Assignment 5 – Group 2

I. Analogies

Constructivism is an educational theory primarily based on the understanding that learners are not blank slates. They have past experiences, personalities, and cultural backgrounds that will all influence the information they are given. Therefore it is important to present information in ways that students can construct their mental model to incorporate new information.

Analogies are a powerful tool for constructivist learning. They utilise previous knowledge to simplify concepts, provide concrete examples for abstract learning and provide more accessible avenues for peer teaching and learning. Because Object Oriented Programming (OOP) is a complex and abstract topic that can be difficult for learners to grasp [1], using analogies has been suggested as a useful tool in teaching their concepts. Despite the benefits of analogies, there are risks involved. Analogies can increase the cognitive load when learners translate between analogy and literal concepts, they are a divergence from learners' mental models, and there is potential for unintended extrapolation of analogous material leading to misunderstandings.

A. Uses

Analogies have proven to be successful in teaching students new concepts across various domains, such as computer science [2]. They are helpful in concretizing, structuring new information, and actively assimilating, which assists learning [3]. Analogies can also be used to overcome misconceptions by restructuring incorrect information and forming new explanatory models. However, it is imperative that analogies are well constructed as a poorly defined or easily misunderstood analogy can lead to students misinterpreting the concept and drawing incorrect parallels, which can be hard to undo [4], [5]. Effective analogies should be semantically and structurally similar to the concept being taught. However, no matter how carefully selected, the comparison will still contain differences between the source and target analogy [4], [5].

Thramboulidis [6] describes adopting a constructivism approach to teaching OOP. A popular fast food chain is used as an example to guide students in exploiting their prior knowledge emanating from real life and building on it the conceptual framework of the OO paradigm. Different sections and machines in the restaurant are used as analogies of different objects that carry out different functions. Upon adopting the new teaching approach, there was a considerable improvement in the course, and results in terms of pass/fail ratio were promising.

University of Auckland

Zhu and Zhou [7] propose an innovative method to teach OOP, first teaching object-oriented methodology and then the language itself. There are six steps to the method, which involve teaching OOP fundamentals using real-world examples. Different types of animals are used as an analogy for inheritance, and a bag of apples is used as an analogy for pointers. It is suggested that this approach provides students with a firm understanding of OOP concepts in any coding language or style, rather than pigeonholing them to the language they learnt in. There is no evaluation done with regards to the success or usefulness of the new methodology, so the effectiveness of the new teaching system cannot be determined.

Several studies have explored the use of analogies in teaching tools for OOP. For example, Tanielu et al. [8] explore a systematic approach to creating interactive learning activities for OOP concepts aligned with fundamental learning outcomes and misconceptions by combining analogies and visualization. The study evaluates the effectiveness of a virtual reality app created to educate OOP concepts by providing students from a computer science course with questionnaires before and after using the application. The study found a significant improvement in the student's understanding of the topic, and the students agreed it was an effective learning tool. However, these results must be taken with a grain of salt as a true experiment design study was not performed. Furthermore, the sample size was quite small (17 participants), and the evaluation could not be correlated to a difference in exam or assignment results.

Lian et al. [1] propose a systematic approach to developing a teaching tool that uses a combination of visualisation and analogies to help students overcome difficulties and misconceptions when learning OOP. An evaluation of the efficacy of the Visualisation of OOP concepts with Analogies (VOOPA) tool was carried out with 253 computer science students. Learning gain and confidence gain of the students were used as measures, with learning gain determined by assessing the students and confidence gain determined by recording students self-perceived competency. The results indicated statistically significant evidence that the student's understanding of OOP concepts improved after using the application. However, there was no statistically significant evidence of a difference in learning gain between students using VOOPA versus the slides. The students' confidence also increased after using both methods, indicating that there is merit in both VOOPA and the slides towards building students' self-perception of ability.

The KEE system described by Kempf & Stelzner [9] combines a step-by-step approach of learning by conceptualisation, learning by analogy, learning by experience, and learning by reinforcement. This again suggests that analogies are a useful tool in OOP education. Research from Nelson et al. [10] proposes different learning types when it comes to OOP. They recommend a guided analogy approach for the "single paradigm programmer" learning type "if OOP is "similar enough" to the learner's prior programming experience, the transfer should be facilitated".

B. Analysis

Overall, using analogies in teaching OOP aligns with the constructivist education theory. By connecting new information to existing knowledge and experiences, learners can construct their own understanding of OOP concepts. Analogies can be a powerful tool for facilitating this process, as they allow learners to make connections between abstract and concrete concepts. The existing examples of the use of analogies in the education of OOP all take something that students are familiar with and use it to demonstrate OOP concepts. There is some variety in the application of these analogies, with some simply being in the course textbook and some existing as part of a visual reality tool. Despite the differing mediums, the intention and basis of the analogies remain the same, helping students to make connections between their experiences and prior knowledge with abstract concepts.

The true effectiveness of analogies in education is very hard to evaluate. None of the studies collated could provide a true experiment design due to ethical limitations and other factors. Although the results are uncertain, they all indicate that analogies can be leveraged as an educational tool. None of the studies found that the use of analogies harmed learning, and in most cases, they improved, especially when utilised in conjunction with other learning methods.

Despite the prevalence of analogies in education, many studies acknowledge their risks. Nelson et al. [10] state that the existing knowledge may interfere with learning a new skill. Where programmers draw flawed analogies between OO concepts and prior knowledge, errors are difficult to overcome. Tanielu et al. [8] describe how analogies carry the danger of being misinterpreted and may give rise to further student misconceptions. This caution is echoed by Lian et al. [1], stating that "analogies need to be designed carefully because students can easily misinterpret them and develop misconceptions".

In brief, provided an analogy is well constructed and applicable to the learner, it can be a great learning tool in teaching abstract concepts, specifically OOP. Analogies leverage previous knowledge to make abstract concepts simpler to understand, exemplifying the constructivism educational theory. When used in conjunction with other educational theories, analogies can be a powerful and useful learning tool.

II. Active Learning

Active Learning encourages students to actively participate in the learning process by engaging in activities aimed at acquiring new knowledge [11]. It is a broad concept that involves anything that does not leave students to passively listen to course content and provides opportunities for students to process new information being presented [12]. Active Learning is strongly related to the constructivist educational theory, which explains that learners construct their own understanding by making sense of new information and fitting it into their mental model [13]. This is developed through "authentic" tasks [13], which Active Learning activities simulate. There are several techniques to incorporate Active Learning into the classroom. These include collaborative learning, problem-based learning, peer teaching, flipped classrooms and project-based or hands-on activities [14], [15]. Many of these techniques have been used to teach object-oriented concepts [16]–[21].

