DOI: https://doi.org/10.59645/tji.v4i1.136

The Journal of Informatics, Vol. 4, Issue 1 (pages 26-55)
ISSN: 2714-1993
Received: December 2023, Published: May, 2024
https://doi.org/10.59645/tji.v4i1.136

****Original Research****

DIFFICULTIES IN LEARNING THE DATA STRUCTURES COURSE: LITERATURE REVIEW

Adam Basigie Mtaho
Department of Information and Communication Technology
Arusha Technical College

Leonard James Mselle
Department of Computer Science and Engineering
The University of Dodoma

Abstract

Data structures is a highly demanding course for computer science students, often associated with high cognitive load and consequently, high failure and dropout rates. This study aimed to identify the primary causes of difficulties in learning data structures course, as reported in the computer science education literature. The study employed a mixed approach. Quantitative analysis methods utilized descriptive statistics, while qualitative analysis involved a systematic literature review method. A total of 99 research publications were searched from digital libraries and used in this study, with 42 papers designated for systematic reviews and analysis after meeting selection criteria. The findings reveal that students in computer science education face several difficulties in learning data structures, stemming from the inherent complexity of the subject, teaching methodologies employed, and individual learner characteristics such as poor student background knowledge and low student motivation. Based on these findings, the study recommends the adoption of new teaching strategies to address the encountered difficulties and enhance the learning experience for students learning data structures.

Keywords: *Data structures, data structures and algorithm, Data structures difficulties, learning data structures*.

1.0 INTRODUCTION

Sajaniemi (2002) defines programming as the act of writing computer programs. It is the science and practice of instructing the computer to solve real-life problems more efficiently. To master programming skills, a novice must learn several programming courses. For most universities, the first programming course that students learn during the first year is known as introductory programming. The next course is called data structures (Sajaniemi, 2002; Shackelford et al., 2005).

The introductory programming course focuses on basic programming concepts such as syntax, loops, conditionals, looping, functions, pointers, and basic problem-solving techniques. This course aims to give students the foundational skills of programming. The data structures course on the other hand is an advanced-level programming course that is mandatory for any computer science student, and it is usually studied after learning an introductory programming course (Fouh et al., 2012; Patel, 2014). The course includes records, linked lists, stacks, queues, trees, graphs, searching and sorting algorithms (Shackelford et al., 2005). It also covers set, interface/abstract data type, and array (Zingaro et al., 2018). The data structures course emphasizes problem-solving skills and algorithmic thinking essential for developing efficient software solutions.

Mastering data structures concepts is crucial as it enables students to solve algorithmic problems in various fields of computer science and engineering. In learning data structures, a student is not only expected to master the principles for effective program design and implementation but also to understand the commonly used data structures, their related algorithms, and most of the design patterns found in programming (Shaffer & Tech, 2010, 2012). Studies show that learning data structures course is very challenging and less motivating for novice students (Danielsiek et al., 2012; Qian & Lehman, 2017; Silva et al., 2019; Su & Zhang, 2021). Due to the difficulties encountered by students in learning data structures courses, the course is reported to result in high failure and dropout rates in computer science education (Gárcia-Mateos & Fernández-Alemán, 2009a; Pérez-sánchez & Morais, 2016).

The difficulties and challenges of learning data structures, along with other programming courses, have consistently drawn the attention of scholars seeking to understand why novice students find programming daunting. Researchers have delved into various factors influencing the difficulties and high failure rates in programming (Gárcia-Mateos & Fernández-Alemán, 2009; Pérez-sánchez & Morais, 2016; Zingaro et al., 2018). While extensive research has explored the difficulties of learning programming globally, only a few have comprehensively explored the underlying causes of difficulties in learning data structures. Recognizing that learning data structures poses challenges stemming from diverse factors, this study aims to comprehensively examine and discuss the hurdles encountered in teaching and learning data structures, as revealed in the literature of computer science education.

The remainder of the document is structured as follows: The objective of the study is presented in Section 2. The overview of cognitive load theory is introduced in section 3. In part 4, the study's methodology is explained. The results of the study are covered in section 5. Section 6 wraps up by offering the study conclusion, suggestions, and opportunities for more research.

2.0 OBJECTIVE OF THE STUDY

The objective of this study is to determine the causes of difficulties in learning data structures courses. The significance of this study is to provide, organize, and document the causes of difficulties in learning data structures, thus making it a valuable resource for instructors and researchers on the topic. The findings of this analysis will benefit individuals who need a reference for such studies on the topic. It will provide a general point of reference for future studies in this area.

3.0 COGNITIVE LOAD THEORY

This section gives an overview of cognitive load theory. The theory has been utilized in this study as the basis for identifying the primary causes of difficulties in learning the data structures course.

3.1 Overview of cognitive load theory

The cognitive load theory is a learning theory that is built upon the idea that working memory has a limited "capacity and duration" to store large amounts of unorganized pieces of new information.

According to empirical research, the human brain is only capable of processing two or three things at once (Sweller, 1988). It will lose data after 20 seconds if it attempts to store more than seven elements at once. Only human long-term memory is capable of permanently storing such information, and only when it is organized and practiced can it be later recalled (Sweller, 1988). Therefore, only a certain quantity of information can be processed at once by working memory and other cognitive processes in humans since information processing takes time and includes several cognitive processes.

According to cognitive load theory, a new phenomenon to be learned can impose some degree of cognitive load on the learner, which must be reduced to make effective learning possible. Thus, in learning, without effort to reduce cognitive load, meaningful learning will not be achieved. Cognitive load theory has been more helpful in guiding instructors to pursue effective instruction design and in developing effective learning materials.

3.2 Types of cognitive loads

Chandler and Sweller (1996) identified three types of cognitive loads, or the mental effort (demand of information processing), that a learner experiences when learning. They include intrinsic cognitive load, extraneous cognitive load, and germane cognitive load. These loads are described as having unique individual characteristics yet having additive properties that impact learning. The total cognitive load is thus a summation of all three loads. For positive learning outcomes, the total load should not exceed the maximum available human cognitive capacity.

The intrinsic cognitive load refers to the load imposed on human cognitive processes due to task complexity. Intrinsic cognitive load occurs when the information presented is complex and "multiple elements must be simultaneously processed in the working memory" instead of "one element at a time" (Owens & Sweller, 2008). intrinsic cognitive load is normally imposed on the learner's working memory by the learning material and by the characteristics of the item to be learned.

The extraneous cognitive load is the load that is caused by the instructional procedures that are used to represent the information to the learner (Schnotz & Kürschner, 2007). This type of cognitive load depends on how information is presented to learners, or the activities required by the learner, and is under the control of the instruction designer. It is also imposed by cognitive processes evoked by the instructional design of a learning task that does not contribute to learning. It occurs when instructional procedures require the learner to split his or her working memory attention "between multiple sources of visual information that have to be integrated for comprehension (Schnotz & Kürschner, 2007).

Germane cognitive load on the other hand is described as "effortful learning resulting in schema construction and schema automation," which enables the working memory to establish reserves for other processes (Schnotz & Kürschner, 2007), 2007, p.476). Germane cognitive load supports learners' motivation and effort to "devote more cognitive resources" to solve assigned tasks, which results in "deep learning" (Moreno, 2004, p.101). germane cognitive load is thus devoted to effortful learning. To raise a germane load, both intrinsic and extraneous loads must be reduced (Sweller, 2010).

