The Instructional Design of Worked Examples to Promote Computational Thinking Skills in Well-structured Programming Problems: An integrative Review

Ghadah Fayez Almutairy

Dissertation submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
In
Curriculum and Instruction

Kenneth R. Potter, Chair
Barbara B. Lockee
Alicia L. Johnson
Mark A. Bond

November 10, 2022

Blacksburg, VA

Keywords: worked examples, computational thinking, programming, instructional design.

The Instructional Design of Worked Examples to Promote Computational Thinking Skills in Well-structured Programming Problems: An integrative Review

Ghadah Fayez Almutairy

ABSTRACT

Educators in the current era face more pressure to meet learners' growing digital age learning needs, which may require fostering more vital computational thinking skills. To ensure the desired learning outcomes are attained, it is critical to know how to provide the appropriate type of guidance and assistance. The findings of this research may be significant to computer science instructors and instructional designers interested in fostering computational thinking skills and improving programming skills by designing effective worked examples. Following the integrative review methodology, the study examined the current literature on worked examples in a programming setting to determine the compelling designs of worked examples. In addition, this study examined the most employed instructional design principles in developing effective worked examples and explored factors and circumstances that may have impacted the effectiveness of those designs. This study's findings indicated several successful designs of worked examples to promote computational thinking skills in programming problems.

The instructional design of Worked Examples to Promote Computational Thinking Skills in Well-structured Programming Problems: An integrative Review

Ghadah Fayez Almutairy

GENERAL AUDIENCE ABSTRACT

Educators focus on fulfilling learners' expanding digital age learning requirements, which may require developing more critical computational thinking skills. It is vital to understand how to give appropriate guidance and support to achieve the intended learning results in programming courses. The outcomes of this study may be helpful to computer science educators and instructional designers who aim to support learners in gaining more advanced computational thinking skills. The study used the integrative review approach to investigate the current literature on worked examples in a programming context to discover the compelling designs of worked examples. The study provides information about the factors that may affect the design in addition to discuss several instructional design principles in regard to worked examples. The outcomes of this study showed numerous successful designs of worked examples that are helping in enhancing computational thinking skills in programming tasks.

DEDICATION

I dedicate this work to my dear parents, and to my one and only brother for their unconditional love and support.

I wouldn’t have made it without you.

AKNOWLEDGEMENTS

The completion of this dissertation marks the end of a demanding and fulfilling Ph.D. program. This page is the most difficult for me to write once I have completed my research. There aren't enough words to thank everyone who helped and encouraged me.

First, I would like to thank the first person that I was lucky to meet and to talk to in Virginia Tech, my advisor, Dr. Kenneth Potter. I would like to thank Dr. Potter specifically for his valuable guidance and patience in supporting me through my Ph.D. journey. I consider myself extremely fortunate to have his assistance and advice. His enthusiasm and generous guidance have motivated me to improve myself for the best. I cannot thank him enough for helping me achieve my academic goals. It was a great honor to be advised by him.

Thanks also go to the members of my valuable committee, Dr. Lockee, Dr. Johnson, and Dr. Bond. I truly appreciate their invaluable recommendations and assistance throughout this process. I appreciate your support in helping me to form my knowledge, your time away from your heavy workload to participate in my journey, and your valuable input and interest in my study.

My best friend Rebecca Clark-Stallkamp deserves special recognition for always being ready to provide support, inspiration, and advice while seeking her academic aims.

I would also like to thank my amazing family for their love and support, as well as their wholehearted support in all of my endeavors. Thank you for always believing in me.

Finally, I must recognize Imam Abdulrahman bin Faisal University, who provided me with a scholarship that made this research possible; I cannot wait to join the faculty members and share the valuable experience I gained from Virginia Tech.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter One

Introduction and Need for the Study

Introduction and background

Over the previous few decades, the world witnessed tremendous economic and technological developments from the twentieth to the twenty-first century. In this extraordinarily challenging, diverse, and developed time, it could be assumed that the new generations who grow up in today's educational system will live their adult lives experiencing a multitasked and technologically advanced world. Equally important, various advanced skills need to be developed to keep up with the world's rapid economic development. To stay competitive with global and economic challenges, more consideration of 21st-century skills is needed to empower learners with the ability to navigate the dynamics that are involved in today's time (Chu et al., 2021). Also, there is a need for creative thinking and problem-solving skills to handle complex technologies such as "artificial intelligence, robotics, the Internet of Things, and analytics" highlighted by the rapid shift in skills needed across all professions (Kale et al., 2018, p.574).

According to the McKinsey Global Institute's research, computational thinking skills, programming literacy, and data analysis are examples of the digital foundational skills that individuals should gain for their future employment (Dondi et al., 2021). The results revealed a significant demand for technological, social, emotional, and higher cognitive skills, which will become increasingly relevant in the future (Dondi et al., 2021). With regard to the significance of computational thinking skills, many efforts have been made to enhance those skills while learning. For instance, many "initiatives by the U.S. Government and Department of Education,

such as Digital Promise, MakerEd, and Nation of Makers" are recognizing the importance of computational thinking skills (Kale et al., 2018, p.575).

Also, "President Obama proposed that every student from kindergarten through high school should have access to computer science and mathematics courses to be equipped with computational thinking skills" (D'Alba & Huett, 2017, p.3). Moreover, the Computer Science Teachers Association (CSTA) and the International Society for Technology in Education (ISTE) worked to discover more information about computational thinking concepts and methods, and how they have been applied in several fields (CSTA & ISTE, 2011; Kale et al., 2018).

Apart from this, the interest in computational thinking (CT) in K-12 schools increased, particularly, the acquisition of higher order thinking skills and digital skills (Angeli & Giannakos, 2020). Researchers have advocated for integrating CT into K-12 curricula because it enables the next generation to improve problem-solving skills (Kong, 2019).

Accordingly, several researchers and instructors are interested on how to embed CT within the curricula and how to promote CT skills (Kong et al., 2018). From an educational standpoint, CT is promoted through programming courses as a tool since it can be induced by programming skills and knowledge, and learners manifest their knowledge about CT skills by employing programming tools (Araujo et al., 2017; Kong et al., 2018).

Programming includes a variety of cognitive abilities that assists the learner in developing crucial skills like critical thinking skills, problem-solving skills and creativity (Durak et al., 2019). All previously mentioned skills are embedded in the computing courses, and it may be accomplished by including them in programming activities (Durak et al., 2019). Programming instruction showcases and promotes CT skills through providing interaction with screen-based

and physical artifacts (Goode et al., 2012; Kafai & Burke, 2013). However, developing effective programming instructions with providing the appropriate support is not an easy task for educators (Santos et al., 2020). Jones and Davis (2008) claimed that "effective teaching begins with effective planning of instruction" (p.24).

Planning for effective instructions required following an organized system based on theoretical foundation. Instructional design (ID) is a systematic approach for designing and developing instructions following learning theories and principles to produce sufficient instructions (Dick et al., 2001). Therefore, it is necessary to consider one of the fundamental beliefs in the field of ID - in order to develop instruction effectively we should recognize that different learning outcomes might require different instructional strategies (Jonassen, 2000; Gagné, 1985). To simplify, different problematic situations could require different abilities to find solutions using different reasoning skills which play a huge role in generating effective solutions. Therefore, exploring ways that educators and instructional designers can successfully embed and promote CT with instruction is critically vital for solving problems.

Need for the Study

It has been shown that providing programming instructions is contributed to be a significant difficulty in the field of computer education (Harangus & Kátai, 2020). CT is an important factor to enhance learners' academic performance in science, technology, engineering, and mathematics (STEM) (Fernández et al., 2018). Barr and Stephenson (2011) stated that in K-12 settings, CT is considered as "a problem-solving methodology that can be automated and transferred and applied across subjects" (p.51). However, the main issue that appears in the existing literature about computational thinking is that there is a need to provide K-12 teachers

with an appropriate preparation of instruction for 21st century skills (Oliveira et al., 2019). It is necessary to provide teachers with curricular content, models, and instructional strategies that could be combined with advanced skills to facilitate the learning process (Barr & Stephenson, 2011).

In addition, "for CT education to further develop, teachers need to be systematically prepared in terms of how to design CT learning activities, how to teach CT, how to assess CT, and how to use technologies to teach CT concepts" (Angeli & Giannakos, 2020, p.3). CT skills are higher-order skills and these "higher-order thinking skills necessitate complex cognitive activities in individuals, it may be a cause of conflict for individuals as they require different learning outcomes" (Güney, 2019, p.145). This sentiment appears clearly in some studies that identify computational thinking as a "difficult concept to study" (Lyon & Magana, 2020, p. 1187).