A. Uses

A MOO (Multi-user dimension Object Oriented) is a text-based online virtual reality where internet users can explore rooms and interact with objects in the MOO environment [16]. Towell used MOO as an active-learning environment to teach object-oriented concepts to business students without using a specific programming language [16]. Assignments in the MOO were goal-oriented and involved creating or finding objects in the MOO environment, examining the object's methods and using the appropriate methods to complete a task. Instructors refrained from helping with completing the tasks, with the only hint being that they must "examine" how an object works.

Chen et al. present a project-based curriculum to help students practice and apply object-oriented programming [17]. The curriculum consists of a series of seven project-based assignments, with each corresponding to one of the seven lecture topics. Completion of the projects leads to the implementation of a multimedia communication application.

Herala et al. [18] present a case study where an object-oriented programming course implemented a flipped classroom approach to teaching. Traditional lectures were replaced with 31 short online videos. Students were given a manual containing information about object-oriented concepts, code examples and tutorials, as well as lecture slides as supplementary material. The previously voluntary weekly exercises were made compulsory. The project component of the course was made larger, but completing the project in pairs was allowed and encouraged.

Kaila et al. describe a redesign of an object-oriented programming course where half of the lectures were transformed into Active Learning tutorials [19]. The tutorials were designed to practice the topics taught in the lectures from the same week. The tutorials were based on ViLLE, a web-based collaborative education tool. ViLLE supports a variety of exercises, including coding, quizzes, program visualisation and program simulation. The assessment for most of the exercises is automated and does not limit the number of submission attempts. Collaboration was encouraged in the tutorials; students worked on the exercises in pairs using one computer. The student controlling the mouse and keyboard was switched every fifteen minutes. In the demonstration sessions, students were randomly divided into groups to discuss their solutions to previously completed assignments. A member from each group was chosen to present their solution to the class. For the remainder of the session, groups worked together on an extra assignment.

B. Analysis

It is well known that people learn best when they are actively engaged in the content, as opposed to passively receiving information [14], [15], [27]. Active Learning has proven to be helpful in a general educational context [15], [27], although there is less research on whether Active Learning is equally beneficial in a software context [14]. There has been some research into using Active Learning for teaching object-oriented programming (OOP) [17]–[22]. However, each study used a different methodology to evaluate success, which makes it impossible to determine whether Active Learning is equally beneficial for teaching object-oriented programming compared to other software concepts.

There is also a considerable variety among the examples of Active Learning in an OOP context. For instance, three examples used a flipped classroom for teaching OOP and found that students were more engaged and had a greater sense of self-responsibility [18], [20]. However, research that considered other software concepts could not conclusively prove that grades had improved [23], [24].

A potential weakness of using a flipped classroom is that some students choose not to engage with the prerequisite content, as it is worth no marks [20]. To mitigate this, many examples have used other Active Learning techniques alongside a flipped classroom [17]–[22]. One such technique is project-based learning. This is a well-established technique in software education [14], [17], which allows students to learn experientially and at their own pace [17].

Another technique is Peer Instruction, which may involve the formation of small groups to discuss topics before discussing or presenting information to the whole class [19], [25]. A key benefit is that students can get help from their peers instead of just the instructor. Peer Instruction is often used alongside other learning techniques [19], [25], making it difficult to quantitatively prove whether it improves student performance. However, student feedback has indicated that Peer Instruction is supported by up to 80% of students in some cases [19], but in other cases, students were less satisfied with Peer Instruction compared to the baseline course [25], which the researchers believe was due to factors unrelated to Peer Instruction itself.

Since many of the foregoing techniques are complementary, Blended Learning has emerged, which can entail a combination of Active Learning techniques and traditional learning methods, such as lecture-based content delivered online [25], [26]. Blended Learning reinforces the commonalities of all Active Learning techniques, such as increased student satisfaction [17]–[22] and improved grades [17]–[22]. It also gives educators the flexibility to use class time for more meaningful experiences than purely lecturing [15].

However, a common weakness amongst all examples is that more preparation is required by teachers upfront, especially when transitioning from a lecture-heavy teaching style [20]. It is generally agreed that this effort is warranted [14], [27], given that students overwhelmingly support Active Learning [17]–[22].

It is important to note that these observations are based on limited research in the context of OOP. Most examples of Active Learning have used several techniques in combination, so there is less research on the benefit of using one technique in isolation for teaching OOP.

III. Gamification

Gamification is the use of game design elements in a non-game context [28]. Introducing gamification into educational settings is intended to increase a person's motivation to learn [29]–[36]. This leads to improvements in their absorption of content. This is not to be confused with game-based learning, which uses games as the learning tool itself to achieve learning outcomes [30]. This review will focus on gamification.

The research of gamification elements and its applications in Software Engineering (SE) education is a young but growing research field [28]. Within SE education, gamification elements include a points system, levels, leaderboards and badges to motivate users and reward desired behaviour [37], [42], and instant feedback [33], [35]. OOP is an abstract concept and can be difficult to grasp, especially when this topic is encountered early in a student's university career. Therefore, gamification can be a useful way to teach OOP [30]–[32], [37].

A. Uses

Throughout the literature review on gamification, increasing motivation was often highlighted as the primary use of gamification [29]–[36]. This is followed by the fact that gamification can provide immediate feedback [33]–[35].

1) Increasing Motivation: Jusas et al. [37] introduced various game elements into a course they taught. Students were given experience points (XP), accompanied by a leaderboard. They were provided interactive activites to supplement course material. Finally, students were put in class-wide teams where students from the team with the highest average XP points can avoid skipping the course exam. These led to fewer repeating students and few number of retakes needed to pass the final exam, though there was not a significant increase observed in the mean final exam score. This highlights that introducing game elements to the classroom could push lower-performing students to work harder. This is supported by Souza et al. [29], who observed that leaderboards allowed students to see their performance relative to the class and motivate worse-performing students to work harder.