Whereas intrinsic and extraneous loads adversely affect learning, germane loads highly influence successful learning. For effective instruction design, it is necessary to enhance the germane load and control both extraneous and intrinsic loads. It is thus emphasized that for better learning results when designing instructional materials, care must be taken to ensure that their total cognitive load does not exceed the human memory processing resources available (Paas et al., 2004).

According to cognitive load theory, the difficulties of learning data structures manifest themselves in the nature or contents of the subject matter and its organization (intrinsic cognitive load), the method used to teach the subjects (extraneous cognitive load), and how the learner's cognitive resources are directed in active learning (germane cognitive load). This study uses cognitive load theory to analyze types of difficulties manifested in learning data structures, associate it with three types of cognitive loads namely intrinsic cognitive load, extraneous cognitive load, and germane cognitive load, and provide recommendations to address such difficulties.

4.0 METHODOLOGY

This section describes how the study was designed and conducted. It covers the research approach, sampling methods, data collection methods, data analysis methods, ethical considerations, and quality control.

4.1 Research approach

This study used mixed methods, incorporating qualitative and quantitative techniques to investigate the causes of difficulties in learning data structures concepts for novice programmers. The qualitative approach involved a systematic literature review to explore existing literature on factors causing difficulties in learning data structures courses. A literature review was conducted based on steps proposed by Kitchenham et al. (2009), involving specifying research questions, database searches, study selection, data extraction, synthesis, and report writing. Google Scholar and databases like ACM, IEEE Explore, ScienceDirect, SpringerLink, and Scopus were used during literature search. Quantitative methods, specifically descriptive techniques were used to quantitatively summarize the content of qualitative data across 42 publications that were used in this study to address the study's main objective.

4.2 Sampling method

Both purposive and snowball sampling methods were used in selection of the papers that were used in this study. The study searched themes attributed to challenges, difficulties, limitations, or barriers to learning data structures and programming courses. The search strategy employed keywords such as "learning data structures" , "programming failure", "data structure difficulties" , "programming difficulties", and "teaching data structures". The study sample comprises programming papers published in reputable journals from 2000 to 2023. Additionally, two papers from 1985 and 1989 were included based on relevance to the discussion by using snowball following their essence in literature. In total, 99 research publications formed the sample of this study, with 42 papers being used for systematic literatures review to meet the study objective. The remaining literature was used in this study to support discussion and interpretation of the results findings.

4.3 Data collection tools

This study primarily used Google Scholar to find research publications. It served as a gateway to other databases like ACM, IEEE Explorer, ScienceDirect, SpringerLink, and Scopus. Additionally, the Google search engine was employed to locate publications not indexed by Google Scholar. Both Google Scholar and the Google search engine allow for keyword searches to identify relevant literature. By using targeted keywords, researchers can efficiently gather publications and documents on their topic of interest. These tools offer access to a vast range of scholarly and non-scholarly sources, facilitating a comprehensive literature review.

4.4 Data analysis

The data analysis in this study employed both qualitative and quantitative approaches. Qualitatively, a content analysis involved the systematic categorization, coding, and interpretation of findings from selected literature to identify recurring themes and patterns related to difficulties in learning data structure courses. Data extracted from the reviewed papers were organized according to research themes. Each theme was descriptively analyzed, indicating its frequency, percentage occurrence, and rank within the cited sample, specifically targeting the study's objective. Through in-depth examination and synthesis, key insights were drawn regarding the nature and implications of these factors, presented thematically for clarity. Comparisons across studies and theoretical frameworks were made to achieve a robust understanding. Explanations were provided based on reported findings, supplemented by a discussion incorporating global literature that both supports and challenges current findings.

4.5 Ethical considerations

This literature study was conducted with ethics in mind, adhering to Wohlin et al. (2012) research ethics principles. To uphold participant rights and advance fairness, inclusion criteria were carefully adhered. During the discussion and interpretation of finding the study used citations to as an attempt acknowledging previous studies as recommended by Wager et al. (2011). Transparency in the collection, analysis and interpretation of data and ethical norms were

DOI: https://doi.org/10.59645/tji.v4i1.136

followed during the study process. These initiatives highlight the dedication to moral rectitude during the evaluation procedure Wohlin et al. (2012).

4.6 Quality control

To ensure quality of the study, data validity was ensured through rigorous selection processes and collective verification. The study adhered to prior research emphasizing data cleaning before analysis. Criteria for paper selection ensure data validity, with collective verification to maintain study quality, aligning with prior research emphasizing data cleaning before analysis (Kadar et al., 2021; Lubua et al., 2022). Overall, the comprehensive approach yielded a robust sample for investigating programming learning challenges.

5.0 RESULTS AND DISCUSSIONS

This section presents the results of the analysis, responding to the major aim of this study, which was to explore the causes of difficulties in learning data structures concepts as reported in computer science education literature.

5.1 Causes of difficulties in learning data structures

Table 1 provides a summary of the most salient causes of difficulties in learning data structures in the literature, indicating the frequency, percentage of occurrence, and rank.

Table 1. Factors contributing to the difficulties in learning data structures course

Factor	**Frequent**	**Percent**	**Rank**
Abstract nature of the data structures concepts	13	31.0	1
Dynamic nature of the data structures concepts	8	19.0	4
Poor instructional methodology	7	16.7	5
Ineffective curriculum organization and implementation	4	9.5	10
Ineffective organization of learning materials,	6	14.3	8
Poor student background knowledge	7	16.7	6
The multidimensional nature of the data structures concepts	9	21.4	3
Difficulties in planning program solution	5	11.9	9
Students' faulty mental model	7	16.7	7
Low student motivation	10	23.8	2

Source: Research data (2024)

Based on Table 1, there are ten main factors contributing to the difficulties in learning data structure courses as reviewed from 42 literature sources. According to the results presented in Table 1, the most reported factor affecting was the abstract nature of the data structures concepts, which had 13 citations amounting to 31%, followed by low student motivation with 10 citations, also amounting to 23.8%. The other remaining factors were the dynamic nature of the data structures concepts (19%), poor instructional methodology (16.7%), ineffective curriculum organization and implementation (9.5%), ineffective organization of learning materials (14.3%), poor student background knowledge (16.7%), multidimensional nature of the data structures concepts (21.4%), difficulties in planning program solutions (11.9%), and students' faulty mental model (16.7%).

5.2. Discussion

This section presents the discussion on how each factor presented above contributes to the difficulties in learning data structures and recommendations to help students understand is presented in the following subsections.

5.2.1 Abstract nature of the data structures concepts

The abstract nature of the data structures concepts is one of the leading causes of difficulty in learning data structures as reported in the literature. Several studies have attributed the difficult of studying data structures to the abstract of the data structures terms and concepts (Allred & Allred, 2009; Anquan et al., 2009; Enstr, 2013; Fairuz & Sadikan, 2012; Fouh et al., 2012; Halim et al., 2012; Patel, 2014; Pérez-sánchez & Morais, 2016; Qian & Lehman, 2017; Supli, 2016a; Wali et al., 2020; Yahong, 2019; Zhu, 2007).

To understand the meaning of the term abstract, let us consider the following example: When a geography student is introduced to volcanic mountains, they have the privilege of seeing a mountain as a land feature (either physically or pictorially). The mountain as a land feature is an elevated area relative to the normal surface. Furthermore, such a student has a chance to see volcanic materials (pictorially or physically) that originate from the earth's crust. In this case, the geography student is privileged with concrete examples of the reality being studied. Concerning computer programming and specifically data structures materials, such opportunities are not there.