Moreover, teachers and educators are challenged on how to teach CT, and there are certain difficulties in learning and improving higher-order thinking skills among learners. For example, some challenges that related to the nature of acquired skills gained by the novices include the lack of problem-solving skills, lack of analytical skills, lack of logic and reasoning skills, and lack of algorithmic skills (Butler & Morgan, 2007; Ismail et al., 2010). Arriving to courses with a lack of these skills negatively affects the learners' academic performance. Also, learners face a variety of challenges when it comes to programming learning, which is a complicated process that necessitates a wide range of skills (Alvarez et al., 2022). According to the findings of computer science research, learners become lost and overwhelmed in coding

procedures (Durak, 2020). As a result, learners prioritize coding over designing the solution of the problem which leads to poor gain of CT skills (Bucci et al., 2001).

Lye and Koh (2014) noticed that one major factor that influenced learners in enhancing their CT skills was the cognitive load that is associated with designing, planning, and implementing solutions for problems. Furthermore, when it comes to managing these solutions and automating them through programming, the learners are exposed to an additional heavy cognitive load (Bakar et al., 2019). The lack of suitable guidance and support instruction for CT practices with programming results in poor programming reflection and prevents achieving instructional outcomes (Grover & Pea, 2013; Lye & Koh, 2014).

Thus, building a "supportive environment with plenty of guidance can help gain the necessary skills to think like a computer scientist and achieve one's full potential" (Seneviratne, 2017, p.31). This is especially significant because learners may have to go through a constant trial-and-error process to get at the proper solution rather than thinking and planning before programming (Brennan & Resnick, 2012). Several studies emphasized the need to become skilled in computational thinking before learning to program, and they suggest that programming instruction, as well as computing theory, must be conveyed in a form that learners can understand (Fletcher & Lu, 2009; Guzdial, 2008; Qualls & Sherrell, 2010; Wing, 2008). Fletcher and Lu (2009) stated that "teaching of computational thinking is a noble goal, but it also has pedagogical challenges" (p. 261). Learners must be well prepared before taking on increasingly complicated computational tasks. Promoting CT in instruction will be recognized gradually as learners go beyond simply using technology to create information and solve problems

(Czerkawski, 2015). Additionally, because of the interrelated skills associated with CT, if learners fail to develop their CT skills due to lack of the teaching method, poor instructional materials, or support, they might fail to improve other related skills.

In addition, a limited body of literature explicitly discusses the relationship between promoting computational thinking in programming problems and the instructional design field (Hsu et al., 2018). Therefore, instructors and instructional designers may find guidelines or methods for applying and promoting CT, based on the instructional design's theoretical bases, significantly productive in promoting CT more effectively in learning.

Purpose of the Study

It is important to recognize that computational thinking is a thinking process that uses fundamental computer science principles to solve problems, computers are just machines, and humans are following computational thought processes to generate solutions for those problems (Hsu et al., 2018; Wing, 2008). Faber et al. (2017) argued that problem solving, and associated cognitive skills are important skills that could be developed through CT as a thought process and ultimately implemented as a tool. Many studies investigated various tools to implement CT solutions such as, programming tools, robotics, and unplugged activities (Hartmann et al., 2001; Kafura & Tatar, 2011; Kong & Abelson, 2019). CT is more than just programming; it is a problem-solving method that uses programming as a tool to develop solutions (Hazzan et al., 2020).

Additionally, ISTE (2011) aims to help learners to be computational thinkers rather than users who apply knowledge without cognition; thus, computational thinking stretches across disciplines to achieve this goal. Consequently, achieving different computational thinking sets

requires “focusing on students’ individual needs, requirements, and learning environments” (Labusch et al., 2019, p. 72). Moreover, according to Lyon and Magana (2021) design-based research recommendation in regard to CT, there is a need to scaffold learners, allow learners to share and compare their knowledge, and allow them to promote their explanations and their thinking.

Furthermore, with programming, the digital "environments may not fully support all functions and features of a programming language," thus, there is a need to pay attention to providing enough assistance for programmers, particularly novices (Song et al., 2021, p.2). Moreover, without adequate guidance on how to employ CT skills in problem solving, learning programming may not be successful enough to achieve instructional goals in CT, and it will reflect insufficient learning experience (Grover & Pea, 2013).

On the other hand, providing effective support instructions in the programming domain necessitates a conscious understanding of the different types of programming tasks as well as the proper instructional strategies to ensure that the required assistance is presented suitably (Raj et al., 2018). Solving programming problems may be different based on the type of the problem. CT may be suited to solve well-structured problems, ill-structured problems, and more complicated problems (Moore et al., 2020). Most programming problems encountered in formal learning environments are more well-structured tasks (Kölling, 1999). One example of an effective instructional strategy that has been used to support well-structured problems is worked examples (Crippen & Earl, 2007).

Worked examples as an instructional technique are used to teach learners how to employ functional programming efficiently (Garces et al., 2022). Yet, in order to guarantee that worked examples are used effectively as a support technique, we must consider how to construct them

properly from an instructional design standpoint (Nainan & Balakrishnan, 2019). Considering the following two points, first, in order to assist knowledge construction, worked examples must be suitably constructed (Atkinson et al., 2000; Moreno, 2006). Second, worked examples should be presented in a way that aids learners in organizing and constructing knowledge so that retrieving similar information fits the information needs for future tasks (Driscoll, 2005).

Moreover, from an instructional design perspective, learning theories offer a solid basis in assisting the learner. Multiple learning theories have been developed by researchers in cognitive psychology and education to create learning environments that enhance learning (Greeno et al., 1996), and they function as the fundamental base in the instructional design field. The theoretical bases are a "significant element in ID practice, especially as it guides designers in the selection of instructional solutions" (Richey et al., 2011, p. 63).

Therefore, for the purpose of this study, it is important to think about how to design and develop appropriate worked examples that aim to promote CT skills in well-structured programming problems. The goal of this study is to provide a comprehensive understanding of existing support strategies and tools for teaching CT using worked examples. Providing comprehensive information may assist instructional designers and educators in designing and developing worked examples and used in well-structured programming problems.

Research Questions

This study aims to address the following questions:

1. Based on the literature, how have worked examples (involving computational thinking skills) been designed?

2. What instructional design principles have been recognized in the literature and need to be considered when designing WEs to promote CT skills?

3. What factors or circumstances might have an impact on the development of WEs for well-structured programming problems?

4. What are some examples of effective WEs for well-structured programming problems that have been identified in the literature?

Study Benefits

Enhancing learners' CT skills is very important because CT "teaches us to think, to find ways to solve a problem, to organize and plan the resolution of a task" (Figueiredo, 2017, p.36). The necessity to equip learners to obtain CT skills and become programmers is critical not just for the computer science field but also for a variety of other professions such as data scientists, engineers, and economists (Giannakos et al., 2017). There are several advantages to investigating principles of designing effective worked examples. According to Lye and Koh (2014), when given programming problems, no less than 11% of learners request support from the content's support resources, such as worked examples, to solve the problems. Employing worked examples and practice problems actives the prior knowledge and helps in helps in acquisition of a wide range of problems schemas (Jonassen, 1997). They are helpful for learners, especially novice learners with little prior knowledge, since reviewing an example can reduce cognitive load (McLaren & Isotani, 2011). Also, they improve a learner's understanding of a problem, which results in deep learning (Sweller & Cooper, 1985). Therefore, providing instructional designers and instructors with comprehensive information on developing efficient worked examples is vital to accomplishing future desired educational goals in the realm of CT and programming.

The Study Definitions and Acronyms

Several definitions and acronyms are included in this study; certain definitions will be addressed in Chapters two and three when they appear.

The study used the following acronyms:

- CT: Computational Thinking.
- WE: Worked Examples.
- ISTE: International Society for Technology in Education.
- CSTA: Computer Science Teachers Association.
- ID: Instructional Design.
- STEM: Science, Technology, Engineering, and Mathematics.
- CLT: Cognitive Load Theory.
- IR: Integrative Review.

Chapter Two

Review of the Literature

This chapter provides literature review about computational thinking (CT), definition, core elements, history of CT, and programming as a tool to deliver CT. After, it presents information about the implications of the field of Instructional Design within CT and problem solving. Then, it provides literature review about problem solving, and problem types. Lastly, it presents information about the instructional design of worked examples.

Introduction to Computational Thinking

Computational thinking (CT) skills are rapidly gaining prominence in conjunction with problem-solving skills that are necessary in the twenty-first century. Interestingly, CT proponents claim that the concepts and skills involved with CT are necessary for the success in an increasingly computerized environment (Weintrop et al., 2021). This success requires individuals to have the knowledge, and the needed practical technological skills to forest the ability to solve problems and troubleshoot them effectively (Angeli et al., 2016; Czerkawski, 2015; National Research Council [NRC], 2011). As a result, globally, several countries contributed such skills to the k-12 curriculum to acknowledge the relevance of CT (Balanskat & Engelhardt, 2014).