Aridana and Loekito [32] applied the Marczewski Gamification Framework [38] to design a gamification method to increase student motivation on learning OOP. The design of the method was aimed at students who were motivated by mastery of a topic which includes the use of game mechanics such as levels, challenges and achievements, leaderboards and competitions. The gamification method had five levels based on OOP concepts: (1) Class and Object; (2) Encapsulation; (3)

Inheritance; (4) Polymorphism; (5) Data Abstraction. These rewarded points and badges.

Kučak et al. [30] sought to test whether the use of gamification elements in the teaching process increased students' motivation to practice more in an OOP course of 112 students they taught. The course was divided into two groups, with 91 belonging to a control group and 21 randomly-selected students being allocated to an experimental group that used of gamification elements. Students from both groups had statstically no significant difference in prior knowledge. The experimental group received gamification elements supplementary to the course content. They found that the average exam score of the experimental group was statistically reasonably higher than the control group.

2) Instant Feedback: While Jusas et al. [37] and Souza et al. [29] found that points system, badges and leaderboards improved motivation, Soepriyanto and Kuswandi [31] observed that gamification provided needed feedback. They introduced the game elements of challenges, goals, points, badges and leaderboards into their OOP course. Points were achieved by completing practical challenges, where completion is validated by professors. Only the first five valid entries get points. Points accumulate throughout the semester and translate to students' final grades. Upon completing a topic, students achieve badges. A public leaderboard is provided to motivate and compete against other students. It was found that not all students were motivated to get the highest gamification achievements (badges and top of leaderboard). However, a significant majority used the game elements to get feedback on their learning. This is supported by Alabbasi [35], who found that game elements are capable of fulfilling the partial need for instant feedback in online learning.

Alabassi [35] observed that teachers perceive instant feedback as a feature of online courses designed to motivate students to perform better, and increase their commitment to online learning. In particular, students can use the instant feedback to measure their level of understanding of the course and apply changes wherever necessary, thereby committing to the course. This benefit is observed by Khaleel et al. [34], who include "Real-Time Reactions and Response to Events" as one of the requirements for learning a programming language in their proposed gamification framework. They point to the fact that students can monitor their progress, thereby reducing lack of interest in learning programming languages.

B. Analysis

Essentially, the backbone of gamification is arguably the educational concept of behaviourism. As found by Alabassi [35], both uses of gamification (increasing motivation and instant feedback) work in tandem. Students' gained opportunity to get instant feedback with game elements allow them to track their learning progress and motivate them to stay committed. Since these students stay committed, they are then more likely to further make use of the game elements for their learning to get feedback, thereby motivating them further. Thus, a positive feedback loop has been established, where students' learning performance improve.

While most papers agree that gamification aims to and can motivate most students, the concept also risks inadvertently causing the reverse. As found by Khaleel et al. [34], there are not many gamification frameworks established especially in the field of teaching OOP at this stage. Consequently, poor design and implementation of gamification applied to courses can cause the presence of loopholes, a risk highlighed by Jusas et al. [37]. Such loopholes can allow cheating, where students can inappropriately gain points or badges for themselves. Not only does this introduce the risk of misrepresenting students' learning, but also demotivate honest students.

A few papers have examined the impact of each individual game element. While many papers observe that points, badges and leaderboards increase motivation [29], [30], [37], Alabbasi [35], Soepriyanto and Kuswandi [31] found that these elements, particularly the leaderboard, were could be viewed less favourably instead. Students could get immediate feedback by looking at the leaderboard after performing some task. Alabbasi noted that students could be demotivated when they spend large amounts of effort on to a task only to find that they have barely moved up the leaderboard. The perceived negative feedback can then demotivate students from learning further, because they lack the positive reinforcement they need to encourage them.

The above contradictions motivated into digging further as to why the same game elements may yield different effects in terms of motivation. Further literature review has revealed that motivation can be classified into two forms—intrinsic and extrinsic. Intrinsic motivation is defined as "the doing of an activity for its inherent satisfaction rather than some separable consequence" [40]. A person has high intrinsic motivation if he is driven by some internal factor from within himself that is not related to or influenced by the outside world. On the other hand, extrinsic motivation is caused by one's will "to participate in an activity based on meeting an external goal, garnering praise and approval, winning a competition, or receiving an award or payment" [41]. It is thus safe to assume that extrinsic motivation is influenced by social factors, particularly the need for humans to garner social approval to validate themselves [29].

The difference between intrinsic and extrinsic motivations is important because it has implications on the impact of feedback from various game elements on different personalities. Denden et al. [42] conducted a course that involved eight game elements—points, levels, leaderboard, progress bar, feedback, badges, avatar and chat. At the same time, they measured their students' personalities using the Big Five Personality Traits, which included extraversion, conscientiousness, neuroticism, openness and agreeableness. At the end of the course, students were surveyed on what game elements they prefer. Students' preferences were then matched up against their personality traits. Denden et al. found that higher extraversion traits preferred points, leaderboard and avatar over other game elements, while lower extraversion preferred level. Higher extraversion traits also used the chat functionality more often. Higher conscientiousness preferred points and avatar, while neuroticism had no impact on preference. Denden et al. could not determine whether agreeableness led to certain preferred game elements, because all but one student had high agreeableness.

Extroverts are more likely to have stronger extrinsic motivations, as they tend to be concerned with outer affairs. This is proven by Denden et al. [42], as high extraversion traits prefer game elements that supported ranking learners. While experience points allow directly comparing each other's performance at a particular time instant, the level number makes comparisons harder. It masks the true rank of a player, while also serving as a numeric measurement to determine the amount of progress a person has achieved. This reduces its extrinsic motivating factor relative to experience points, while still serving intrinsic motivations. This may explain why lower extraversion traits prefer levels. This finding is crucial as it highlights the need to study the demographic of the target audience when gamifying an existing learning process.

While the literature on gamification provides valuable insights towards the design and implementation of a software-based learning tool, it is also true that gamification is still at its infancy stages. This means that much of the literature on the topic has not been well-validated outside the courses or universities that their authors teach or belong to. Most of the authors of the literature reviewed applied gamification concepts on their own courses, and gathered empirical data from their students. The number of students involved per paper reviewed were usually less than 100, and tied to a specific subset of courses from a university faculty. Their methods have not been tested outside their universities. Consequently, more empirical evidence is needed to verify the claim that gamification causes improved learning outcomes. This is especially true with regards to learning OOP, on which the literature is lacking. On the other hand, this lack of literature on applying gamification for teaching OOP presents an opportunity to advance understanding in that area and make valuable academic contributions at this stage.