The properties of a computer code cannot be seen or touched as a reality. Even when they are pictorially presented, such drawings are still imaginative. Such a situation is what is referred to as "abstract." Data structures concepts, like other programming concepts, are expressed using high-

DOI: https://doi.org/10.59645/tji.v4i1.136

level terms, while implementation details are hidden. In learning data structures, students learn new abstract concepts such as arrays, liked lists, trees, etc. These concepts are not familiar to students. Instructors are challenged to represent such unfamiliar concepts with clarity by using real-life examples in a way that students can quickly comprehend (Patel, 2014). Since they are abstract, they are difficult to understand, interpret, and visualize, which consequently leads to the student's frustration and demotivation.

From a cognitive load theory perspective, teaching student the concept that is abstract and unfamiliar is usually very hard and challenging for novice students to learn and understand. To address the difficulties of learning data structures on abstractness, researchers have recommended different teaching approaches using education visualization tools, case studies, and project-based learning approaches (Fernandez, 2014; Kátai & Osztián, 2022; Moselle, 2014; Vagianou, 2006).

5.2.2 Dynamic nature of the data structures concepts

The dynamic nature of the data structures concepts is another cause of difficulty in learning data structures (Fouh et al., 2012; Supli, 2016a, 2016b; Yahong, 2019). Using the geography analogy again, a volcanic mountain such as Kungwe-Mahale in western Tanzania is a static reality. It does not change when it is sunny, windy, or rainy. The computer program is different. In a text editor program, it is text and static, but in a running program, it is dynamic. That is, it's the state of the program, which many times changes during the program's runtime. The dynamic execution of data structures concepts poses challenges for novice users running programs in traditional compilers, as it occurs invisibly in the background, making it difficult for them to understand (Fouh et al., 2012; Yahong, 2019). Even following them in visualization tools is not easy for novices due to the runtime behavior of the program under execution. Programming constructs such as recursion, loops, functions, dynamic memory allocation, and pointer manipulation, are examples of code dynamics that have been reported to be difficult for novices to grasp.

For one to be a programmer, they must know in advance what will happen when the codes execute, even before writing the codes. However, most students fail to imagine what will happen when the code is executed by the notional machine (Thuné & Eckerdal, 2009). The situation is worse if such a program contains data structures and algorithms. To address the challenges associated with the dynamic execution of the data structures programs, researchers have recommended the use of both static and software visualization tools (Supli, 2016b; Applications & Education, 2013; Urquiza-fuentes, Angel, et al., 2009). Other recommendations include using collaborative problem-solving sessions where students discuss and experiment with dynamic data structures (Kumbo et al., 2023; Naps et al., 2002). Another approach is to use real human game-based videos (Kátai & Osztián, 2022). A Hungarian dance is an example of game that demonstrates how sorting algorithms work (Kátai & Osztián, 2021).

5.2.3 Curriculum organization and implementation

The order in which topics are organized in the computer science education curriculum can contribute to difficulties in learning data structures. Curriculum organization may impose difficulties in learning data structures in three aspects. First, the order of precedence between learning procedural programming as a first programming course (imperative first) or object-oriented programming (object first). Some researchers, such as as Allred and Allred (2009) argue that object first approach imposes more intrinsic cognitive load on novice students than the imperative first. This is because in object first, apart from using abstract concepts and complex syntax which are hard for novices to learn, it also introduces other advanced concepts such as class, object, and encapsulations. These concepts impose much intrinsic cognitive load on the learner since they demand many skills and efforts from the novices.

The second aspect concerns whether the computer science curriculum used in a particular college or university emphasizes practical work over theoretical work during its implementation. If the course focuses more on theory than practical work, it makes the subject hard to learn. This is because the lack of practical work limits students' ability to understand abstract data structures concepts and hinders the development of problem-solving abilities essential for real-world programming tasks (Yu et al., 2019). Additionally, it adversely affects students' program planning and problem-solving skills (Wu et al., 2021).

The third aspect is the consideration of students’ backgrounds. The computer science curriculum that is intended for teaching students with no foundation in mathematics and programming basics should be introduced with the bridging course; otherwise, such students will experience a high intrinsic cognitive load (Wei & Burrows, 2016). To address the difficulties resulting from poor organization of the computer science curriculum, it is recommended to ensure that the computer science curriculum progresses logically from foundational programming concepts to more advanced topics and to regularly review such curriculum to reflect advancements in the field (Garcia & Al-Safadi, 2014). Other researchers, such as Robins et al. (2003) recommend that computer science curricula should focus more on practical work than knowledge when it comes to computer programming courses including data structures.

5.2.4 Ineffective organization of learning materials

Research in computer science education reports poor organization of the learning materials as another cause of difficulties in learning data structures. The poor organization of the learning materials affects learning data structures in many ways. According to Allred and Allred (2009), the organization of the learning materials in most textbooks is not learner-friendly for most novice students. There is a disorganization of topics in data structures textbooks and a lack of connectivity between concepts (Allred & Allred, 2009; Su & Zhang, 2021; Yu et al., 2019). There is also a lack of logical flow and improper organization of the topics and how they are presented (Luukkainen et al., 2012). So far, there is no formal agreement on how topics should be organized in books and learning materials. Each author or instructor organizes them based on his own decisions. As a result, all these shortcomings add both intrinsic cognitive load and extraneous cognitive load to novice learners (Su & Zhang, 2021).

Traditionally, programming learning materials are extracted from books. Again, most programming books are written as “language books,” where the author concentrates on covering the syntax of a language such as C, C++, Java, etc. One should realize that knowledge of language syntax is not the ability to use it to implement correct code. Most instructors prepare their teaching notes following the book's coverage. This approach is biased towards language syntax as opposed to learning programming (Mselle & Ishengoma, 2022). The cognitive load is made light by stating simple concepts such as arrays (to represent the initial data structures) and proceeding to other familiar objects such as liked lists, which are easier to handle than queues, etc. The study by Mtaho et al. (2023) showed that preparing data structures materials with memory-based visualization assisted learners. They further showed that presenting data structures material using three different approaches (text, memory diagram, and animation) in a congruent fashion did reduce both intrinsic cognitive load and extraneous cognitive load while increasing germane cognitive load, hence improving their interest and motivation to learn computer science courses.

To help improve the organization of learning materials in a structured manner with clear navigation and indexing, it is also recommended to ensure there is an interconnection between the way basic data structures are taught and make sure that instructional materials are organized in such a way that they do not add more cognitive load to the learners (Luukkainen et al., 2012; Maurer, 2005).

5.2.5 Poor instructional methodology

The use of poor instructional methodology is also another cause of difficulties in learning in teaching data structures (Anquan et al., 2009; Fouh et al., 2012; Halim et al., 2012; Pérez-sánchez & Morais, 2016; Supli, 2016a; Turner & Zachary, 1999; Yahong, 2019). In this study, we define poor instructional methodology as a teaching approach that does not fully engage the learner in the learning process. Such approach fails to reduce both intrinsic cognitive load and extraneous cognitive load, thereby hindering learners' abilities to devote their maximum cognitive resources to the learning process.