For instance, the United Kingdom introduced several CT courses across all fields, that consist of computer science course, information technology course, and digital literacy course (Brown et al., 2013). Another example from Poland, where students are taught CT, computing, and problem-solving skills at the primary school level (Sysło & Kwiatkowska, 2013). While Australia has developed a specific CT course for primary and elementary levels with developing CT training as a national teaching course (Armoni, 2013). On the other hand, the South Korean

experience includes integrating 34 credits of computing courses that focus on CT, programming and problem-solving skills (Heintz et al., 2016). In the US, as it stated previously, President Barack Obama established the *Computer Science for All* to provide the current generation of American students with the computer science skills that are important in today's world (Kale et al., 2018).

Additionally, as a recognition of the importance of computational thinking, Carnegie Mellon University in Pittsburgh established a center for Computational thinking for conducting Problem-oriented Explorations (PROBEs), which attempts to apply innovative computing practices and solve problems (Voskoglou & Buckley, 2012).

In conclusion, it can be observed that CT has become an essential topic of national education in many countries, either by providing CT as a specific course or developing a new teaching content that promotes those skills. Obviously, CT is not specialized in one field; it has occurred through many fields, and the current focus requires thinking about designing, delivering, and emerging skills within instruction (Wilkerson et al., 2020).

History of Computational Thinking

The history of computational thinking may be traced back to the late 50s. According to Tedre and Denning (2016), Alan Perlis was one of the earliest computer pioneers to emphasize the importance of coding as a mental tool for solving a wide range of problems. During that period of time, computational thinking has been identified "as algorithmic thinking" that belongs to applying a specific order or sequence of actions to solve a problem and using a computer to automate that solution (Denning, 2009, p.28).

After, in the 1960s, Perlis claimed that computers are more valuable for fostering a particular mindset about problems and designing solutions than using them as means (Katz,

1960). In the 1970s, educators worked to separate computing from mathematics, but it was hard to be accepted into conventional research institutions, which favored theoretically based courses rather than technical subjects (Tedre, 2014).

Seymour Papert was the first who used the terminology of computational thinking (CT) in the LOGO programming language in their book Mindstorms in the 1980s (Papert, 1980). With Papert's LOGO language, learners create several action procedures to control and navigate a turtle that follows the sequenced actions (Bråting & Kilhamn, 2021). Teaching with LOGO begins with sequencing turtle movements and applying recurrence before moving on to the abstraction by assigning sets of several procedures and operating inputs (Michaelson, 2018). Weir (1987) stated "that: Using variables in this way provides a concrete way of approaching the abstract notion of an algebraic variable as it occurs in school mathematics" (p. 23). According to Kafai and Burke (2013) learning and teaching with computers were revolutionized by Papert's work. However, working with computers is not only about running machines. Over various years, computer pioneers, like Dijkstra (1974) and Knuth (1974) reconfirmed that computing's discipline identification is derived from its distinct mental processes, and it is linked to algorithmic thinking.

In addition, before joining MIT, Papert worked closely and was influenced by Jean Piaget and the cognitive development theory known as constructivism (Stager, 2016). At MIT, where LOGO programming developed, Papert defined "mechanical thinking" as the thinking type that learners are exposed to when learning to program in LOGO (Papert, 1993, p. 27).

Consequently, Papert's work inspired various researchers, for example, Mitch Resnick, who was a former doctoral student of Papert's project (Resnick, 2017). He was the director of the Lifelong Kindergarten research group at MIT that developed Scratch, which is a coding platform

for kids, and it inspired by LOGO, but it indicates that programming is “more tinkerable”, “more meaningful”, and “more social” (Resnick, 2014, p. 14).

In conclusion, previous interventions and innovations were the start to educate learners to master the concept of "thinking like a computer" or "thinking like a computer scientist." (FLOYD, 2020, p.30).

The Definition of Computational Thinking

As previously noted, Jeannette M. Wing (2006), in their seminal article, popularized the term computational thinking with the definition: “computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science” (p.33). As a result, CT has gained popularity in the United States K-12 education system (Lye & Koh, 2014). Several scholars have attempted to define computational thinking in a way that is both straightforward and understandable but there are many views regarding a concise definition (Barr & Stephenson, 2011; Brennan & Resnick, 2012; Grover & Pea, 2013). For example, CSTA and ISTE (2011) developed an “operational definition” of computational thinking that represents it “as a problem-solving process” that encompasses many characteristics “but not limited to”, like:

- Formulating problems in a way that enables us to use a computer and other tools to help solve them.
- Logically organizing and analyzing data
- Representing data through abstractions such as models and simulations
- Automating solutions through algorithmic thinking (a series of ordered steps)
- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources

- Generalizing and transferring this problem-solving process to a wide variety of problems (ISTE, 2011, paras.2).

Mannila et al. (2014) stated that CT is “a term meant to encompass a set of concepts and thought processes that aid in formulating problems and their solutions in different fields in a way that could involve computers.” (p.1). The Royal Society (2012) described CT as “the process of recognizing aspects of computation in the world that surrounds us and applying tools and techniques from computer science to understand and reason about both natural and artificial systems and processes” (p. 29). Also, Aho (2012) viewed CT as a thought process included in problems formulation, and “their solutions can be represented as computational steps and algorithms” (p.832).

Similarly, Grover and Pea (2018) explained CT as "thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer-human or machine – can effectively carry out” (p. 21). CT is also seen as a strategy for solving various problems, including systematic design, and planning based on computer science concepts (Soleimani et al., 2016). Likewise, Gretter and Yadav (2016) defined the CT in terms of core computer science principles that include many fundamental skills like abstraction, algorithmic thinking, pattern recognition, and decomposition.

In other words, defining computational thinking, according to some experts, goes beyond computer science conspiracies. It has been stressed that it does not just encompass computer science practical concepts, but it also includes skills like problem-solving skills and algorithmic thinking skills (Kafai & Burke, 2013).

For the purpose of this study, CT is defined as a systematic problem-solving approach that incorporates several computer science fundamentals to solve problems with or without the use of a computer machine.

Core Concepts of Computational Thinking

The benefit of computational thinking is that it could be employed in several forms and applied in numerous fields (Barr & Stephenson, 2011). The primary premise underlying the computational thinking trend is that computer science knowledge and skills often have implications that may also benefit other areas (Ch'ng et al., 2019). According to the literature, CT encompasses a variety of skills that are included throughout CT models and frameworks (Hsu et al., 2018).

Next, Table 1. shows a wide range of CT skills that were discovered in previous studies.

Table 1

Computational thinking skills.

CT Skill	Definition	Resource
Abstraction	The ability to simplify and deal with the main/most critical ideas in a way that makes it easier to develop solutions.	(Barr & Stephenson, 2011; Dasgupta et al., 2017; Grover & Pea, 2013; Wing, 2006)
Decomposition	The ability to break the given problem/task into smaller subproblems/sub-tasks and chunks, that helps to facilitate reaching the solutions.	(Barr & Stephenson, 2011; Dasgupta et al., 2017; Grover & Pea, 2013; Kilpeläinen, 2010)
Data collection	The ability to collect information that facilitates reaching the solutions.	(Dasgupta et al., 2017; CTSA, 2011; Google, 2018).

Data analysis	The ability to evaluate the existing inputs that facilitates reaching the solutions.	(Angeli et al., 2016; Atmatzidou & Demetriadis, 2016; Basu et al., 2017)
Pattern recognition	The ability to identify the common features and the differences among the current inputs to facilitate reaching solutions.	(Dasgupta et al., 2017; Lee et al., 2014)
Data representation	The ability to form inputs in symbols, words, images, or any form of representations to facilitate reaching solutions.	(Barr & Stephenson, 2011; Moreno-León et al., 2015)
Mathematical reasoning / or logical	The ability to apply logical thinking to facilitate reaching solutions.	(Grover & Pea, 2018)
Algorithm design	The ability to construct a step-by-step method to facilitate reaching solutions.	(Barr & Stephenson, 2011; BBC, 2018; Dasgupta et al., 2017)
Parallelization	The ability to deal with tasks simultaneously to facilitate reaching solutions.	(Barr & Stephenson, 2011; Google, 2018; Moreno-León et al., 2015)
Automation	The ability to use a computer (not necessary for programming) or a machine or a medium to deliver the solutions.	(Grover & Pea, 2018; Fletcher & Lu, 2009; Forrest & Mitchell, 2016; Kafai & Burke, 2013).
Modeling	The ability to build a model that emulate the solutions	(Dasgupta et al., 2017)
Transformation	The ability to convert a collection of data.	(Wing, 2006)
Simulations	The ability to build a model that imitates real-world elements.	(Barr & Stephenson, 2011; Grover & Pea, 2013; Wing, 2006)

Conditional logic	The ability to identify the pattern that connects several events.	(Grover & Pea, 2013)
Visualization	The ability to connect between elements visually to have a better understanding.	(Hsu et al., 2018)
Testing and debugging	The ability to recognize errors in dealing with tasks/problems that prevent from finding the correct solutions.	(Berland & Lee, 2011; Kazimoglu et al., 2012)
Generalization	The ability to develop rule model solutions for common problems	(Palts & Pedaste, 2020).