IV. Cognitive Load

Cognitive Load Theory (CLT) posits that overloading the human working memory can impede students' ability to learn [48]. The types of cognitive load described in this theory include intrinsic, extrinsic, and germane. Intrinsic cognitive load refers to the complexity of the subject matter in relation to the learner's prior knowledge. Extrinsic cognitive load related to the way in which learning material is presented to the learner. Germane cognitive load refers to the effort required to learn. Extraneous cognitive load is seen to induce unnecessary strain on working memory resources, hence why many strategies based on CLT focus on reducing extraneous cognitive load to improve learning effectiveness.

CLT in computer science education is a broad and well-researched domain; studies range from effectiveness and efficiency of individual techniques such as visualisation [56], [57], [59] and Parsons problems [43]–[46] to entire pedagogical frameworks that can be used to design computer programming courses [47], [49]–[55].

A. Uses

1) Visualization: Visualization is a way of applying cognitive load theory in teaching programming. The main methods for applying visualization that is analyzed in this section are: Visual output, block programming, and hybrid interface (visual and textual).

Bakar et al. [57] discussed the development and validity of a Java learning module called VJava for teaching coding concepts through a visual output approach. The VJava module is developed intending to teach coding at a higher educational level with the consideration of teaching Object-First while also incorporating visual output elements to better cater this course to Gen Z students. This module applied cognitive load theory through the use of Graphical and Animated Libraries and produce a visual output, which is supposed to help students understand concepts such as "sequence, repetition and selection structure". The graphical outputs should also help students understand program flow and make it easier to identify errors. The modules are evaluated by 3 Programming experts each with around 20 years of experience in the field of education and programming, along with one Instructional Design expert with 11 years of experience in their own field. The evaluation is done through questionnaires filled out by the 4 experts, and a summary of the results are that: the visual outputs make concepts easier to understand and is a "simple, interesting and appropriate approach for 21st-century learning".

Yousoof, Sapiyan and Kamaluddin [59] proposed a framework for reducing cognitive load on learning programming language through the use of a concept map during the run time of the program. This is quite similar to Bakar's VJava idea, which also uses visual output in attempts to reduce cognitive load and improve learning capabilities. The difference being the way the output is visualized. Yousoof et al. proposed the use of concept maps which would be able to visually demonstrate run time on more complex programs, and independent of implementing a library function. Although this framework is only a proposal and no implementation or evaluation has been done on its effectiveness.

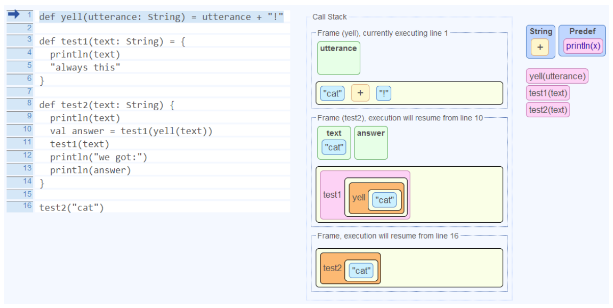
Unal and Topu [56] evaluated the effectiveness of Blockly which is a hybrid coding interface (visual as well as textual) in terms of the students' computer programming anxiety, cognitive load level, and achievement. Textual interface for coding can have a high ICL while only learning through visual interface like visual code blocks cannot effectively teach students a real programming language. Their research implemented a 10 week learning course that had a 44 people experimental group (EG) using Blockly (hybrid interface), and a 46 people control group (CG) using Python editor (non-hybrid interface). The research found that hybrid interface had no significant effect on student's programming anxiety, while the CL for the EG were lower especially starting from week

5 of the study. The programming scores of the two groups showed that the EG had higher achievements through the course than the CG, which suggests a better learning outcome through the use of hybrid interface. This study concluded that the use of hybrid interface is a effective way of teaching a real programming language while reducing the cognitive load on the students and improving the learning effectiveness.

2) Course Design: Elliott, Stephen N., et al. [47] focused on the inclusion of students with disabilities in standardised testing. They do this by examining the application of CLT strategies across multiple studies and evaluating them. The first strategy is proposed by [49] and suggests offloading some content to the auditory channel producing a modality effect (Learning is enhanced when textual information is presented in an auditory format rather than entirely in the visual format [50]). The strategy has a median effect size (ES) of 1.17 across six studies. The second strategy is to provide pre-training to facilitate the key ideas into long-term memory before exposing novel materials (ES = 1.00; 3 studies). The third strategy address cognitive overload from extraneous material by weeding (removing extraneous material; ES=0.90; 5 studies) or signalling (providing cues for guidance; ES = 0.74; 1 study). The fourth indicates reducing redundant information (ES = 0.69; 3 studies). The fifth and final aims to minimize the need for representational holding by synchronizing information (e.g. by showing animation and text simultaneously; ES = 1.80; 3 studies). Cognitive load is reduced because the learner no longer has to store visual animation in the working memory before linking it with the corresponding text afterwards. However, the work has limited mentions of the effectiveness of the strategies for testing students with disabilities. Nonetheless, the CLT guidelines and their applications to testing are transferable to OOP education and should be evaluated and considered when designing the educational application.

Bharathan, Rasiah, et al. [51], use the LAP Mentor VR laparoscopic simulator for teaching Ectopic Pregnancy to novice surgeons. The experiment was conducted across 25 trainees and nine senior gynaecologists. The study is one of the first to show a significant correlation between cognitive load and dexterity parameters (e.g. when one becomes more dexterous at the task, their perceived cognitive load decreases). Hence, a reduction in cognitive load significantly improved the learning curves. The results accentuate the role of cognitive load in aiding or prohibiting learning.

