedagogy supports computer science technique learning and promotes critical thinking among business students. Firstly, by prioritising student-centred learning, this pedagogy integrates humanistic principles and constructivist theory to acknowledge diverse backgrounds, experiences and needs of business students. Secondly, recognising the unique cognitive demands of programming, especially for students with a primary focus on business, this pedagogy incorporates spiral learning and backward design theory to support and enhance cognitive development. Thirdly, this pedagogy combines problem-based learning with peer instruction and live demonstration to engage students in hands-on and practical programming tasks that solve real-world business challenges. Moreover, to optimise learning outcomes, this pedagogy balances individual and group programming tasks and recognises the importance of both independent problem-solving and collaborative learning. This study also has implications for public policy and educational practices, as it suggests the need for curriculum adjustments to accommodate computer science learning in business schools.

Despite these contributions, the study has limitations. This is just a new start to implementing an interdisciplinary pedagogy in teaching technical computer programming in business school. The findings are based only on final-year students at the business school in a post-92 university and may not be generalisable across all educational settings. The observational and thematic analysis may not capture the full spectrum of students experiences and challenges. Future research should explore the applicability of the proposed pedagogy in different types of universities, levels of students, and educational backgrounds. Quantitative study and other theories could be used to further analyse business students perception on this interdisciplinary pedagogy.

## Claim

This study includes human participants while all ethical approvals have been approved by Research, Business and Innovation, University of the West of England.

## References

Alavi, M., Wheeler, B.C. and Valacich, J.S. (1995), “Using IT to reengineer business education: An exploratory investigation of collaborative telelearning”, *MIS quarterly*, Vol. 19 No. 3, pp.293-312.

Allinjawi, A.A., Al-Nuaim, H.A. and Krause, P. (2014), “An achievement degree analysis approach to identifying learning problems in object-oriented programming”, *ACM Transactions on Computing Education*, Vol.14 No.3, pp.1-15.

Arcos, G., Aguirre, G. L., Hidalgo, B., Rosero, R. H., and Gómez, O.S. (2018), “Current Trends of Teaching Computer Programming in Undergraduate CS Programs: A Survey from Ecuadorian Universities”, *KnE Engineering*, Vol.3 No.1, pp.253-275.

Ataxanovich, M.U. (2022), “Methods of teaching and improving web programming in higher education organizations”, *International Conference on Information Science and Communications Technologies (ICISCT), Tashkent, Uzbekistan*, pp.1-3.

Balan, P. and Metcalfe, M. (2012), “Identifying teaching methods that engage entrepreneurship students”, *Education+ Training*, Vol.54 No.5, pp.368-384.

Bandura, A. (1982). “Self-efficacy mechanism in human agency”, *American Psychologist*, Vol.37 No.2, pp.122-147.

Bati, T.B., Gelderblom H. Biljon, J. (2014), “A blended learning approach for teaching computer programming: design for large classes in Sub-Saharan Africa”, *Computer Science Education*, Vol.24 No.1, pp.71-99.

Benford, S., Burke, E., Foxley, E., Gutteridge, N. and Zin, A.M. (1993), “Early experiences of computer-aided assessment and administration when teaching computer programming”, *Research in Learning Technology*, Vol.1 No.2, pp.55-70.

Berényi, L. and Deutsch, N. (2018), “Effective teaching methods in business higher education: a students’ perspective”, *International Journal of Education and Information Technologies*, Vol.12, pp.37-45.

Brod, G., Werkle-Bergner, M. and Shing, Y.L. (2013), “The influence of prior knowledge on memory: a developmental cognitive neuroscience perspective”, *Frontiers in Behavioral Neuroscience,* Vol.7, pp.139.

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S. and Danies, G. (2017), “Changing a generation’s way of thinking: Teaching computational thinking through programming”, *Review of Educational Research*, Vol.87, No.4, pp.834-860.