There are several teaching methods and strategies that are used in teaching programming courses in computer science education. Examples of the most common teaching methods used in teaching programming courses are class lectures, laboratory practice, projects, seminars, and tutorials. Others are simulations, animation, games, and videos (Sarpong & Arthur, 2013). Likewise, some popular teaching strategies used to teach programming courses are problem-based teaching, project-based learning, flipped classroom, scaffolding, and inquiry-based learning (Almanza Cortés et al., 2019; Amresh et al., 2013; Chis et al., 2018; Feijóo-García et al., 2021; Sarpong & Arthur, 2013; Yee-King et al., 2017).

One example of a poor teaching method that is mostly used to teach programming is the traditional lecture method. This method involves the presentation of the programming concepts and the provision of worked examples, which are verbally explained by the instructor, followed by the demonstration of codes on the computer using compilers. This method causes a split attention effect because the learned contents, working environments, and demonstrating tools are separate, leading to a high extraneous cognitive load (Romanowska et al., 2018a). Another example of poor teaching practice is the direct use of online or hardcoded presentations with animation and quick transitions that run beyond the human information processing limit (Halim et al., 2012). This practice affects learners because it creates a split-attention effect. This, in turn, leads to a loss of concentration and makes it difficult for students to focus on the lesson material and achieve meaningful learning (Pieter, 2007).

To make learning effective, researchers have recommended the use of lectures along with other teaching strategies (Hidalgo-Céspedes et al., 2016; Sarpong & Arthur, 2013). Some of the effective methods that have been recommended to be used in teaching data structures include complementary congruent visualizations (Burkhard, 2005). Other recommendations include using lectures, group

DOI: https://doi.org/10.59645/tji.v4i1.136

discussions, and hands-on activities (King, 2021). Some of the new approaches are think-pair-share, flipped classroom, and role-play approaches (King, 2021; Lokare & Jadhav, 2017).

5.2.6 Poor student's background knowledge

Several studies have shown that a learner's prior knowledge, particularly in basic programming and mathematics, is the first prerequisite for understanding the data structures course (Alvarado et al., 2018; Fouh et al., 2012; Krause-Levy et al., 2020; Tennyson & Beck, 2018; Wei & Burrows, 2016; Yahong, 2019). Knowledge of programming thresholds such as pointers, procedural abstraction, program dynamics, etc. is also mandatory for students who learn data structures; otherwise, such students will not understand data structures due to its high intrinsic cognitive load.

Studies have shown that there is evidence that if a student fails to understand fundamental programming concepts, which are prerequisites for understanding data structures, they will also fail to understand the data structures course (Sanders & Mccartney, 2016; Tennyson & Beck, 2018; Wei & Burrows, 2016). Allred and Allred (2009) argue that a student who formed non-viable mental models of programming basics cannot understand data structures course. Likewise, Sweller (1988) contends that many would agree that people learn better when they can build on what they already understand (known as schematization or constructivism), but the more a person has to learn in a shorter amount of time, the more difficult it is to process that information.

To help learners strengthen their prior knowledge when learning to program and hence minimize the intrinsic cognitive load, researchers have recommended various approaches. Such approaches include introducing bridge courses (Allred & Allred, 2009; Cooper et al., 2017), and using visualization tools (Colaso et al., 2005; Naps et al., 2003).

5.2.7 Multidimensional nature of the data structures concepts

Gaining an understanding of data structures requires students to possess a wide range of programming abilities, including problem-solving, program-planning, and coding-rule comprehension (Allred & Allred, 2009). The need for all these talents at once places a significant mental strain on students. Due to the wide variety of data structures, there are numerous obstacles within a single subject. Among the issues that students encounter are understanding the operation of various algorithms, such as those for sorting or searching (Fouh et al., 2012); finding it difficult to apply these algorithms when working with data structures in programs (Supli, 2016b); having difficulty formulating solutions (Spohrer & Soloway, 1989); and experiencing difficulties debugging and tracing programs (Qian & Lehman, 2017).

Additionally, there are problems with comprehending fundamental programming ideas such as recursion and nested loops, which call for strong problem-solving abilities (Yarmish & Kopec, 2007). Students often struggle to appropriately use algorithms because they are simply too lengthy and complex for them to understand (Patel, 2014). Other frequent difficulties include understanding problems to formulate solutions and dealing with syntax issues (Qian & Lehman, 2017).

Due to the multidimensional nature of data structures elements, several skills are needed to make data structures learning possible. This in turn increases the intrinsic cognitive load, thus complicating the learning process. Spohrer & Soloway (1989) say students learning advanced programming concepts, like data structures, find it hard to bring everything together. They might understand programming tools like loops or sequences and explain common algorithms; they can also define and explain certain common algorithms and how they work, but they may fail to apply those constructs to solving real programming problems.

To address the high cognitive load imposed due to the multidimensional nature of the data structures course, researchers have recommended the use of multiple teaching tools and strategies. Some of the recommended tools include visualization tools (Gárcia-Mateos & Fernández-Alemán, 2009b; Urquiza-fuentes, Azquez-iturbide, et al., 2009; Végh & Stoffová, 2017). Other recommended teaching strategies include the use of think-pair-share, flipped classroom, role-play approaches, and project-based learning (Fernandez, 2014; King, 2021; Lokare & Jadhav, 2017).

5.2.8 Students' faulty mental models

Learning to code can be tough because students struggle to develop a clear picture in their minds of how programs run (Almadhoun, 2021; Heinsen Egan & McDonald, 2014). In the world of programming, "mental model" refers to how people organize their knowledge. It's basically an internal map, created by each person, that shows how things and systems work in the real world (Norman, 1983).

A mental model is a student's comprehension of how computers work and the fundamental information processing that goes into coding (Bayman & Mayer, 1988). Appropriate mental models refer to a match between one's imagination of a certain reality and the reality itself. A structure such as a queue of passengers in a computing scenario is different from a queue of fruits. But even

DOI: https://doi.org/10.59645/tji.v4i1.136

a simple structure, such as a two-field node, can evoke multiple mental representations from different novices. This property of programming concepts being represented in distinct images by different learners results in what is known as inappropriate mental models. The big problem with an inappropriate mental model is that once such a feature takes hold of a learner’s mind, it distorts all that is supposed to be learned.

Due to the multidirectional nature of data structures concepts, most students find it very difficult to form appropriate mental models for many data structures concepts. Most of the literature points out that the lack of the correct mental models of some programming concepts has a direct impact on the actual implementation of data structures elements (Butler & Morgan, 2007; Danielsiek et al., 2012; Decker & Simkins, 2016). Danielsiek et al. (2012) carried out the study to determine the misconceptions that students had regarding binary search trees and piles. They discovered that while a few students had a proficient understanding of the formal definition of a heap, they were unable to distinguish between heaps that had binary search trees. They describe a situation in which a student accurately identified a heap as a binary tree, but when asked to define a binary tree, he responded that all binary trees are search trees. In another study, Decker and Simkins (2016) found that some students would define a data structures concept in one way and implement it differently. Students with inappropriate mental models end up committing misconceptions when learning to program.

In another study, Yarmish and Kopec (2007) studied a group of students who were nearing proficiency in programming (advanced novices). Their research focused on how these students handled nested loops, arrays, and recursion. Interestingly, the study found that even after classroom instruction, these students still struggled with fundamental concepts like nested loops and recursion, which they had already been exposed to.

Misconceptions resulting from students’ inappropriate mental models in learning programming are even more perverse when learning algorithms. Most students cannot show the dynamic properties of a simple algorithm such as the bubble sort algorithm. If one cannot show the dynamic operation of a bubble sort, which is based on nested loops, it is impossible for such a person to decipher the dynamic properties of a quicksort, which employs recursion (Decker & Simkins, 2016). The level of conceptual difficulty varies from concept to concept. The insight revealed from this review shows that misconceptions in conceptual knowledge have a direct impact on program design and implementation.