Sorva, Juha, and Otto Seppälä [52] illustrate the application of three CLT-based frameworks on CS1 (an introductory programming course). Motivate-isolate-practice-integrate combines project-based learning by offering a complex project that is challenging but exciting. Cognitive load is reduced by breaking down learning activities into isolated tasks before getting the students to apply them. Head straight for objects suggests introducing OOP concepts early in the course. OOP theories are often interdependent and difficult, hence will cause cognitive overload if taught simultaneously. Finally, the principle of explicit program dynamics suggests exposing the runtime dynamics of programs to students (e.g. visualizing the call stack shown in Figure 1). Since the introduction of this design to the course, there has been a high pass rate (89.1% among beginner programmers) and feedback has reached an all-time high (4.31 out of 5). However, there is a lack of concrete empirical evidence.



```
 1  def yell(utterance: String) = utterance + "!"
 2
 3  def test1(text: String) = {
 4    println(text)
 5    "always this"
 6  }
 7
 8  def test2(text: String) {
 9    println(text)
10    val answer = test1(yell(text))
11    test1(text)
12    println("we got:")
13    println(answer)
14  }
15
16  test2("cat")
```

Call Stack
Frame (yell), currently executing line 1
utterance
"cat"
+
"!"
Frame (test2), execution will resume from line 10
text
"cat"
answer
test1
yell
"cat"
Frame, execution will resume from line 16
test2
"cat"
String
+
Predef
println(x)
yell(utterance)
test1(text)
test2(text)

Fig. 1. A “low”-level program animation; adapted from Sorva et al. [52]

Similarly, Caspersen, Michael E., and Jens Bennedsen [53] present the instructional design of an OOP course with regard to CLT. The authors advocate the principle of Consume-before-produce, in other words allowing students to call methods and experiment with them before modifying and creating them. After students are confident with methods, abstraction is reduced and the same procedure is repeated with classes. This prevents cognitive overload because producing often involves extensive knowledge about class internals that is difficult to acquire without understanding interactions. Furthermore, a pattern-based approach should be followed because patterns capture programming knowledge in chunks which reduces cognitive load and aids schema acquisition (encoding and decoding between long-term and working memory). The instructional design was used for four years with over 400 students per year however also suffers from missing formal evaluation.

Salleh et al. [54] apply CLT by suggesting a scaffolding model for introductory programming. Two main approaches are examined; Task-example provides problem solutions and every step is explicitly pointed out. The burden of searching for relevant information is alleviated reducing the load on their working memory. In goal-free, problem statements do not provide the objective or tasks. Cognitive load is reduced because the students can focus on the initial state instead of means-ends. Both approaches should begin by outlining the question statement and support with scaffolding (worked-examples for task-example, idea boosters questions (e.g. inputs/outputs to the problem) for goal-free).

Chang et al. [55] propose an object-focused model, programmers get the sense that objects on the screen are the objects in the program, and hence think about working with objects directly as opposed to the environment. The key difference between view-focused and object-focused is that in object-focused models, objects will change their representation after manipulation whereas a view-focused model only explicitly shows the programming activities. The motivation is to reduce the distance between the programmer's mental model of objects and the environment's representation of them. However, there is a lack of concrete evidence supporting this approach.

3) Parsons problems: Parsons problems are a popular strategy used to teach beginner programmers that typically involve assembling a set of programming statements in the correct order. These problems can be extended to include incorrect statements known as distractors, requiring the correct indentation of statements (two dimensional), or blanking out components that learners must fill in (blank-variable or faded). This method aligns with the completion problem effect seen in cognitive load theory which suggests that providing partial solutions reduces the size of the problem space, and thus reduces extraneous cognitive load for novices that have yet to acquire the necessary knowledge to solve such exercises in their long-term memory.

Zhi et al. [43] designed and integrated Parsons problems into block-programming interface Snap! to investigate their effectiveness. The experiment involved comparing student performance and efficiency in a non-majors CS0 course across 6 semesters, consisting of 3 lab assignments, 2 homework assignments, and a project. The experimental group were given lab assignments as Parsons problems, while previous semesters had completed the same assignments as conventional coding problems. The homework assignments remained as conventional coding problems extending each of the labs and acted as an immediate post-test. The project acted as a delayed post-test. The authors found that the Parsons problems-based labs saved students time (between 43-65%) with no negative impact on performance compared to students who wrote code from scratch. There was also no performance difference between semesters for homework assignments and the final project.

Harms et al. [44] used Parsons problems with distractors to determine the impact of distractors on learning effectiveness. In the first phase of their two-phase study, two groups of middle-school students were each given a set of Parsons problems, one with sub-optimal path distractors and one without, to learn three programming concepts independently. In the second phase, students' learning was evaluated by giving them three more tasks. Cognitive load surveys were conducted after each task. The results indicated that students with distractors spent 14% more time on their learning tasks while completing 26% less tasks than students with no distractors (i.e. they were more likely to give up), meanwhile when evaluated in the second phase there were no differences between the two groups of any measure. Consequently, the authors found that distractors offer little benefit and negatively impact learning efficiency.

Ericson, Foley and Rick [45] employed both adaptive and non-adaptive two-dimensional Parsons problems with distractors to study their compared learning effectiveness and efficiency between each other and equivalent write code problems. Adaptive Parsons problems adopt scaffolding principles such that in the case a student struggles to solve a given Parsons problem, current problem can be made easier (e.g. remove distractors) to guide the learner to the correct solution by pressing a "Help Me" button. This is known as intra-problem adaptation. Alternatively, adaptation can occur on the next Parsons problem, making it harder or easier based on the number of attempts it took to complete the last problem (inter-problem adaptation). The study involved bringing undergraduate students in an introductory computer science course into a 2.5 hour session to perform a pre-test, one of four practice problems with worked examples, and a post-test, and another hour-long session a week later for delayed post-test. They found that both adaptive and non-adaptive Parsons problems were more efficient than write code problems, however effectiveness could not be definitively determined.

Caynes and Ericson [46] used similar forms of adaptive Parsons problems with distractors in a within-subjects and think-aloud observation study with undergraduate students. The within-subjects study had students complete two sets (A and B) of practice problems of varying difficulty at the end of their course. Set A alternated the problem type with the first problem as a Parsons problem, the second problem as a write-code problem etc. Set B presented the corresponding problems in A in the opposite form. The within-subjects study found that the median completion time and mean cognitive load rating for the adaptive Parsons problems was less than the equivalent write-code problems. The think-aloud observation study revealed valuable student insights about the use of Parsons problems such as despite them being easier, most of the interviewed students preferred write-code problems. Two students, who had prior programming experience, expressed strong dislike for Parsons problems and felt they were a waste of time. On the other hand, a few students preferred Parsons problems for the support they provided.