Camilleri, M.A., 2024. Factors affecting performance expectancy and intentions to use ChatGPT: Using SmartPLS to advance an information technology acceptance framework. *Technological Forecasting and Social Change*, Vol. 201, 123247.

Chukwunweike Nwangwu, E. (2015), “Attitudes of Computer Education Students towards Teaching and Learning of Programming Courses in Nigerian Higher Education: The Way Forward”, *International Journal of Technologies in Learning*, Vol.22, No.4, pp.13-22.

Clarke, V. and Braun, V. (2017), “Thematic analysis”. *The Journal of Positive Psychology*, Vol.12, No.3, pp.297-298.

Cohen, L., Manion, L. and Morrison, K., 2002. *Research methods in education*. Routledge.

Corney, M., Teague, D. and Thomas, R. (2010), “Engaging students in programming”. *In Proceedings of the twelfth Australasian computing education conference*, pp.63-72.

Crouch, C.H. and Mazur, E. (2001), “Peer instruction: Ten years of experience and results”, *American Journal of Physics*, Vol.69, No.9, pp.970-977.

Crossman, J and Bordia, S (2021). “Handbook of Qualitative Research Methodologies in Workplace Contexts”, Elgar Publications.

Đambić, G., Keščec, T. and Kučak, D. (2021), “A blended learning with gamification approach for teaching programming courses in higher education”, In *2021 44th International Convention on Information, Communication and Electronic Technology*, pp.843-847.

Deci, E.L. and Ryan, R.M. (2012), “Self-determination theory”, *Handbook of Theories of Social psychology*, Vol.1 No.20, pp.416-436.

Ehirheme, P.E. and Iyiola, O. (2018), “Rethinking teaching in business education with current trends in technology”, *Nigerian Journal of Business Education*, Vol.3, No.1, pp.347-360.

Elçiçek, M. and Karal, H. (2020), “Cognitive problems in the process of programming teaching in higher education: Learner-instructor experiences”, *Turkish Online Journal of Qualitative Inquiry*, Vol.11, No.1, pp.140-160.

Erdogan, C. (2016), “Analysis on the relationship between trust culture and prejudices in primary schools”, *Eurasian Journal of Educational Research*, No.63, pp.153–168.

Estacio, B. et al. (2015), “Evaluating the Use of Pair Programming and Coding Dojo in Teaching Mockups Development: An Empirical Study,” in *48th Hawaii International Conference on System Sciences*, pp.5084–5093.

Haas, C. and Furman, N. (2008), “Operation Recreation: Adventure Challenge: Teaching Programming through Problem-Based Learning Theory”, *SCHOLE: A Journal of Leisure Studies and Recreation Education*, Vol.23, No.1, pp.97-101.

Hamdan, A.R., Jiar, Y.K., Talib, R., Naim, H.A., Jaffri, H., Samah, N.A., Abu, B., Kosnin, A.M., Daud, K. and Xi, L. (2014), “Lecture-Centred or Student-Centered: A Case Study in a Public University”, Journal of Education and Practice, Vol.5, No.10, pp.1-8.

Harden, R.M. (1999), “What is a spiral curriculum?” *Medical Teacher*, Vol.21, No.2, pp.141-143.

Ideris, N., Baharudin, S.M., and Hamzah, N. (2019), “The effectiveness of scratch in collaborative learning on higher-order thinking skills in programming subject among year-six students”, *4th ASEAN Conference on Psychology, Counselling, and Humanities*. Atlantis Press.

Ivanović, M. and Aleksić, V. (2016), “Introductory Programming Subject in European Higher Education”, *Informatics in Education*. Vol.15, No.2, pp.163–182.

James, O. (2013), “Strategies for Enhancing the Teaching of ICT in Business Education Programmes as Perceived by Business Education Lecturers in Universities in South Nigeria”, *International Education Studies*, Vol.6, No.10, pp.78-89.

Li, X. (2016), “Application of Cognitive Load Theory in Programming Teaching”, *Journal of Higher Education Theory and Practice*, Vol.16 No.6, pp.57-65.