As a result of combining the skills mentioned above, researchers construct a set of computational thinking frameworks that are utilized in a variety of studies and investigations. There are several frameworks for introducing computational thinking, each intended to promote particular basic concepts or components of computational thinking (Romero et al., 2017).

For instance, Barr and Stephenson (2011) developed a framework based on a combination of CT skills in the specific domain of context. The framework consists of data collection, decomposition, data representation, abstraction, algorithms, automation, and parallelization and simulation, and it has been applied in different contexts like computer science, science, mathematics, arts, social studies, and languages (Barr & Stephenson, 2011). While problem decomposition, pattern identification, abstraction, and algorithmic thinking are all part of Hoyles and Noss's (2015) framework.

Moreover, Grover and Pea (2013) developed a CT framework that includes several skills like problem decomposition, data representation, abstraction, pattern recognition, and algorithmic design to implement possible efficient and effective solutions to a problem. Also, a

two-dimensional framework developed by Gouws et al. (2013) includes a specific dimension that combines processes and transformations, modulization and abstract, patterns recognition and algorithmic design, and evaluations. The second dimensional set covers the various degrees of practicing for these skills: recognize, understand, apply, and integrate (Gouws et al., 2013).

Furthermore, an alternative framework suggested by Brennan and Resnick (2012) consists of three categories as the following:

1. Computational concepts indicate "sequences, loops, parallelism, events, conditionals, operators, and data" (Brennan & Resnick, 2012, p.3).
2. Computational practices indicate experimentation and iteration, test and debugging, and abstracting and modularization.
3. Computational perspectives indicate learners' understanding and their communication with others in the digital world.

Additionally, Weintrop et al. (2016) have designed a computational thinking taxonomy that includes a wide range of practices to be applied in mathematics and science. The taxonomy includes many computational thinking skills in association with other practices such as data representation, modeling and simulation, problem decomposition, and algorithmic thinking, as core concepts that are applied in math and science (Weintrop et al., 2016).

As well, Selby and Woollard (2013) expanded the idea based on Wing's perspective by establishing a framework comprising five CT abilities: problem decomposition, abstraction, algorithms, evaluation, and generalization.

For the purpose of this study, we will concentrate on Selby and Woolard (2013) computational thinking set skills as: decomposition, abstraction, algorithms, evaluations and generalization.

Programming as a Vehicle to Promote Computational Thinking

Problem-solving and cognitive skills are important skills that could be developed through CT as a thought process that leads to implementing a solution through a tool (Faber et al., 2017). That is to say, CT skills explore and solve problems in various disciplines, and those skills in any combination are required to be supplied or automated via particular tools, which result in a final outcome of the computational process (Hsu et al., 2018; Lye & Koh, 2014; Zhong et al., 2016). Numerous researchers have looked into various tools for incorporating computational thinking design, such as programming tools, robots, and unplugged activities (Kong & Abelson, 2019; Kafura & Tatar, 2011; Hartmann et al., 2001).

Several instructors have claimed that teaching CT using programming languages is the most manageable and appropriate method to promote CT skills (Bers et al., 2014; Kazimoglu et al., 2012; Lye & Koh, 2014; Wolz et al., 2011; Zhong et al., 2016).

However, there is a debate concerning the assumption that applying computer science concepts is the same as programming, which is not correct (Fletcher & Lu, 2009). To illustrate, Fletcher and Lu (2009) claimed that "as proficiency in basic language arts helps us to effectively communicate and proficiency in basic math helps us to successfully quantitate, proficiency in computational thinking helps us systematically and efficiently process information and tasks" (p. 23).

Furthermore, learning to develop CT skills is not the same as learning computer science as a topic, where computer science includes programming as a sub-field (Cansu & Cansu, 2019). According to the Computer Science Teachers Association (CSTA), computer science is defined as "the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society" (Seehorn et al, 2011, p. 1).

Whereas computer programming is a broad term that refers to a variety of skills and activities that all include writing instructions for a computer which is required to achieve a particular result (Nouri et al., 2020).

According to Yadav et al. (2017) “the idea of programming to support computational thinking goes beyond just coding and syntax and includes problem decomposition and algorithmic thinking” (p.1061). Therefore, programming helps to promote CT skills as being a vehicle to automate the outcomes of those skills.

Given the importance of computational thinking in problem-solving, various projects utilized programming tools as a vehicle to promote CT skills (Yadav et al., 2017). For instance, several K-9 teachers from different disciplines implemented CT concepts in various ways, including data collection, data analysis, data representation, and algorithm design with programming (Mannila et al., 2014). Likewise, Garneli et al. (2015) conducted a systematic review of the most commonly used programming tools for promoting CT skills in the K-12 computer curriculum. The research discussed multiple programming obstacles and provided examples of the most effective instructional practices for introducing CT skills (Garneli et al., 2015).

Also, Werner et al. (2012) developed a model for teaching and promoting CT skills using an innovative digital game that incorporates programming concepts. This game project allows learners to discover fundamental computer programming practices by supporting key CT ideas and explaining how they might be transferred to programming structures (Werner et al., 2012).

Moreover, Han et al. (2015) developed Entry, a visual programming environment that utilizes HTML5-based visual programming to assist learners in improving their computational

thinking skills. Learners with little or no programming experience may use Entry to build, upload, and present their own programming projects and learn programming in an exciting and simple method (Han et al., 2015). With relevance to visual programming, Scratch is one of the most common programming tool associates with the application of CT, it is a block programming language designed and developed at MIT media lab by the lifelong Kindergarten group, and it includes interacting with an online community (Figueiredo, 2017). When learners create programming projects with Scratch they are supposed to create, think, play and share thoughts collaboratively (Brennan & Resnick, 2012).

In addition to Scratch, Alice is another example of a visual programming environment that has been utilized with CT skills and emphasized Java language concepts. Alice enables learners to create animations, design interactive stories, and program simple 3D games through the use of computational thinking skills and fundamental programming concepts (Papadakis et al., 2014). Furthermore, in collaboration with MIT, Google created the APP Inventor (AI) programming environment, which was built specifically for beginner programmers (Park & Shin, 2019). Thus, programming plays an important role in delivering CT skills as Sanz (2015) stated that "computational thinking conducts a detailed analysis of problems. This helps not only in coding, but also in being able to analyze and thoroughly understand problems, identify patterns, and extrapolate solutions" (para.7).

The preceding programming project examples, as well as other projects such as TACCLE-3, Blockly, Code.org, Tynker, and Kodu are intended to enhance computational thinking skills inside and outside the classroom (Aggarwal et al., 2017; Baratè et al., 2017; Garcia-Penalvo, 2016; Kale & Yuan, 2021; Ouyang et al., 2018).

To sum up, it is essential to consider that computer programming should be regarded as a tool for developing CT-related knowledge and skills. It is not only an information and communications technology (ICT) tool, and it is an important tool for developing CT skills (French Academy of Sciences, 2013). As a result, rather than just teaching learners how to write code, programming courses when integrating with CT-related skills will lead to more successful learning objectives (Buitrago-Flórez et al., 2017). Learners who develop CT skills via programming will be able to gain algorithmic thinking, problem solving, reasoning, and more skills (Tu & Johnson, 1990).

Instructional Design, Computational Thinking and Problem Solving

When it comes to comparison, there is a high degree of similarity between computational thinking and problem solving and information processes (Labusch et al., 2019). With computational thinking skills:

> The students demonstrate the ability to identify a problem, break it down into manageable steps, work out the important details or patterns, shape possible solutions and present these solutions in a way that a computer, a human, or both, can understand. Computational thinking can also involve structuring and manipulating data sets to support the solution process (IEA, 2016, p.1).

Additionally, learners can enhance their critical and problem-solving skills by relying on the core skills and practices of CT (Zhang & Biswas, 2019). CT encourages us to think, seek solutions to problems, arrange and plan the implementation of a task, and utilize techniques and strategies for dealing with various tasks (Figueiredo, 2017). Therefore, several studies have attempted to achieve moderate success in assisting learners' performance in completing such activities (Song et al., 2021).

It is increasingly expected that the new generations would be an effective problem solver (National Research Council [NCR], 2012), educational systems, teachers and instructors have to promote problem-solving skills with a decent instructional design approach. Educational scholars and researchers have argued about the correlative connection between context and instructional design in addition to its influence on learning outcomes for a long time (Hiemstra & Sisco, 1990; Tessmer & Richey, 1997). Aside from the fact that learning takes place in context, it is also well known that learning is influenced by context (Snow, 1994), in particular with respect to acknowledging that instructional designers might "adjust contextual factors to facilitate instructional needs" (Tessmer & Richey, 1997, p.88). According to Gagne (1985) "the central point of education is to teach people to think, to use their rational powers, to become better problem solvers" (p.85). In the context of problem solving, CT can be employed to develop an appropriate strategy for problem solving, including utilizing a proper and efficient algorithm to solve problems in a digital system via programming (Park et al., 2015).