B. Analysis

Most papers examining the use of Parson's problem show improved efficiency compared to their write-code problem counterpart. The results include less time spent while having no adverse effects on performance [43] and reduced cognitive load rating [46]. The improvement is likely because Parson's problem allows students to focus on structure compared to syntax, hence saving time from common mistakes such as missed brackets and colons. This abstraction is a key benefit of using Parson's problem to reduce cognitive load.

In the paper by Harms et al. [44], they mention that distractors increase cognitive load without providing much benefit. However, Ericson, Foley and Rick argue that some difficulty is needed for learning gains, which distractors can be used to provide, to be made. The resulting cognitive overload could be mitigated using adaptive Parson's problems, in which distractors can be removed if the problem proves too difficult for the learner [45]. These results indicate the uncertainty of extending Parson's problems with distractors and hence they should be implemented with care when considering applying it to educational OOP.

On the other hand, students' attitudes to Parson's problem varied significantly. There are many students who prefer write-code problems over Parson's problems and even showed despise for Parsons's problem despite the overall lower cognitive load rating [46]. On the other hand, there are also students who liked Parson's problems. In other papers, there is no mention of the student's opinions on whether Parson's problem is helpful in their learning. The negative opinions may be because Parson's problems are limited to one solution compared to the freedom of write-code solutions, hence are not valuable for learners with programming experience (expertise reversal effect [58]). Therefore, a gap that needs to be acknowledged is perhaps a user study on whether students actually found Parson's problem to be an effective means of applying CLT to educational OOP.

Furthermore, most results focused on time reduction while none of them mentioned a concrete improvement in academic results. In fact, many showed indeterminate or insignificant results [43] [45]. Hence, there remains the question of whether Parson's problem is a means to improve learning outcomes or simply simplify the task at hand.

A key difference in the papers examined regarding course design is the degree of supporting empirical evidence. Although [47] argues the effectiveness of CLT strategies through median effect size, it does not mention whether the same results can be derived when applied to testing for students with disabilities. Other evaluation metrics include pass rate and feedback [52], and the correlation between cognitive load and learning curves [51]. Other works simply proposed strategies and principles without explicitly showing their results in practice. Hence, the lack of a unified evaluation metric for CLT makes it difficult to compare the studies. This is likely because of the nature of cognitive burden being a qualitative metric and difficult for one to express numerically. Therefore, during the design of an educational OOP application, it is indispensable to consider the users holistically in terms of which strategies will likely be most effective.

A recurring suggestion is the use of scaffolding to alleviate cognitive overload. Elliott et al. [47] mentions techniques such as cues, removing extraneous materials, and pre-training to aid learning. Similarly, Sorva et al. [52] argue teaching OOP concepts initially without using explicit OOP jargon reduces cognitive burden allowing students to focus on core relationships and concepts rather than memorising terms. Likewise, Salleh et al. [54] demonstrates a scaffolding model for introductory programming courses however its effectiveness is yet to be evaluated. Therefore, scaffolding could be a promising means to reduce cognitive overload in teaching complex topics such as OOP by guiding and introducing students to OOP gradually compared to exposing all information at once.

Another key idea is the decomposition of learning outcomes into smaller digestible chunks. Although [52] proposes using large but captivating projects to stimulate students' interests, the actual tasks should be learnt in isolation before being applied. Likewise, the principle of Consume-before-produce in [53] advocates experimentation before creation. This decomposition is helpful because the working memory has limited capacity and hence is better suited to learning concepts in isolation before creating links to avoid overwhelming the student.

Furthermore, most papers echo the theme of practice. In [52], the authors indicate that practice is indispensable to learning the isolated concepts in OOP before integrating them back into the overarching project. In [53], the authors suggest students should practice understanding functions by practising through interactions before creating them themselves. Hence, practice is a key component in applying CLT to facilitate the transfer of information from working memory into long-term memory.

Visualisation in reducing cognitive load can be put into visualising the interface and visualising the output. Block-based programming is mentioned in many of the cognitive load related articles. Yet, the use of block-based coding/ visualising the interface seems complicated to apply when relating to OOP, which is already a more advanced concept. Visualisation of the interface is mentioned in [56], where a hybrid interface is used when trying to incorporate more complicated learning into the simple block-based interface style. By having a textual interface alongside the block-based interface, users are able to gain knowledge of the actual code while also being able to code with blocks. This approach was able to reduce the learners' cognitive load, while not hindering their learning effectiveness, proving this method's effectiveness.

The use of visualising output is more prevalent when introducing slightly more advanced concepts such as OOP. In [57], the VJava module is designed with cognitive load and constructism in mind, and these educational concepts are applied to the module through effective course design, and visualisation of the output while also keeping the Object-first teaching approach. This realises the possibility of teaching OOP concepts through effective course design while also lessening the learners' cognitive load through visualisation. Visualising the output is also mentioned in [59] where the output is, instead of a line-drawing-based visualisation, is a concept map of the run time that is able to produce more complex outputs.

When discussing visualisation in CLT, it is often seen that visualisation is used in parallel to other techniques and often not used along. In [57], visualisation is used along side effective course design in order to apply CLT., and in [59], visualisation is used along with part-code to apply CLT. This shows that it is most effective in reducing CL when multiple methodologies are applied alongside visualisation.

References

[1] V. Lian, E. Varoy, and N. Giacaman, "Learning Object-Oriented Programming Concepts Through Visual Analogies," *IEEE Transactions on Learning Technologies*, pp. 1–1, 2022, doi: https://doi.org/10.1109/tlt.2022.3154805.

[2] J. J. Dupin and S. Johsua, "Analogies and 'Modeling Analogies' in Teaching: Some Examples in Basic Electricity," *Science Education*, vol. 73, no. 2, pp. 207–24, 1989, Accessed: May 06, 2023. [Online]. Available: https://eric.ed.gov/?id=EJ392806

[3] P. R. Simons, "Instructing with analogies.," *Journal of Educational Psychology*, vol. 76, no. 3, pp. 513–527, 1984, doi: https://doi.org/10.1037/0022-0663.76.3.513.

[4] M. Forišek and M. Steinová, “Metaphors and Analogies for Teaching Algorithms *.” Accessed: May 06, 2023. [Online]. Available: https://people.ksp.sk/ misof/publications/2012metaphors.pdf

[5] P. Thagard, “Analogy, explanation, and education,” *Journal of Research in Science Teaching*, vol. 29, no. 6, pp. 537–544, Aug. 1992, doi: https://doi.org/10.1002/tea.3660290603.