Liu, L. (2016). “Using generic inductive approach in qualitative educational research: A case study analysis”, *Journal of Education and Learning*, Vol.5, No.2, pp.129-135.

Liu, Z.X., Grady, C. and Moscovitch, M. (2017), “Effects of prior-knowledge on brain activation and connectivity during associative memory encoding”, *Cerebral Cortex*, Vol.27 No.3, pp.1991-2009.

Leidner, D.E. and Jarvenpaa, S.L. (1995), “The use of information technology to enhance management school education: A theoretical view”, *MIS Quarterly*, Vol.19, No.3, pp.265-291.

Lopez-Pernas, S. et al. (2019), “Analyzing Learning Effectiveness and Students’ Perceptions of an Educational Escape Room in a Programming Course in Higher Education”, *IEEE access*, Vol.7, pp.7184221-184234.

Lourenço, F. and Jones, O. (2006), “Developing entrepreneurship education: comparing traditional and alternative teaching approaches”, *International Journal of Entrepreneurship Education*, Vol.4, No.1, pp.111-140.

Marr, B. (2021), “Data strategy: How to profit from a world of big data, analytics and artificial intelligence”. *Kogan Page Publishers*, pp.15-22.

Medeiros, R.P., Ramalho, G.L. and Falcão, T.P. (2018), “A systematic literature review on teaching and learning introductory programming in higher education”, *IEEE Transactions on Education*, Vol.62, No.2, pp.77-90.

Merrill, M.D. and Gilbert, C.G. (2008), “Effective peer interaction in a problem-centered instructional strategy”, *Distance Education*, Vol.29 No.2, pp.199-207.

Miller, K., Lasry, N., Chu, K., and Mazur, E. (2013), “Role of physics lecture demonstrations in conceptual learning”, *Physical Review Physics Education Research*, Vol.9, 020113.

Najafabadi, M.M., Villanustre, F., Khoshgoftaar, T.M., Seliya, N., Wald, R. and Muharemagic, E. (2015), “Deep learning applications and challenges in big data analytics”, *Journal of Big Data*, Vol.2, No.1, pp.1-21.

Nambisan, S. (2017), “Digital entrepreneurship: Toward a digital technology perspective of entrepreneurship”, *Entrepreneurship Theory and Practice*, Vol.41, No.2, pp.1029-1055.

Olds, S.A., McKenna, A. and Pazos, P. (2007), “Work in progress-Promoting conceptual understanding through effective peer discussions in large classes”. *37th Annual Frontiers In Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports*, pp. T1D-7.

Oweida, T.J., Mahmood, A., Manning, M.D., Rigin, S. and Yingling, Y.G. (2020). “Merging materials and data science: opportunities, challenges, and education in materials informatics”, *MRS advances*, Vol.5, pp.329-346.

Pattanaphanchai, J. (2019), “An investigation of students’ learning achievement and perception using flipped classroom in an introductory programming course: A case study of Thailand higher education”, *Journal of University Teaching & Learning Practice*, Vol.16, No.5, pp.36-53.

Porter, L., Bailey, L.C., Simon, B., Cutts, Q., and Zingaro, D. (2011), “Experience Report: A Multi-classroom Report on the Value of Peer Instruction”, *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, pp. 138–142.

Porter, L., Guzdial, M., McDowell, C., and Simon, B. (2013), “Success in introductory programming: What works?”, *Communications of the ACM*, Vol.56, No.8, pp. 34-36.

Qiao, P., Zhu, X., Guo, Y., Sun, Y. and Qin, C. (2021), “The development and adoption of online learning in pre-and post-COVID-19: Combination of technological system evolution theory and unified theory of acceptance and use of technology”, *Journal of Risk and Financial Management*, Vol.14, No.4, pp.162.