To help address the problems associated with faulty mental models, researchers recommend encouraging the provision of more practical works for students, and regularly identifying common misconceptions through formative assessments and addressing them explicitly in class (Gallego-Durán et al., 2023). The use of reflective learning practices where students articulate their understanding and confront misconceptions in peer discussion is also another strategy that can help improve students' mastery of the programming threshold concepts ~~(~~Almadhoun, 2021; Jonsson, 2015).

A study by Milne and Rowe (2002) has further emphasized to help student improve their mental model in learning programming, instructors should focus first on making their students understand specific concepts, which are thresholds in this discipline. Threshold concepts are those basics that form the foundation or building blocks for other concepts (Sanders & Mccartney, 2016). Threshold concepts must be well understood before any other concept is ventured into. For example, if a student does not thoroughly and correctly understand the concept of pointers and their ability to store the address of another variable, together with the operations of pointers, such a learner can't understand the concept of a queue, stack, linked list, or tree. Concerning algorithms, if the student is not well grounded in loops, nested loops, and recursions, they will never be able to understand how sorting algorithms work. Again, it has become more evident that visualization, and specifically memory-based visualizations, provide mechanisms for learners to build viable mental models and surmount the hurdles present in data structures.

5.2.9 Difficulties in planning program solution

Difficulties in planning program solution is another challenge that novice learners face in learning data structures. According to Spohrer and Soloway (1989), program planning difficulties are divided into two groups, namely, construct-based problems and plan-composition problems. Construct-based problems, on the other hand, are problems that make it difficult for novices to learn the correct semantics of language constructs. They manifest themselves in three forms: (i) inability to understand the semantics of the construct due to the natural language used to formulate a problem; (ii) human interpreter problem; failure to interpret the programming construct like the machine; and (iii) inconsistency problem; if the machine will always work in the manner expected by novices.

DOI: https://doi.org/10.59645/tji.v4i1.136

Plan composition problems, on the other hand, are problems that make it difficult for novices to put up plans correctly. According to Spohrer and Soloway (1989), these problems are interdependent, but they always occur because most novice programmers face problems mapping programming expressed in a natural language into an actual programming language. In other words, such novices tend to interpret programming problems by overlooking some of the essential problem requirements, whether implicitly or explicitly. This, in turn, may lead to the wrong solution.

In addition to that, plan-composition problems are problems that make it difficult for novices to put up plans correctly. Such problems may occur in one of the following ways: (i) The plan component is not in the program (missing); (ii) The plan component is in the program, but it is not correctly implemented (malformed); (iii) The plan component is in the program but should not be (spurious); (iii) The plan component is present but has been wrongly positioned in the program (misplaced). Even though they can think critically, some students develop approaches that don't solve the problem. These students struggle to create new solutions and instead try to apply old ones inappropriately (Spohrer et al., 1985).

To address these difficulties, researchers have recommended using teaching strategies and methods that encourage critical thinking and problem-solving, such as think-pair-share (Concept et al., 2002; Rahman et al., 2020; Reddy et al., 2015), and providing problem-solving guidance to novice students (Mtaho, 2023). Another recommended strategy is to use project-based learning methods, whereby students implement projects that apply the use of data structures concepts (Fernandez, 2014). The use of congruent-based visualization has also proved to help students who learn data structures to gradually think computationally and therefore improve their planning ability and problem-solving skills hence improving their programming comprehension (Mtaho, 2023).

5.2.10 Low student motivation

Another factor that affects student achievement in learning data structures is low student motivation to learn the course (Allred & Allred, 2009; Bergin & Reilly, 2005; Enstr, 2013; Hawi, 2010; Park & Ahmed, 2017; Patel, 2014; Santana, 2018; Settle, 2014; Supli, 2016a; Velazquez-Iturbide et al., 2017). The term motivation refers to a student's willingness, need, desire, and determination to take part in the learning process (Bomia et al., 1997). The low student motivation in learning data structures course is caused by several factors. The data structures content is usually deep, logical, and abstract; thus, students tend to experience high cognitive overload at the beginning of learning, such that they feel that the course is overwhelming and thus lose interest in learning the course

(Wang & Bednarik, 2012).

Additionally, the course is inherently problem-solving in nature. Understanding it requires both theory and practice. Even with a solid grasp of the theory, one may still encounter other difficulties in practice. This in turn leads to low cooperation among students and consequently affects their ability to implement the algorithm in a real-life setting (Allred & Allred, 2009). The subject demands too much of students' reasoning and thinking abilities, and as a result, some students, especially those who do not expect to pursue their careers in programming, become demotivated to learn programming (Hawi, 2010).

According to Sweller (2010), there is a strong linkage between student motivation and perceived student cognitive load. When student motivation is high, all available working memory resources are devoted to dealing with both intrinsic cognitive load and extraneous cognitive load (Sweller, 2010). Even if students seem equally enthusiastic about a task, their reasons for that enthusiasm might be different. Studies show that intrinsic motivation, where students are driven by their own interest, leads them to put in more effort to learn and understand the material (Bomia et al., 1997).

One strategy that can help promote learners' motivation in learning data structures courses is using animated models (Pieter, 2007). Such models, according to Hidalgo-Céspedes et al. (2016) should be designed in such a way that they can help the learner associate prior knowledge with new concepts to be learned and which pedagogically map with the used textbooks and instructional learning materials (Romanowska et al., 2018a; Vagianou, 2006).

5.3 Difficulties in learning data structures from the cognitive load theory perspective

Table 2 summarizes the discussion from the previous subsection and identifies how each type of cause of difficulties is associated with types of cognitive load as perceived from cognitive load theory.

Table 2. Factors contributing to the difficulties in learning data structures course as considered from the cognitive load theory perspective

Factor	**Citations**	**Type of cognitive load**
Abstract nature of the data structures	Allred & Allred, 2009; Anquan et al., 2009; Enstr, 2013; Fairuz & Sadikan, 2012; Fouh et al., 2012;	Intrinsic cognitive load