[6] K. Thramboulidis, “A Constructivism-Based Approach to Teach Object-Oriented Programming,” *Journal of Informatics Education and Research*, vol. 5, no. 1, Jan. 2003.

[7] H. Zhu and M. Zhou, “Methodology first and language second,” *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, Oct. 2003, doi: https://doi.org/10.1145/949344.949389.

[8] T. Tanielu, R. Akau’ola, E. Varoy, and N. Giacaman, “Combining Analogies and Virtual Reality for Active and Visual Object-Oriented Programming,” *Proceedings of the ACM Conference on Global Computing Education*, May 2019, doi: https://doi.org/10.1145/3300115.3309513.

[9] R. Kempf and M. Stelzner, “Teaching object-oriented programming with the KEE system,” *Conference proceedings on Object-oriented programming systems, languages and applications* - OOPSLA ’87, 1987, doi: https://doi.org/10.1145/38765.38809.

[10] H. James. Nelson, G. Irwin, and D. E. Monarchi, “Journeys up the mountain: Different paths to learning object-oriented programming,” *Accounting, Management and Information Technologies*, vol. 7, no. 2, pp. 53–85, Jan. 1997, doi: https://doi.org/10.1016/s0959-8022(96)00024-0.

[11] A. Gaspar and S. Langevin-Gaspar, “Active learning in introductory programming courses through Student-led ’live coding’ and test-driven pair programming,” *International Conference on Education and Information Systems, Technologies and Applications*, Orlando, FL, USA, 2007, vol. 1.

[12] R. M. Felder and R. Brent, “Active Learning: An Introduction,” in *ASQ Higher Education Brief*, 2009, vol. 2, no. 4.

[13] S. Cooperstein and E. Kocevar-Weidinger, “Beyond active learning: A constructivist approach to learning,” in *Reference Services Review*, vol. 32, no. 2, pp. 141-148, Jun. 2004. doi: 10.1108/00907320410537658.

[14] J. H. Berssanette and A. C. de Francisco, “Active learning in the context of the teaching/learning of computer programming: A systematic review,” in *Journal of Information Technology Education: Research*, 2001, vol. 20, pp. 201-220, doi: 10.28945/4767.

[15] Z. Zayapragassarazan and K. Santosh, “Active Learning Methods,” in *NTTC Bulletin*, 2012, vol. 19, no. 1, pp. 3-5.

[16] J. Towell, “An Active Learning Environment for Teaching Object-Oriented Concepts in Business Information Systems Curricula,” in *Journal of Information Systems Education*, 2000, vol. 11, pp. 147-150.

[17] Y. -L. Chen, C. -Y. Chiang, Y. -P. Huang and S. -M. Yuan, “A Project-Based Curriculum for Teaching C++ Object-Oriented Programming,” *2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*, Fukuoka, Japan, 2012, pp. 667-672, doi: 10.1109/UIC-ATC.2012.94.

[18] A. Herala, E. Vanhala and U. Nikula, “Object-oriented programming course revisited,” in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, 2015, doi: 10.1145/2828959.2828974.

[19] E. Kaila, E. Kurvinen, E. Lokkila, and M.-J. Laakso, “Redesigning an Object-Oriented Programming Course,” *ACM Transactions on Computing Education*, vol. 16, no. 4, 2016, doi: 10.1145/2906362.

[20] P. Pugsee, “Effects of using flipped classroom learning in object-oriented analysis and design course,” *2017 10th International Conference on Ubi-media Computing and Workshops (Ubi-Media)*, Pattaya, Thailand, 2017, pp. 1-6, doi: 10.1109/UMEDIA.2017.8074130.

[21] S. Yang, H. Park and H. Choi, “Impact of Active Learning on Object-Oriented Programming Instruction : Transforming from 3D to Text-based coding,” *2021 IEEE Integrated STEM Education Conference (ISEC)*, Princeton, NJ, USA, 2021, pp. 252-255, doi: 10.1109/ISEC52395.2021.9763964.

[22] R. Cheung, “A web-based learning environment for object-oriented programming”, *Int. J. of Information and Operations Management Education*, vol. 1, pp. 140–157, 01 2006.

[23] E. F. Gehringer and B. W. Peddycord, “The Inverted-Lecture Model: A Case Study in Computer Architecture”, in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, Denver, Colorado, USA, 2013, pp. 489–494.

[24] G. C. Gannod, J. E. Burge, and M. T. Helmick, “Using the Inverted Classroom to Teach Software Engineering”, in *Proceedings of the 30th International Conference on Software Engineering*, Leipzig, Germany, 2008, pp. 777–786.

[25] D. Gasevic, G. Siemens, and S. Dawson, “Preparing for the digital university: a review of the history and current state of distance, blended, and online learning”. 01 2015.

[26] P. Fadde and P. Vu, “Blended online learning: Misconceptions, benefits, and challenges”, pp. 33–48, 01 2014.

[27] R. M. Felder and R. Brent, “Teaching and learning STEM : a practical guide”. San Francisco, Ca: Jossey-Bass, A Wiley Brand, 2016.

[28] M. M. Alhammad and A. M. Moreno, “Gamification in software engineering education: A systematic mapping,” *The Journal of Systems and Software*, vol. 141, pp. 131-150, 2018. Available: https://dx.doi.org/10.1016/j.jss.2018.03.065. DOI: 10.1016/j.jss.2018.03.065.

[29] M. R. d. A. Souza et al, “Gamification in Software Engineering Education: An Empirical Study,” *The Institute of Electrical and Electronics Engineers, Inc. (IEEE) Conference Proceedings*, pp. 276, 2017. Available: https://search.proquest.com/docview/1974441705.

[30] D. Kučak, D. Bele and D. Pašić, “Climbing up the Leaderboard: An Empirical Study of Improving Student Outcome by Applying Gamification Principles to an Object-Oriented Programming Course on a University Level,” *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, Opatija, Croatia, 2021, pp. 527-531, doi: 10.23919/MIPRO52101.2021.9596709.

[31] Y. Soepriyanto and D. Kuswandi, “Gamification Activities for Learning Visual Object-Oriented Programming,” *2021 7th International Conference on Education and Technology (ICET)*, Malang, Indonesia, 2021, pp. 209-213, doi: 10.1109/ICET53279.2021.9575076.