Rahman, T.F.B.A., Anuar, N. and Said, R.F.M. (2018), “How the nature of programming and learning materials affects novice learner's motivation and programming ability”, *In Proceedings of the 6th International Conference on Information and Education Technology*, pp.124-128.

Raza, S. A., Qazi, W., Khan, K. A., & Salam, J. (2021), “Social Isolation and Acceptance of the Learning Management System (LMS) in the time of COVID-19 Pandemic: An Expansion of the UTAUT Model”, *Journal of Educational Computing Research*, Vol.59, No.2, pp.183-208.

Reyes, M.R., Brackett, M.A., Rivers, S.E., White, M. and Salovey, P. (2012), “Classroom emotional climate, student engagement, and academic achievement”, *Journal of Educational Psychology*, Vol.104 No.3, pp.700-712.

Rodrigues, C.A. (2004), "The importance level of ten teaching/learning techniques as rated by university business students and instructors", *Journal of Management Development*, Vol.23, No.2, pp. 169-182.

Roussev, B. (2003), “Teaching introduction to programming as part of the IS component of the business curriculum”, *Journal of Information Technology Education: Research*, Vol.2, No.1, pp.349-356.

Sari, H. and Chou Liu, M. (2023), “Exploring e-learning success during the covid-19 pandemic: indonesia empirical study.”, *Croatian Journal of Education: Hrvatski časopis za odgoj i obrazovanje*, Vol.25, No.4, pp.1225-1259.

Shing, Y.L. and Brod, G. (2016), “Effects of prior knowledge on memory: Implications for education”, *Mind, Brain, and Education*, Vol.10, No.3, pp.153-161.

Soeharto, S., Csapó, B., Sarimanah, E., Dewi, F.I. and Sabri, T. (2019), “A review of students’ common misconceptions in science and their diagnostic assessment tools”, *Jurnal Pendidikan IPA Indonesia*, Vol.8, No.2, pp.247-266.

Sollosy, M. and McInerney, M. (2022), “Artificial intelligence and business education: What should be taught”, *The International Journal of Management Education*, Vol.20, No.3, pp.100720.

Strzelecki, A. (2024), “Students’ Acceptance of ChatGPT in Higher Education: An Extended Unified Theory of Acceptance and Use of Technology”, *Innovative High Education*, Vol.49 No.2, pp. 223–245.

Sweller, J. (1994), “Cognitive load theory, learning difficulty, and instructional design”, *Learning and instruction*, Vol.4, No.4, pp.295-312.

Tondeur, J., Van Braak, J., Ertmer, P.A. and Ottenbreit-Leftwich, A. (2017), "Understanding the relationship between teachers' pedagogical beliefs and technology use in education: A systematic review of qualitative evidence", *Educational technology research and development*, Vol.65, No.3, pp.555-575.

Van Der Stuyf RR. (2002), "Scaffolding as a teaching strategy". *Adolescent Learning and Development,* Vol.52, No.3, pp.5-18.

Venkatesh, M. G., Davis, G. B., and Davis, F. D. (2003), "User acceptance of information technology: toward a unified view", *MIS Quarterly*, Vol.27, No.3, pp.425-478.

Williams E. A., Duray R., Reddy V. (2006), "Teamwork orientation, group cohesiveness, and student learning: A study of the use of teams in online distance education", *Journal of Management Education*, Vol.30, No.4, pp.592-616.

Xu, J.J. and Babaian, T. 2021. "Artificial intelligence in business curriculum: The pedagogy and learning outcomes", *The International Journal of Management Education*, Vol.19, No.3, 100550.

Yazedjian, A. and Kolkhorst, B.B. (2007), "Implementing small-group activities in large lecture classes". *College Teaching*, Vol.55, No.4, pp.164-169.

Yildiz, S.N., Cobanoglu, A.A. and Kisla, T. (2020), "Perceived acceptance and use of scratch software for teaching programming: a scale development study", *International Journal of Computer Science Education in Schools*, Vol.4, No.1, pp.53-71.