concepts	Halim et al., 2012; Patel, 2014; Pérez-sánchez & Morais, 2016; Qian & Lehman, 2017; Supli, 2016a; Wali et al., 2020; Yahong, 2019; Zhu, 2007.	
Dynamic nature of the data structures concepts	Fouh et al., 2012; Halim et al., 2012; Kátai & Osztián, 2021; Mtaho et al., 2023; Pérez-sánchez & Morais, 2016; Supli, 2016a, Qian & Lehman, 2017; Yahong, 2019.	Intrinsic cognitive load
Poor instructional methodology	Anquan et al., 2009; Fouh et al., 2012; Halim et al., 2012; Pérez-sánchez & Morais, 2016; Supli, 2016a; Turner & Zachary, 1999; Yahong, 2019	Extraneous cognitive load
Ineffective curriculum organization and implementation	Allred & Allred, 2009; Çakıroğlu & Öztürk, 2017; Robins et al., 2003; Yu et al., 2019.	Intrinsic cognitive load, extraneous cognitive load
Ineffective organization of learning materials,	Allred & Allred, 2009; Fouh et al., 2012; Luukkainen et al., 2012; Romanowska et al., 2018b; Su & Zhang, 2021; Yu et al., 2019.	Extraneous cognitive load
Poor student background knowledge	Allred & Allred, 2009; Alvarado et al., 2018; Fouh et al., 2012; Krause-Levy et al., 2020; Tennyson & Beck, 2018; Wei & Burrows, 2016; Yahong, 2019).	Intrinsic cognitive load
Multidimensional nature of the data structures	Allred & Allred, 2009; Fouh et al., 2012; Lahtinen & Ala-mutka, 2005; Patel, 2014; Qian & Lehman, 2017; Spohrer & Soloway, 1989; Supli, 2016b; Wali et al., 2020; Yarmish & Kopec, 2007.	Intrinsic cognitive load, extraneous cognitive load
Difficulties in planning program solution	Butler & Morgan, 2007; Danielsiek et al., 2012; Decker & Simkins, 2016; Spohrer et al., 1985; Spohrer & Soloway, 1989.	Intrinsic cognitive load
Students’ faulty mental model	Almadhoun, 2021; Butler & Morgan, 2007; Danielsiek et al., 2012; Decker & Simkins, 2016; Heinsen Egan & McDonald, 2014; Lahtinen & Ala-mutka, 2005; Yarmish & Kopec, 2007.	Low germane cognitive load, intrinsic cognitive load, extraneous cognitive load
Low student	Allred & Allred, 2009; Bergin & Reilly, 2005;	Low germane

motivation	Enstr, 2013; Hawi, 2010; Park & Ahmed, 2017; Patel, 2014; Santana, 2018; Settle, 2014; Supli, 2016a; Velazquez-Iturbide et al., 2017.	cognitive load, intrinsic cognitive load, extraneous cognitive load

Source: Research data (2024)

As presented in Table 2, the abstract and dynamic nature of the data structures concepts, as well as the multidimensional nature of the data structures, contribute to the learner's intrinsic cognitive load. Other causes of difficulties in learning data structures are associated with the extrinsic cognitive load, which mainly depends on the teaching methodology used by the instructors of the course, the curriculum organization of learning materials, and student background knowledge. As presented in Table 2, factors such as students' faulty mental model and low student motivation are attributed to all three types of cognitive load because improving them depends on the instructor's competence in teaching the data structure concept, how the instructor uses effective teaching tools and approaches that stimulate learning, and the effective utilization of the learner's cognitive resources.

6.0 CONCLUSION, RECOMMENDATIONS, AND PROSPECTS FOR FUTURE RESEARCH

This study has examined, analyze, and discussed the causes of difficulties in learning data structures course as documented in computer science literature. The study has also cognitive load theory as a framework to examine the primary cause of each identified difficulty experienced by novice student in learning data structures course. The study has revealed that difficulties associated with data structures concepts can be attributed to several factors, which include abstract and dynamic nature of these concepts, ineffective curriculum organization and implementation, poor organization of learning materials, inadequate instructional methodology, students' limited background knowledge, the multidimensional nature of data structures concepts, students' faulty mental models, difficulties in planning program solution, and low motivation. Though these are not the only difficulties that exist when learning data structures, it is believed that making the effort to understand each type of difficulty and measure its resolution is crucial to successful teaching and learning data structures.

To address the difficulties encountered by novice students in learning data structures, instructors in computer science education are advised to change their attitudes and strategies in teaching programming. Instructors should avoid what is called a “traditional teaching culture”, a kind of teaching college students that heavily relies on teaching programming courses using slides with limited assessments, and practical and project works. Instructors are encouraged to incorporate more engaging methods like peer marking, peer assessment, and peer discussion. Additionally, instructors can boost student motivation in data structures by conducting workshops showcasing real-world applications in their fields, establishing programming clubs, and fostering regular competitions and demonstrations.

The findings presented in this study provide a new roadmap for researchers to come up with innovative methods to address the causes of difficulties analyzed in this study. The study provides a reference for both course designers and developers of computer programming teaching tools for the effective development of programming teaching tools. The study has used the systematic review method to study the causes of difficulties in learning data structures in general. Empirical studies that examine the difficulties experienced by novices in learning specific data structures topics such as trees, graphs, and linked lists are still needed in the future.

REFERENCES

Allred, J., & Allred, J. (2009). A recipe for game development assignments in A Recipe for Game Development Assignments in CS2.

Almadhoun, E. (2021). Exploratory Study to Uncover Student Mental Models of Singly Linked Lists in the C Programming Language.

Almanza Cortés, D. F., Del Toro Salazar, M. F., Urrego Arias, R. A., Feijoo Garcia, P. G., & De la Rosa, F. (2019). Scaffolded block-based instructional tool for linear data structures: A constructivist design to ease data structures’ understanding.

Alvarado, C., Umbelino, G., & Minnes, M. (2018). The Persistent Effect of Pre-College Compu-ting Experience on College CS Course Grades. 876–881. https://doi.org/10.1145/3159450.3159508

Amresh, A., Carberry, A. R., & Femiani, J. (2013). Evaluating the effectiveness of flipped classrooms for teaching CS1. 2013 IEEE Frontiers in Education Conference (FIE), 733–735.

Anquan, J., Dengwen, G., Qinghong, Y., Lan, W., & Yunqing, L. (2009). Research and Prac-tice of the PBL Model for Data Structure Curriculum. 1512–1515.

Applications, C., & Education, E. (2013). An empirical study on factors influencing the effectiveness of algorithm visualization An Empirical Study on Factors Influencing the Effectiveness of Algorithm Visualization. September. https://doi.org/10.1002/cae.20485

Bayman, P., & Mayer, R. E. (1988). Using conceptual models to teach BASIC computer programming. Journal of Educational Psychology, 80(3), 291.

Bergin, S., & Reilly, R. (2005). The influence of motivation and comfort-level on learning to program. Ppig 17, June, 293–304. https://www.researchgate.net/publication/228969033

Bomia, L., Beluzo, L., Demeester, D., Elander, K., Johnson, M., & Sheldon, B. (1997). The Impact of Teaching Strategies on Intrinsic Motivation.

Burkhard, R. A. (2005). Knowledge visualization: The use of complementary visual representations for the transfer of knowledge. A model, a framework, and four new approaches. ETH Zurich.

Butler, M., & Morgan, M. (2007). Learning challenges faced by novice programming students studying high level and low feedback concepts. 99–107.

Çakıroğlu, Ü., & Öztürk, M. (2017). Flipped classroom with problem based activities: Explor-ing self-regulated learning in a programming language course.

Chandler, P., & Sweller, J. (1996). Cognitive load while learning to use a computer program. Applied Cognitive Psychology, 10(2), 151–170.

Chis, A. E., Moldovan, A.-N., Murphy, L., Pathak, P., & Muntean, C. H. (2018). Investigating flipped classroom and problem-based learning in a programming module for computing conversion course. Journal of Educational Technology & Society, 21(4), 232–247.

Colaso, V., Kamal, A., Saraiya, P., North, C., Mccrickard, S., & Shaffer, C. A. (2005). Learning and Retention in Data Structures : A Comparison of Visualization , Text , and Combined Methods. 1–2.

Concept, P., Impact, P., & Plan, P. (2002). Project Title : Development of the Townend Computer Centre Name of Organization : Townend Citizens ' Association. October 2001.

Cooper, K. M., Ashley, M., & Brownell, S. E. (2017). A bridge to active learning: A summer bridge program helps students maximize their active-learning experiences and the active-learning experiences of others. CBE—Life Sciences Education, 16(1), ar17.