[32] D. P. Y. Ardiana and L. H. Loekito, “Gamification design to improve student motivation on learning object-oriented programming,” *Journal of Physics*. Conference Series, vol. 1516, (1), pp. 12041, 2020. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1516/1/012041. DOI: 10.1088/1742-6596/1516/1/012041.

[33] M. Kosa et al, “Software Engineering Education and Games: A Systematic Literature Review,” Dec 1, 2016.

[34] F. L. Khaleel et al, “Gamification-based learning framework for a programming course,” *The Institute of Electrical and Electronics Engineers, Inc. (IEEE) Conference Proceedings*, 2017. Available: https://search.proquest.com/docview/2013258270.

[35] D. Alabbasi, “Exploring teachers perspectives towards using gamification techniques in online learning,” *TOJET the Turkish Online Journal of Educational Technology*, vol. 17, (2), pp. 34-45, 2018.

[36] M. Thongmak, “Creating gameful experience in the object-oriented programming classroom: A case study,” *The Online Journal of Applied Knowledge Management*, vol. 6, (1), pp. 30-53, 2018. DOI: 10.36965/OJAKM.2018.6(1)30-53.

[37] V. Jusas, D. Barisas and M. Jančiukas, “Game Elements towards More Sustainable Learning in Object-Oriented Programming Course,” *Sustainability* (Basel, Switzerland), vol. 14, (4), pp. 2325, 2022. Available: https://search.proquest.com/docview/2633334510. DOI: 10.3390/su14042325.

[38] A. Marczewski, “Gamification: a simple introduction”, 2013.

[39] G. Ivanova, V. Kozov and P. Zlatarov, “Gamification in Software Engineering Education,” *Mipro*, pp. 1445-1450, 2019. Available: https://ieeexplore.ieee.org/document/8757200. DOI: 10.23919/MIPRO.2019.8757200.

[40] R. M. Ryan and E. L. Deci, “Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions,” *Contemporary Educational Psychology*, vol. 25, (1), pp. 54-67, 2000. Available: https://dx.doi.org/10.1006/ceps.1999.1020. DOI: 10.1006/ceps.1999.1020.

[41] V. S. J. Paula Thomson, *Creativity and the Performing Artist*. San Diego: Elsevier Science, 2016.

[42] M. Denden et al, “Does Personality Affect Students’ Perceived Preferences for Game Elements in Gamified Learning Environments?” *The Institute of Electrical and Electronics Engineers, Inc. (IEEE) Conference Proceedings*, pp. 111, 2018. Available: https://search.proquest.com/docview/2088108059.

[43] R. Zhi, M. Chi, T. Barnes, and T. W. Price, “Evaluating the effectiveness of Parsons problems for block-based programming,” *Proceedings of the 2019 ACM Conference on International Computing Education Research*, 2019.

[44] K. J. Harms, J. Chen, and C. L. Kelleher, “Distractors in Parsons problems decrease learning efficiency for young novice programmers,” *Proceedings of the 2016 ACM Conference on International Computing Education Research*, 2016.

[45] B. J. Ericson, J. D. Foley, and J. Rick, “Evaluating the efficiency and effectiveness of adaptive Parsons Problems,” *Proceedings of the 2018 ACM Conference on International Computing Education Research*, 2018.

[46] C. C. Haynes and B. J. Ericson, “Problem-solving efficiency and cognitive load for adaptive Parsons problems vs. writing the equivalent code,” *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021.

[47] Elliott, Stephen N., et al. “Cognitive load theory: Instruction-based research with applications for designing tests.” *Proceedings of the National Association of School Psychologists’ Annual Convention*, Boston, MA, February. Vol. 24. 2009.

[48] J. Sweller, J. J. van Merriënboer, and F. Paas, “Cognitive Architecture and Instructional Design: 20 years later,” *Educational Psychology Review*, vol. 31, no. 2, pp. 261–292, 2019.

[49] R. E. Mayer and R. Moreno, “Nine Ways to Reduce Cognitive Load in Multimedia Learning,” *Educational Psychologist*, vol. 38, (1), pp. 43-52, 2003. DOI: 10.1207/S15326985EP3801_6.

[50] P. Ginns, “Meta-analysis of the modality effect,” *Learning and Instruction*, vol. 15, (4), pp. 313-331, 2005. Available: https://dx.doi.org/10.1016/j.learninstruc.2005.07.001. DOI: 10.1016/j.learninstruc.2005.07.001.

[51] R. Bharathan et al, “Psychomotor skills and cognitive load training on a virtual reality laparoscopic simulator for tubal surgery is effective,” *European Journal of Obstetrics & Gynecology and Reproductive Biology*, vol. 169, (2), pp. 347-352, 2013. DOI: 10.1016/j.ejogrb.2013.03.017.

[52] Sorva, Juha, and Otto Seppälä. “based design of the first weeks of CS1.” *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*, 2014.

[53] Caspersen, Michael E., and Jens Bennedsen. “Instructional design of a programming course: a learning theoretic approach.” *Proceedings of the third international workshop on Computing education research*, 2007.

[54] Salleh, Syahanim Mohd, Zarina Shukur, and Hairulliza Mohamad Judi. “Scaffolding model for efficient programming learning based on cognitive load theory.” *International Journal of Pure and Applied Mathematics*, 2018.

[55] Chang, Bay-Wei, David Ungar, and Randal B. Smith. “Getting close to objects: Object-focused programming environments.” *Prentice-Hall*, 1995.

[56] A. Unal and F. B. Topu, “Effects of teaching a computer programming language via hybrid interface on anxiety, cognitive load level and achievement of high school students,” *Education and Information Technologies*, Apr. 2021, doi: https://doi.org/10.1007/s10639-021-10536-w.

[57] M. Abu Bakar, M. Mukhtar, U. Kebangsaan, M. Bangi, and M. Khalid, “The Development of a Visual Output Approach for Programming via the Application of Cognitive Load Theory and Constructivism,” *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 11, 2019.

[58] Kalyuga, Slava. “The expertise reversal effect. Managing cognitive load in adaptive multimedia learning.” *IGI Global*, 2009. 58-80.

[59] M. Yousoof, M. Sapiyan and K. Kamaluddin, “Reducing Cognitive Load In Learning Computer Programming,” *International Journal of Information, Control and Computer Sciences*, Dec 20, 2007.