Nonetheless, it is essential to highlight that here are several obstacles to overcome in the learning process of CT, with a particular focus on the learner's programming performance. According to Song et al. (2021), those difficulties include task difficulty, coding difficulties, time management, and lack of guidance while practicing. CT does not imply that problems should be addressed in the same way that a computer does, yet instead, critical thinking should be utilized in conjunction with computer science principles (Kazimoglu et al., 2012). Further, teaching computational thinking is more challenging than teaching some CS courses like CS0 or CS1 (Fletcher & Lu, 2009). It is also commonly known that learners lacking problem-solving skills would fail to develop their solutions in programmed tasks (Koulouri et al., 2014).

Also, many learners begin coding without understanding the critical stages in CT, which causes them to make several errors and prevents them from reaching the correct solution (Rajaravivarma, 2005). Consequently, teaching CT is necessary and demanding before creating programming codes since it provides learners with a description of the problems and proper planning for the solution before carrying it into the computer (Kazimoglu et al., 2012).
That led us to think about the importance of the instructional design role in prompting those skills (CT and problem-solving skills) on one hand, and its critical role in developing effective teaching and learning strategies on the other hand (Branch & Kopcha, 2014). As stated by Jonassen (2000)" if we believe that cognitive and effective requirements of solving different kinds of problems vary, then so, too, must the nature of instruction that we use to support the development of problem-solving skills" (pp.81-82). It is also critical to provide a sufficient support structure for learners as they acquire computational thinking skills and programming to solve problems (Seneviratne, 2017). Furthermore, from instructional design perspective "the problem-solving process depends upon the problem solver's understanding and representation of the problem state and goal state" (Jonassen, 1997, p.66). It is necessary to provide information on problem solving in the next part to better understand the problem-solving process and how it relates to programming instructions and CT skills. The following section will provide extensive information on problem solving.

Problem Solving

As we are constantly confronted with problem-solving scenarios in daily life, problem solving is becoming one of the most important skills that students should learn. "Problem-solving is inescapable in human life and is crucial for human survival" and it is one of the most crucial and fundamental skills in the modern world (Rahman, 2019, p.72). According to Binkley

et al. (2012), problem-solving skills, digital skills, and creativity are regarded as important in the twenty-first century.

In the research field, problem-solving has a long and varied history (Lintern et al., 2018; Newell & Simon, 1972; NRC, 2012). For several decades, different disciplines investigated problem solving, for instance, mathematics (Taplin, 2006), physics (Hoskinson, 2013), chemistry (Potvin, 2019), software engineering (Gorschek & Mendez, 2021), and aerospace engineering (Altybayeva & Wood, 2019) and many other fields. Furthermore, there is a large and expanding body of literature investigating problem solving, particularly in education (Kim & Hannafin, 2011).

Before proceeding to problem solving and problem types, it is important to define the problem-solving process. Anderson (2015) defined problem solving as a "goal-directed behavior that often involves setting subgoals to enable the application of operators" (p.183). In addition, Gagné and Briggs (1974) viewed problem solving as high-order intellectual skills.

A mental representation of the problem, as well as internal or external modification of the problem space, are all part of the problem-solving process (Jonassen, 2000; Reed, 201 6; Simon & Newell, 1971). It is viewed as performing a set of functions on an initial state that is identified by the problem constraints in order to reach a specific goal (Chi & Glaser, 1985). To clarify, "problem solving requires the mental representation of the situation", or what it is known as problem space (Walker, 2015, p.32). Also, it necessitates "activity-based manipulation of the problem space," either by internal or external manipulation (Jonassen, 2000, p.65). The problem space is seen as an essential feature that distinguishes problem types (Reed, 2016). According to Newell and Simon (1972) it "viewed as the space that is generated by starting with a set of initial objects and working outwards from these to other objects that can be reached from them, without

imposing any particular direction on the search" (p.428). In addition, manipulating the mental representation of the problem space occurs via several methods like thinking, searching for solutions, discussing with others, and so forth (Pel, 2018).

Further, problems can appear as an impediment that must be solved in order to achieve the desired goal and the solver interacts with problem components that are contained within the problem space (Wood, 1983). The problem solver constructs a mental model of the problem space, beginning with the stated goal, understanding limitations, and recognizing the provided information about the problem (Reed, 2016). The mental representation expands as the solver gains knowledge about the problem or the task, including previous procedural expertise, domain specific knowledge, and the new information that is associated with the problem space (Jonassen, 2000).

On the other hand, the problem space holds information regarding the steps that are required to solve the problem (e.g., the equation) (Reed, 2016; Simon & Newell, 1971; Zelazo et al., 1997). Therefore, problem solvers would employ various strategies to search through the problem space to find a path that leads to the desired goal (Dunbar, 1998).

Problem Classification/ Types

Greeno (1976) assumed that different types of problems require several skills; therefore, distinguishing between those problem's types is important. The typology of problems introduces several categories that result in different outcomes, need distinct abilities, and are employed in various contexts (Arlin, 1989; Brabeck & Wood, 1990; Chi & Glaser, 1985; Jonassen, 2000). What we know about problem types is largely based upon empirical studies that investigated how they are varied and their features. The distinction classification of the types includes well-structured problems and ill-structured problems (Dillon, 2002; Jonassen, 2000; Kitchner, 1983).

The classification of the problems is made based on the existence of the problem components. Several researchers have shown that problems with definite initial states, solutions, and limitations refer to well-structured problems; in contrast, vague statements refer to ill-structured problems (Jonassen, 1997; Kitchener, 1983; Lee, 2004). This distinction between well-structured and ill-structured problems is made to illustrate the accuracy of problem solving rather than composing a precise line between types of problems (Sternberg, 1994). The current research focuses on well-structured problems.

Distinguishing Between Well-structured and Ill-structured Problems

Numerous scholars have attempted to define the ill-structured problem to prevent misunderstanding compared to the well-structured problem and thus propose appropriate solutions to specific problems (Allaire & Marsiske, 2002; Dillon, 2002; Duch, 2001; Gick, 1986; Greeno, 1976; Kitchener, 1983; Lee, 2004, Meacham & Emont, 1989; Schraw et al., 1995; Sinnott, 1989; Voss & Post, 1988). In addition, Newell and Simon (1972) considered the classification in terms of the degree to which the problem's goal is defined, the provided information, and whether or not problem-solving constraints are provided.

Well-structured problems that include a "well-defined initial state (what is known), a well-defined goal state (nature of the solution is well defined), and a constrained set of logical operators (known producer of solving)" (Jonassen, 2000, p.67). Kitchener (1983) and Sternberg (1994) attempted to describe a well-structured problem in the context of possible solutions. As reported by Kitchener (1983), well-structured problems are problems that involve all the essential elements to solve problems effectively. Sternberg (1994) described well-structured problems as problems with well-defined solutions approaches.

On the contrary, ill-structured problems tend to be problems with no definitive or specific one answer, and they have limited assurance about possible solutions (Kitchener, 1983; Sinnott, 1989). Likewise, Chi and Glaser (1985) explained that ill-structured problems are challenging due to the difficulty in specifying the starting state, formulating feasible actions to alter the initial state, and achieving the goals. In addition, ill-structured problems have many pathways and techniques for achieving different solutions and goals (Schraw et al., 1995).

In conclusion, to detect the structuredness of the problem, several judgments may be formed, including identifying the solution starting state, the desired state, the problem restrictions, and application rules (Chi & Glaser, 1985; Greeno, 1976; Jonassen, 2000; Lee, 2004; Luszcz, 1989; Wood, 1983). Structuredness has an enormous impact on problem-solving procedure because it facilitates the problem's representation, which is necessary for efficient problem solutions (Brabeck & Wood, 1990; Dabbagh, 2002; Duch, 2001; Jonassen, 1997). Lastly, well-structured problems define a specific circumstance that necessitates the application of a set of rules and principles, confined by specified constraints, and using predicted activities that are commonly employed to handle similar problems (Pulgar, 2019).

Next section will provide an overview about the characteristics of both problems classifications, and it will start with ill-structured problems, then the well-structured problems characteristics.

Ill-structured Problems Characteristics

Ill-structured problems as a term are types of problems that refer to complex, real-life situations; also, they are ones that learners encounter on a daily basis (Shin & McGee, 2018). Those problems are generally framed in real-life scenarios rather than particular limitations;

nonetheless, the constraints may include social, economic, and cultural factors subject to interpretation and negotiation by problem solvers (Goel, 1992).

Ill-structured problems, like well-structured problems, have distinct features, which include the following:

- They contain components that are unknown to us (Wood, 1983).
- They have undefined or unclear goals (Voss & Post, 1988).
- They can have numerous solutions and various paths, or none at all (Kitchner, 1983).
- They required various criteria for evaluating the effectiveness of the possible solutions (Jonassen, 1997).
- They do not have any average cases since the problem components are context-dependent and distinctive (Spiro et al., 1988).
- They either have multiple solutions or nor solution (Kitchner, 1983), they also have "no consensual agreement on the appropriate solution" (Jonassen, 1997, p.68).