Danielsiek, H., Paul, W., & Vahrenhold, J. (2012). Detecting and understanding students' misconceptions related to algorithms and data structures. Proceedings of the 43rd ACM Technical Symposium on Computer Science Education, 21–26.

Decker, A., & Simkins, D. (2016). Uncovering Difficulties in Learning for the Intermediate Programmer.

Enstr, E. (2013). Dynamic programming – structure , difficulties and teaching.

Fairuz, S., & Sadikan, N. (2012). Role of Interactive Computer Programming Courseware in Facilitating Teaching and Learning Process Based on Perception of Students in. 3(8), 1–5.

Feijóo-García, P. G., Kapoor, A., Gardner-McCune, C., & Ragan, E. (2021). Effects of a block-based scaffolded tool on students' introduction to hierarchical data structures. IEEE Transactions on Education, 65(2), 191–199.

Fernandez, R. (2014). A Cognitive Apprenticeship Approach for Teaching Abstract and Com-plex Skills in an Online Learning Environment. 2.

Fouh, E., Akbar, M., Shaffer, C. A., & Tech, V. (2012). The Role of Visualization in Computer Science Education.

Gallego-Durán, F. J., Compañ-Rosique, P., Villagrá-Arnedo, C. J., García-Sánchez, G. M., Satorre-Cuerda, R., Molina-Carmona, R., Llorens-Largo, F., Viudes-Carbonell, S. J., Real-Fernández, A., & Valor-Lucena, J. (2023). Decoding Student Error in Programming: An Iterative Approach to Understanding Mental Models. International Conference on Human-Computer Interaction, 256–273.

Gárcia-Mateos, G., & Fernández-Alemán, J. L. (2009a). A course on algorithms and data structures using on-line judging. Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE, 45–49. https://doi.org/10.1145/1562877.1562897

Garcia, R. A., & Al-Safadi, L. A. (2014). Intervention Strategies for the Improvement of Stu-dents' Academic Performance in Data Structure Course. International Journal of Infor-mation and Education Technology, 4(5), 383.

Halim, S., Koh, Z. C., Bo, V., Loh, H., & Halim, F. (2012). Learning Algorithms with Unified and Interactive Web-Based Visualization. 6(method 2), 53–68.

Hawi, N. (2010). Computers & Education Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course. Computers & Education, 54(4), 1127–1136. https://doi.org/10.1016/j.compedu.2009.10.020

Heinsen Egan, M., & McDonald, C. (2014). Program visualization and explanation for novice C programmers. Conferences in Research and Practice in Information Technology Series, 148, 51–57.

Hidalgo-Céspedes, J., Mar\'\in-Raventós, G., & Lara-Villagrán, V. (2016). Learning principles in program visualizations: A systematic literature review. Proceedings - Frontiers in Education Conference, FIE, 2016-Novem. https://doi.org/10.1109/FIE.2016.7757692

Jonsson, H. (2015). Using flipped classroom, peer discussion, and just-in-time teaching to in-crease learning in a programming course. 2015 IEEE Frontiers in Education Conference (FIE), 1–9.

Kadar, R., Wahab, N. A., Othman, J., Shamsuddin, M., & Mahlan, S. B. (2021). A study of difficulties in teaching and learning programming: a systematic literature review. International Journal of Academic Research in Progressive Education and Development, 10(3), 591–605.

Kátai, Z., & Osztián, E. (2022). Visualizing algorithms: Schematic computer animations versus realistic dance choreography illustrations. Acta Polytechnica Hungarica, 19(1), 193–210.

King, J. (2021). Combining Theory and Practice in Data Structures & Algorithms Course Pro-jects: An Experience Report. Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, 959–965.

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering–a systematic literature review. Information and Software Technology, 51(1), 7–15.

Krause-Levy, S., Valstar, S., Porter, L., & Griswold, W. G. (2020). Exploring the link between prerequisites and performance in advanced data structures. Proceedings of the 51st ACM Technical Symposium on Computer Science Education, 386–392.

Kumbo, L., Mero, R. F., & Hayuma, B. J. (2023). Navigating The Digital Frontier: Innovative

DOI: https://doi.org/10.59645/tji.v4i1.136

Pedagogies for Effective Technology Integration in Education. The Journal of Informatics, 3(1).

Lahtinen, E., & Ala-mutka, K. (2005). A Study of the Difficulties of Novice Programmers. Acm Sigcse Bulletin, 37(3), 14–18.

Lokare, V. T., & Jadhav, P. M. (2017). A holistic approach for teaching Data Structure Course in the Department of Information Technology. Journal of Engineering Education Trans-formations, 30(Special Issue).

Lubua, E. W., Semlambo, A. A., & Mkude, C. G. (2022). Factors Affecting the Security of Information Systems in Africa: A Literature Review. University of Dar Es Salaam Library Journal, 17(2), 94–114.

Luukkainen, M., Vihavainen, A., & Vikberg, T. (2012). A Software Craftsman ' s approach to Data Structures. 1–16.

Maurer, W. D. (2005). Redesigning the data structures course with user-constructed resizable arrays. Journal of Computing Sciences in Colleges, 20(4), 236–241.

Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming - Views of stu-dents and tutors. Education and Information Technologies, 7(1), 55–66. https://doi.org/10.1023/A:1015362608943

Mselle, L. (2014). Formalization of memory transfer language with C, C++ and java on the mold of register transfer language. ICISA 2014 - 2014 5th International Conference on Infor-mation Science and Applications, 0–3. https://doi.org/10.1109/ICISA.2014.6847404

Mselle, L., & Ishengoma, F. (2022). Memory transfer language as a tool for visualization-based-pedagogy. Education and Information Technologies, 27(9), 13089–13112.

Mtaho, A. B. (2023). Effects of using problem-solving guide and explanatory support in pro-gram visualization tool on reducing students' misconceptions in learning data structure concepts. Bulletin of Social Informatics Theory and Application, 7(2), 125–140.

Mtaho, A. B., Mselle, L. J., & Masoud, M. M. (2023). The Effectiveness of Using a Congruent Visualization Framework on Learning a Data Structures Course. Education & Pedagogy Journal, 2 (6), 60–76.

Naps, T., Guido, R., Darmstadt, T. U., College, M., Dann, W., Cooper, S., Fleischer, R., Techn, C. U., Koldehofe, B., Korhonen, A., Kuittinen, M., Mcnally, M., Leska, C., & Malmi, L. (2003). Evaluating the Educational Impact of Visualization.

Naps, T. L., oßling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., & Angel Veí azquez-Iturbide Rey Juan Carlos, J. U. (2002). Exploring the Role of Visualization and Engagement in Computer Science Education Report of the Working Group on &quot; Improving the Educational Impact of Algorithm Visualization &quot; Acm, 35(2), 131–152.

Norman, D. A. (1983). Some observations on mental models. Mental Models, 7(112), 7–14.

Owens, P., & Sweller, J. (2008). Cognitive load theory and music instruction. Educational Psychology, 28(1), 29–45.

Paas, F., Renkl, A., & Sweller, J. (2004). Cognitive load theory: Instructional implications of the interaction between information structures and cognitive architecture. Instructional Science, 32(1/2), 1–8.

Park, B., & Ahmed, D. T. (2017). Abstracting Learning Methods for Stack and Queue Data Structures in Video Games. 2017 International Conference on Computational Science and Computational Intelligence (CSCI), 1051–1054. https://doi.org/10.1109/CSCI.2017.183

Patel, S. (2014). A Literature Review on Tools for Learning Data Structures. 1–7.