To summarize, ill-structured problems are identified by a lack of specified and apparent goals, in addition to indicate predictable solutions (Chin & Chia, 2006).

Well-structured Problems Characteristics

Well-structured problems are often seen in formal educational settings, schools, educational institutions, and textbooks (Mason, 2019). As previously said, solving these problems requires implementing a particular set of steps and rules, and they have a well-defined initial solution state, a well-defined desired state, and a limited number of logical procedures (Chi & Glaser, 1985; Greeno, 1976; Jonassen, 1997).

The following are some essential characteristics that differentiate the features of well-structured problems:

- They are problems "for which there are absolutely correct and knowable solutions" (Kitchner, 1983, p.223).
- They required a stated criterion to evaluate the suggested solutions (Simon, 1973).
- They required presenting all the components of the problem (Jonassen, 2000).
- The initial problem state, the goal state, and any additional information that require at least one problem space to reach the solution (Simon, 1973).
- The problems involve well-structured concepts and rules in a particular domain that are further well-structured and anticipated (Jonassen, 1997).
- They have identifiable solutions with a known or deterministic connection between decisions and problem states (Wood, 1983).

To sum up, logic, algorithmic, or story problems, which are frequently seen on schools' exams or "at the end of textbook chapters", are primarily well-structured in nature (Jonassen, 2000, p.67). Initial states, goal states, and transformation functions are all well-defined in these problems (Goel, 1992). Subsequently, success is measured by how well the solution works and how well it matches the correct answer (Toy, 2007).

This study is focusing on promoting CT skills in programming instructions within well-structured problems as being the most common problem included in formal educational settings in schools and universities (Jonassen, 1997). They are also recognized as routine problems, requiring the application of suitable algorithms to produce solutions that may fully meet the main conditions of the solutions of the problems (Cross, 2021; Visser, 1996). Especially, that "solutions to ill-defined problems may be ambiguous, it is not possible to forecast whether an algorithm may fit the initial requirements" (Casakin, 2002, p.2).

The Knowledge of Problem-solving to Promote Computational Thinking

Voskoglou and Buckley (2012) affirmed that CT is a problem-solving technique used in the computer science field and integrated into other fields, and it assists in promoting critical thinking and knowledge to solve problems.

Hence, with instructions based on problem solving, instructors should pay attention to the knowledge base. To prepare skilled problem solvers, learners need to be encouraged and supported in gaining the knowledge required to solve problems (Kale & Yuan, 2021). Effective problem solving relies on domain-specific knowledge relevant to the problem, problem-solving techniques, and feelings and beliefs about one's interests and skills to solve the problem (Mentz, 2013).

Based on problem-solving research, Mayer and Wittrock (2006) classified four types of knowledge necessary to solve problems as the following:

- Factual/Conceptual knowledge: refers to the fundamental information, such as facts and concepts, that must be recognized to solve a problem (Mayer & Wittrock, 2006).
- Strategic knowledge: refers to the approaches required to develop solutions to a particular problem (Mayer & Wittrock, 2006).
- Procedural knowledge: refers to knowing a series of specific procedures for completing a task (Mayer & Wittrock, 2006).
- Metacognitive knowledge: refers to “the awareness of the one’s own thinking process, which supports the self-regulation in the problem-solving process” (Kale & Yuan, 2021, p.621).

Aside from the necessity of a knowledge foundation in designing instructions to promote various skills, instructional design pays attention to producing, maintaining, and evaluating instructional materials in terms of numerous theoretical basic elements (Martin, 2011).

The Role of Instructional Design

Instructional design, as "an applied, decision-oriented field", emphasizes the importance of learning theories as paradigms for understanding human learning and knowledge transfer (Smith & Regan, 2005, p.18). It plays a significant role in supporting and enhancing several skills including problem solving. According to the research, specific instructional techniques supporting several previous types of knowledge can assist the problems solving processes (Akcaoglu, 2014; Kale et al., 2018; Mayer & Wittrock, 2006).

Instructional design provides numerous theoretical guidelines for efficiently developing instruction that promotes skills such as problem-solving skills. Smith and Ragan (2004), suggested several techniques for improving acquisition skills and assisting learners, particularly novices in learning, such as:

- Using various problem representations, such as visuals and analogies.
- Helping learners to solve problems by providing them clues and hints at multiple levels.
- Providing search information techniques to assist learners in problem-solving.
- Decreasing the cognitive load by using job aids and visuals.

Instructional design is critical for improving the effectiveness of learning experiences. The significance of instructional design varies between guiding learners to acquire knowledge, engaging learners in the learning process, and achieving the instructional goals (Neelakandan, 2021). Instructional design is aiming to simplify challenging capabilities during the initial

learning phase so that learners may develop schemata without becoming overwhelmed (Margulieux et al., 2020). A schema (the plural of schemata) is “a data structure for representing the generic concepts stored in memory” (Rumelhart, 2017, p.34). Schemata known as “packets of knowledge” is associated with the schema theory, which explains its representations, and operates in particular ways to promote knowledge acquisition (Driscoll, 2005, p.129).

Moreover, “if schemes are the primary determinants of expertise, then good instructional design may be defined as that which facilitates schema acquisition” (Cooper, 1990, p.109). On the other hand, cognitive load theory indicates that effective instructional material is necessary to promote learning by addressing “cognitive resources” into tasks related to schema acquisition (Cooper, 1990, p.109; Sweller, 1988). Cognitive load theory (CLT), developed by Sweller (1988), is concerned with the human cognitive architecture, schema acquisitions, and a variety of other cognitive processes.

Managing the Cognitive Load

Various learning theories have been developed to enhance the learning process and ensure achieving the desired goals (Greeno et al., 1996). In cognitive science, cognitive load theory (CLT) is considered as one of the most prominent topics (Zavgorodniaia, 2020). CLT is identified as “the load imposed on an individual's working memory by a particular (learning) task” (Van Gog & Paas, 2012, p.599).

CLT highlighted the transformative value of learning through the observation and/or the imitation of others' performance (Sweller, 2004; Sweller & Sweller, 2006). The imitation of other’s performance is agreed with social learning theory by Bandura (Bandura, 1986). Discovering a vast amount of information on one's own is difficult and drawing knowledge from others is beneficial with "reorganize it to fit in with one's existing knowledge and use it to one's

own purposes" (Van Gog & Rummel, 2010, p.156). CLT differentiates three types of cognitive load that may affect the learning process as the following:

- Intrinsic cognitive load is underlying complexity of the new information (Chandler & Sweller, 1991)
- Extraneous cognitive load is underlying distracting or unnecessary information (Chandler, & Sweller, 1991)
- Germane cognitive load is underlying the linking between the new information and the current information (Sweller et al., 1998)

CLT as a learning theory is based on the idea that working memory and long-term memory comprise the human cognitive architecture and that cognition occurs through the development of schemas stored in long-term memory (Sweller, 1988; Sweller, 2010). The load in working memory includes the mental process like, solving a problem, reasoning, and thinking (Caspersen & Bennedsen, 2007). Also, working memory is estimated to be restricted to handle about seven chunks of information, where humans can only handle two or three elements simultaneously (Miller, 1956). According to Garner (2002) “the degree of interactivity between the elements also affects the capacity of working memory” (p.2). Sweller (1988) noticed that when learners solved problems, they employed means-ends analysis, which necessitated a significant amount of cognitive processing and influenced schema development.

In that regard, three essential cognitive processes are required to enhance the learning process; the initial process involves the “selection of the relevant information,” while the second and third processes include the “organization of the incoming information and its integration with existing structures in long-term memory” (Wouters, 2017, p.187). As a result, the majority of CLT research is dedicated to discovering instructional strategies that actively regulate the

cognitive process with a particular emphasis on minimizing the cognitive load (Caspersen & Bennedsen, 2007).

To illustrate, observing others' performance on a variety of activities is one consideration of CLT that has captivated the researchers' interest in improving the learning process (Sweller, 1988; Van Merrienboer & Sweller, 2005). Several studies have been conducted on investigating the influence of observing others' performance, often known as modeling, which has a sharing perspective with social-cognitive theory (Bandura, 1986; Collins et al., 1988; Van Merriënboer, 1997). Modeling examples "involve an adult or peer demonstrating how to solve a problem" (Mayer, 2020, p.912). While worked examples (WEs) provide learners with a written description, precisely, they provide learners with step-by-step instructions for how the problem could be solved (Van Gog & Rummel, 2010).

In some scenarios, examples may be provided in a different style than traditional (e.g., video model), and this resulting in indicating those examples as a type of observational learning theory (Renkl, 2011; Rummel et al., 2009; Schworm & Renkl, 2007; Van Gog & Rummel, 2010). For more clarification, Mayer (2020) provided a classification of those types of examples, including, traditional worked out examples (description with or/and graphic), modeling examples (demonstration via videos or graphics), practice problem (problem exercise on paper or computer), and concrete analogy (text-graphics, audio-graphics, game actively, virtual activity).