Pérez-sánchez, B., & Morais, P. (2016). Learning Data Structures – Same Difficulties in Differ-ent Countries ? 8540(c). https://doi.org/10.1109/RITA.2016.2619140

Pieter, W. (2007). How to optimize cognitive load for learning from animated modelse. In The Netherlands Organisation for Scientific Research.

Qian, Y., & Lehman, J. (2017). Students ' Misconceptions and Other Difficulties in Introductory Programming : A Literature Review. 18(1), 1–24.

Rahman, M. M., Sharker, M., & Paudel, R. (2020). Impact of infusing interactive and collaborative learning in teaching introductory programming in a dynamic class. Proceedings of the 51st ACM Technical Symposium on Computer Science Education, 1315.

Reddy, P. D., Mishra, S., Ramakrishnan, G., & Murthy, S. (2015). Thinking, pairing, and shar-ing to improve learning and engagement in a data structures and algorithms (DSA) class. 2015 International Conference on Learning and Teaching in Computing and Engineering, 144–151.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming : A Re-view and Discussion. 13(2), 137–172.

Romanowska, K., Singh, G., Dewan, M. A. A., & Lin, F. (2018a). Towards Developing an Effective Algorithm Visualization Tool for Online Learning. 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), October, 2011–2016. https://doi.org/10.1109/SmartWorld.2018.00336

Sajaniemi, J. (2002). An Empirical Analysis of Roles of Variables in Novice-Level Procedural Programs. Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments . IEEE,.

Sanders, K., & Mccartney, R. (2016). Threshold Concepts in Computing : Past , Present , and Future. 91–100.

Santana, B. L. (2018). Increasing Motivation of CS1 Non-Majors through an Approach Contextualized by Games and Media. July. https://doi.org/10.1109/FIE.2018.8659011

Sarpong, K. A., & Arthur, J. K. (2013). Causes of Failure of Students in Computer Program-ming Courses : The Teacher – Learner Perspective. 77(12), 27–32.

Schnotz, W., & Kürschner, C. (2007). A reconsideration of cognitive load theory. Educational Psychology Review, 19(4), 469–508.

Schunk, D. H. (2012). Learning theories an educational perspective sixth edition. Pearson.

Settle, A. (2014). Three Views on Motivation and Programming. 321–322.

Shackelford, R., Mcgettrick, A., Sloan, R., Davies, G., University-calumet, P., Cross, J., Impagliazzo, J., Leblanc, R., & Lunt, B. (2005). Computing Curricula 2005 : The Overview Report. 2004–2005.

Shaffer, C. A., & Tech, V. (2010). A Practical Introduction to Data Structures and Algorithm

Analysis Third Edition (Java Version).

Shaffer, C. A., & Tech, V. (2012). Data Structures and Algorithm Analysis (Vol. 2).

Silva, D. B., Aguiar, R. D. L., Dvconlo, Di. S., & Silla, C. N. (2019). Recent Studies about Teaching Algorithms (CS1) and Data Structures (CS2) for Computer Science Students. Proceedings - Frontiers in Education Conference, FIE, 2019-Octob. https://doi.org/10.1109/FIE43999.2019.9028702

Spohrer, J. C., & Soloway, E. (1989). Simulating Student Programmers. IJCAI, 89, 543–549.

Spohrer, J. C., Soloway, E., & Pope, E. (1985). Where The Bugs Are. April, 47–53.

Su, S., & Zhang, E. (2021). A Game-Based Approach for Teaching Algorithms and Data Structures using Visualizations. 1128–1134.

Supli, A. A. (2016a). Critical Analysis on Algorithm Visualization Study Critical Analysis on Algorithm Visualization Study. September. https://doi.org/10.5120/ijca2016911633

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. Cognitive Sci-ence, 12(2), 257–285.

Sweller, J. (2010). Element interactivity and intrinsic, extraneous, and germane cognitive load. Educational Psychology Review, 22(2), 123–138. https://doi.org/10.1007/s10648-010-9128-5

Tennyson, M. F., & Beck, M. (2018). A study of knowledge retention in introductory programming courses. Journal of Computing Sciences in Colleges, 33(4), 13–20.

Thuné, M., & Eckerdal, A. (2009). Variation theory applied to students' conceptions of com-puter programming. European Journal of Engineering Education, 34(4), 339–347.

Turner, J. A., & Zachary, J. L. (1999). Using Course-Long. 43–47. https://doi.org/10.1145/299649.299674

Urquiza-fuentes, J., Angel, J., Azquez-iturbide, V. E. L., Rey, U., & Carlos, J. (2009). A Survey of Successful Evaluations of Program Visualization and Algorithm Animation Systems. 9(2), 1–21. https://doi.org/10.1145/1538234.1538236.http

Vagianou, E. (2006). Program Working Storage : A Beginner ' s Model. 69–76.

DOI: https://doi.org/10.59645/tji.v4i1.136

Végh, L., & Stoffová, V. (2017). Algorithm animations for teaching and learning the main ideas of basic sortings. Informatics in Education, 16(1), 121–140. https://doi.org/10.15388/infedu.2017.07

Velazquez-Iturbide, J. A., Hernan-Losada, I., & Paredes-Velasco, M. (2017). Evaluating the Effect of Program Visualization on Student Motivation. IEEE Transactions on Education, 60(3), 238–245. https://doi.org/10.1109/TE.2017.2648781

Wager, E., & Wiffen, P. J. (2011). Ethical issues in preparing and publishing systematic re-views. Journal of Evidence-Based Medicine, 4(2).

Wali, H. G., Eligar, V., Mane, V., & Iyer, N. C. (2020). A Channelizing Approach to teach Data Structures. Procedia Computer Science, 172, 581–584.

Wang, P., & Bednarik, R. (2012). During automatic program animation , explanations after animations have During automatic program animation , explanations after animations have greater impact than before animations. February 2016. https://doi.org/10.1145/2401796.2401808

Wei, D., & Burrows, A. N. (2016). Tracking students' performance to assess correlations among computer science programming series courses. Journal of Computing Sciences in Colleges, 32(1), 9–16.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in software engineering. Springer Science & Business Media.

Wu, D., Guo, P., Zhang, C., Hou, C., Wang, Q., & Yang, Z. (2021). Research and practice of data structure curriculum reform based on outcome-based education and Chaoxing plat-form. International Journal of Information and Education Technology, 11(8), 375–380.

Yahong, S. (2019). The Research of Data Structure Teaching Based on Computational Think-ing. Icemeet. https://doi.org/10.25236/icemeet.2019.452

Yarmish, G., & Kopec, D. (2007). Revisiting Novice Programmer Errors. 39(2), 131–137.

Yee-King, M., Grierson, M., & d'Inverno, M. (2017). Evidencing the value of inquiry based, constructionist learning for student coders. International Journal of Engineering Peda-gogy, 7(3), 109–129.

Yu, L., Zheng, X., & Biao, Y. (2019). Research on Data Structure Course Teaching System Based on Open Teaching Model. 2nd Symposium on Health and Education 2019 (SOHE 2019), 441–450.

Zhu, Y. (2007). The Teaching of Data Structures Course for Computer Specialty.

Zingaro, D., Taylor, C., Porter, L., Clancy, M., Lee, C., & Webb, K. C. (2018). Identifying Student Difficulties with Basic Data Structures. 169, 169–177.