As reported by Van Gog and Rummel (2004), the primary distinction between modeling the example and providing WEs is that with modeling, the individual illustrates and performs the procedure and explains what is occurring. In general, observing or noticing other people's performance is crucial since it improves learning and guides the construction of cognitive representations (Wouters et al., 2008). This performance is divided into two categories; physical

skills and process performance (e.g., playing tennis) and cognitive skills and process performance (e.g., problem-solving) (Collins et al., 1988). Observing physical skills performance occurs immediately, whereas cognitive skills performance is not noticeable, consequently, further explanation such as visual scaffolding is necessary (Wouters et al., 2008; Wouters et al., 2007).

Modeling and WEs are guiding the learner in solving problems, allowing "the learner to construct an initial mental model" that can assist learning throughout implementation (Wouters et al., 2008, p.187). Furthermore, it is essential to recognize the instructional design of those approaches. Both Sweller (2004) and Bandura (1986) stress the significance of using symbolic representations in words and visuals as a means of sharing knowledge with others. Example-based theory, on the other hand, provides principles for the implementation of effective examples (Renkl, 2014b). It is crucial for instructional designers to understand the impacts of CLT since it helps them in creating materials more effectively and it advocates reducing the learning load (Sweller, 1988).

However, to reduce the cognitive load while prompting computational thinking skills, a limited number of techniques have been identified. For example, worked examples, scaffolding activities, short videos, gestures with languages, multiple representations of the computational tasks, and interactive web-based environments (Grover et al., 2014; Krugel & Hubwieser, 2017; Looi et al., 2018; Moore et al., 2020). Accordingly, the common technique that also has several applications in programming domains is worked examples.

Example-based learning and Worked Examples (WEs)

Observational learning, analogical reasoning, and learning from worked examples are three areas that are integrated with example-based learning (Renkl, 2014b). According to Dyer et

al. (2015) these research's areas revealed common principles that underlined the significance of the following:

- "Learning with examples of a particular case" (Dyer et al., 2015, p.2).
- "Providing multiple examples" (Dyer et al., 2015, p.2).
- "Linking examples to underlying principles" (Dyer et al., 2015, p.2).
- Motivating learners to establish links between examples and concepts (Dyer et al., 2015).

Worked examples as a form of example-based learning are one of the most well-established effects in learning and instruction research (Chen et al., 2020; Renkl, 2012).

In consonance with cognitive load theory, learning problem-solving topics is aided by using WEs rather than traditional problem-solving techniques (Sweller & Cooper, 1985**;** Cooper & Sweller, 1987). Jonassen (2011) reported that most problem-solving support strategies, such as worked examples, assist learners in constructing their schema while also reducing cognitive load. Also, researchers have made significant contributions to worked examples when it comes to learning (Atkinson et al., 2000; Chi et al., 1989; Ward & Sweller, 1990).

Worked examples (WEs) defined as a method of providing step-by-step instructions to accomplish a particular task or solve a problem (Atkinson et al., 2000; Ayres, 2012). In other words, worked examples illustrate "algorithmic solutions" as techniques until reaching the completion of the task (Renkl, 2014a, p.392). Worked examples "give learners concrete examples of the procedure being used to solve a problem" (Morrison et al., 2015, p.22). They consist of typical components that include problem statements and sets of instructions that define the best producers to use to complete a specific task (Chen et al., 2016).

Worked examples have been widely and effectively employed in several fields, specifically with well-structured problems, such fields include algebra (Cooper & Sweller, 1987; Sweller & Cooper, 1985), mathematics (Hesser & Gregory, 2015; Retnowati, 2017), statistics (Paas, 1992), physics (Badeau et al., 2017; Saw, 2017; Van Gog et al., 2006), and engineering (Dart et al., 2020).

Employing WEs in educational contexts was not a novel idea. Its origins may be drawn back to the mid-1950s through mid-1970s (Chen & Kalyuga, 2020). Research with WEs is linked to the problem-solving context, and it was the first example-based study conducted within the context of cognitive load theory (CLT) (Sweller & Cooper, 1985; Cooper & Sweller, 1987). Where learners were given worked examples first and then afforded the opportunity to solve problems on their own (Chen & Kalyuga, 2020). In comparison to traditional problem solving, Sweller and Cooper (1985) believed that the use of worked examples might improve productivity and skill development. Research shows that if novice learners are provided with worked examples and a variety of problems, they may obtain an additional load in working memory (Sweller & Cooper, 1985; Cooper & Sweller, 1987). As a result, numerous suggestions on this matter propose presenting worked examples and then repeating the same problems in practice (Chen et al., 2020; Sweller & Cooper, 1985).

Worked examples are critical in offering constructive problem-solving guidance. They eliminate the necessity for applying ineffective problem-solving techniques, enabling learners to concentrate all of their mental effort on following the worked-out solution method (Van Gog & Rummel, 2010; Sweller & Cooper, 1985). Based on the research, worked examples may be utilized as a metaphor for problem solving that is likely to address physically practical examples or mentally recalled previously studied examples (Reed et al., 1994). Furthermore, learners are

able to generalize fundamental rules from examples, enabling them to solve problems in which portions of the solution method must be modified (Cooper & Sweller 1987; Paas, 1992; Paas & Van Merriënboer, 1994).

Benefits of Worked Examples

The benefits of employing worked examples within problem solving have been extensively proved in several studies across various disciplines. Beginning with the first study in algebra, Sweller and Cooper (1985) conducted a series of experimental research in algebra courses, within a diverse sampling including 9-year-olds, 11-years-olds, and undergraduate students. According to the findings of the study, effective worked examples aided the learning process by reducing consumer time and eliminating errors (Sweller & Cooper, 1985). Another research in the statistics domain, Paas (1992) revealed that using a sequence of worked examples followed by completion practices yielded successful outcomes. Also, Caspersen and Bennedsen (2007) looked at the usefulness of providing worked examples in programming courses. Their research came to the conclusion that combining worked examples with problem solving is a successful move.

Consequently, it is obvious that worked examples are an effective educational method for providing step-by-step instructions on how to solve a specific problem (Kirschner et al., 2006; Wittwer & Renkl, 2010). Research shows that employing worked examples reveal numerous advantages. For instance, it enables learners to build on prior knowledge when creating new ones in order to understand how to solve problems (Chen et al., 2019). Also, it aids in reducing cognitive load when dealing with problems by helping learners to envision problem solutions with step-by-step instructions (Chen & Kalyuga, 2020). Moreover, this results in the use of recently acquired information in new problem-solving scenarios (Lange et al., 2021). Also,

providing inexperienced or novice learners with worked examples earlier in the problem-solving process can eventually help them solve problems effectively (Chen et al., 2019). Furthermore, learning with worked examples has been considered to be more beneficial for learning than problem solving procedure apart because it decreases unnecessary processing that may occur when solving problems without worked examples (Klepsch & Seufert, 2020; Retnowati et al., 2017).

Generally, worked examples as an instructional pedagogy has been shown to be successful in decreasing cognitive load and providing more attention to be invested in enhancing the development of long-term memory and gaining knowledge (Davenport, 2019).

Worked Examples with Programming Domain and Computational Thinking

Learning programming entails not just a thorough understanding of programming concepts, but also a thorough understanding of the programming process. Although there have been several studies that used worked examples for research purposes, there have been fewer studies in the programming domain. The first notable research demonstrating the importance of examples for programming domain began in the early 80s (Hosseini et al., 2020). According to Pirolli and Anderson (1985), utilizing examples can assist in guiding learners to solve multiple programming problems. Several investigations have emphasized the importance of employing worked examples to help learners gain programming skills in many programming languages. Some programming languages that have been taught using worked examples are listed below:

- LISP, a functional programming language, developed in the 1950s and used in Artificial Intelligence programming AI. (Lieberman, 1986; Weber & Brusilovsky, 2001).
- SQL was developed in the early 1970s, and is used for managing data in databases (Melton & Simon, 1993; Shareghi Najar et al., 2015).

- Java was developed in the early 1990s, and it is used for internet-based applications and building web pages (Austerlitz, 2002; Brusilovsky & Yudelson, 2008).
- JavaScript, a text-based programming language developed in the middle of the 1990s and used for designing interactive web pages (Jensen et al., 2009).
- Snap*!,* a block-based educational graphical programming language developed in (Zhi et al., 2019).
- Scratch, a block-based visual programming language, was developed in 2003 and targeted young learners (Maloney et al., 2010; Zhou et al., 2016).

The implementation within worked examples in programming courses took many forms. For instance, using advanced tutoring to provide suitable support for learners while they are coding. Kumar (2016) utilized a computer-assisted tutorial method based on questions that directed learners in their learning. If a learner couldn't answer the questions, the system gave them an example with the identical problem and a precise explanation as feedback. Also, learners can select from a list of self-explanation questions with feedback (Kumar, 2016). Another example is research conducted by Abdul-Rahman and Du Boulay (2014) to investigate the effect of using an interactive web-based interface that includes several interactive worked examples, self-explanations, and hints as a strategy to guide learners while they learn to code. In addition, Najar and Mitrovic (2013) found that integrating WEs within an intelligent tutor contributed to a comprehensive scaffolding system consisting of various problems with encouraged self-explanations and queries assessing learners' ways to solve those problems. Additional forms of worked examples can be found in textbooks, both printed and online. According to Hosseini et al. (2020) this form of WEs include programming tasks, WEs of a code, and an explanation of the coding algorithm.

As it has been shown, WEs may be found in textbooks and tutorials as written comments linked to lines of code, videos and animated examples have taken their place as another way of modeling the coding work. Sorva et al. (2013) used visualization to teach the coding process in introductory programming courses. The research recommended employing visuals to assist inexperienced programmers and realizing the importance of the cognitive load and engagement when developing future programming courses (Sorva et al., 2013).

Yet, few researchers have looked at using WEs to prompt CT skills using a range of techniques (for example, programming). Although programming is the most popular delivery mechanism for boosting CT skills because they are viewed as problem-solving strategies (Ch'ng et al., 2019), only a few research studies have addressed supporting developing CT skills using WEs. For instance, Grover (2017) analyzed a curriculum design in Foundations for Advancing Computational Thinking (FACT), "a middle school introductory computing curriculum" (p.269). As per the study, one technique for supporting learners was to use Khan Academy short tutorial videos followed by open-ended programming tasks. Furthermore, the researcher claimed that the tutorial videos were influenced by the effective utilization of WEs to minimize cognitive load, particularly for novice programmers (Grover, 2017).

Another example of integrating technology with teaching material is the Basogain et al. (2016) study, where researchers employed a series of video tutorials as a support system to allow learners to access the programming course and learn more about the subjects.

Whereas the studies mentioned above showed how to utilize WEs successfully via different modes, several aspects impact the effectiveness of WEs as an educational method (Hoogerheide & Roelle, 2020).

Factors Affecting the Effectiveness of Worked Examples

Worked examples demonstrate that they are extremely helpful for skill acquisition and problem-solving tasks. Meanwhile, the efficacy of using worked examples in problem solving may be influenced by several factors (Neelen & Kirschner, 2021). Atkinson et al. (2000) provided a framework consists of three categories of the important factors affecting the worked examples design as:

- Intra-example features are concerned with the presence of designing an example and the presentation of the step-by-step solution (Atkinson et al., 2000).
- Inter-example features concerned with the precise connections between many examples and problems practiced within a single lesson (Atkinson et al., 2000).
- Environmental/ situational features are concerned about how learners are processing examples (Atkinson et al., 2000).

It also showed that integrated various types of elements of worked examples, text, visuals, aural information, and subgoals are all considered key to the efficient design of worked examples (Atkinson et al., 2000; Spanjers et al., 2012). However, there are other aspects that should be considered when thinking about employing worked examples, those aspects concerned more on the "how" side of the developing process.

The Instructional Design of Worked Examples

Worked examples, also known as worked-out examples, have steadily evolved as an essential instructional strategy that has been supported by the field of instructional design over the past 30 years (Atkinson et al., 2000). While learning through examples is preferable to problem solving for inexperienced learners, it is necessary to pay attention to the method through which WEs are presented and developed (Kalyuga et al., 2003). As a result, pioneers in

instructional design field created a diverse set of guidelines for developing compelling working examples.

However, in the programming domain and specifically for promoting CT skills, there is little guidance on how to develop effective support and guidance activity (Caskurlu et al., 2021). Accordingly, this study aims to figure out how the WEs should be designed, developed, and presented if they are used to promote CT skills and via well-structured programming problems. Furthermore, instructional design incorporates various principles for designing and developing WEs that have been studied in a variety of fields, with only a limited investigation in computer science in general (Beege et al., 2021). These instructional design principles cover a wide range of functions and aims. Clark and Mayer (2016), for example, produced a set of guidelines to help designers create instructions, particularly for supporting e-learning material, including the development of the following instructional design principles for worked examples.

- Principle 1: fade from worked examples to problems. The offered examples of fading will begin with a detailed description of the technique, followed by many examples that fade the procedure stages and require the learner to complete the missing steps (Clark & Mayer, 2016)
- Principle 2: promote self-explanations. Self-explanation is an instructional method in which learners are prompted to explain to themselves the provided procedures as a reflection of their understanding (Clark & Mayer, 2016).
- Principle 3: include instructional explanations of worked examples in some situations. It entails integrating worked examples with instructional explanations. In e-learning, for example, a "help" button could provide more precise explanations or justification for the steps shown in the worked example (Clark & Mayer, 2016, p.234).

- Principle 4: apply the multimedia principles to examples that include the effective use of the graphics, text, and audio in the content (Clark & Mayer, 2016).
- Principle 5: support learning transfer implies ensuring that learners adopt the same experience to a new context.

In addition to Clark and Mayer (2016) instructional design principles for worked examples, Atkinson et al. (2000) suggested some instructional principles for the design of WEs that include:

- Providing multiple representations of WEs.
- Providing subgoals.
- Providing multiple problem types within WEs.
- Providing self-explanation.

Renkl (2014a) has suggested a similar list of principles as the following:

- Explanation-help principle which indicates that learners who are having difficulty will receive assistance in the form of an instructional explanation (Renkl, 2014a).
- Self-explanation principle and Comparison principle which indicates that learners are expected to start comparing examples and to explain the reasoning to themselves (Renkl, 2014a).
- Imagery principle which indicates that learners are required to visualize the steps in the solution of previously covered examples (Renkl, 2014a).
- Easy-mapping principle which indicates that learners are required to visualize the steps in the solution of previously covered examples
- Focus-on-learning-domain principle which indicates that “learners' attention is not “bound” to the exemplifying domain“ (Renkl, 2014, p. 402).

- Instructions-for-use principle which indicates that learners are shown how to use various representations and information sources (Renkl, 2014a).
- Fading principle which indicates that problem-solving steps are presented in a fading mode (Renkl, 2014a).
- Example-set principle which indicates that learners are provided examples that highlight crucial and critical elements (Renkl, 2014a).
- Studying-errors principle which indicates that learners are provided with correct and incorrect worked examples (Renkl, 2014a).
- Meaningful-building-blocks principle which indicates that by segmenting a procedure, the essential solution phases become more visible (Renkl, 2014a).

In addition, there were numerous design effects to consider in addition to the instructional design principles of worked examples. For instance, split-attention effect which indicates that learners must integrate diverse information sources in order to cognitively integrate them without incurring excessive unnecessary load (Sweller et al., 2011). Also, there is a redundancy effect that deals with various sources of information, and it means that any information that is neither required nor relevant to learning should be eliminated (Sweller et al., 2011).

Moreover, the variability effect is accomplished by employing examples with varying contexts to control the intrinsic load and improve the learning transfer (Clark et al., 2011).

Furthermore, the preceding discussion leads to the search for relevant instructional design principles for designing WEs that may improve CT skills while also being used for well-structured programming problems, considering the different design and implementation considerations. The goal of this study is to produce a comprehensive understanding of designing

and developing WEs that is based on instructional design principles. The next chapter will go through the research method used in the study.

Chapter Three

Research Methodology

The third chapter aims to provide an introduction of the research methodology, present the rationale of the study, and demonstrate how the study was conducted.

The purpose of this research was to conduct an integrative review related to the instructional design of worked examples for promoting computational thinking skills in well-structured programming problems. This integrative review is anticipated to assist instructional designers and educators in developing worked examples that are appropriate for promoting computational thinking skills within programming tasks. The integrative review may assist instructional designers and educators through the integration of theory and research to recognize the instructional principles in designing effective worked examples.

Study Design

This study employed the integrative review (IR) as a research methodology. Integrative review IR is defined as the process of analyzing the previous empirical or theoretical literature to gain a better understanding of a specific problem or phenomenon (Broome, 2000). IR is considered as "a form of research that reviews, critiques, and synthesizes representative literature on a topic in an integrated way such that new frameworks and perspectives on the topic are generated" (Torraco, 2005, p.356).

According to Cooper (1982), IR adopts a comprehensive approach and analyzes a wide range of sources, including empirical and theoretical literature, or both. IR composes and synthesizes (1) empirical studies; include quantitative and/or qualitative studies on a specific area, (2) methodological reviews; include designs and methodologies of various studies, and (3) theoretical reviews; includes review of theories on a specific area (Toronto and Remington,

2020; Soares et al. 2014; Whittemore et al. 2014). The current study included quantitative studies, mixed methods studies, systematic reviews and theoretical reviews.

Integrative Review Background and Purpose

The IR method has a long history, going back to the 1980s when it became popular in several fields such as education, psychology, and nursing (Cooper, 1982; Ganong,1987; Jackson, 1980; Toronto and Remington, 2020).

The term "*integrative review"* has been associated with similar terms with different purposes such as "*narrative review*" and "*systematic review*" (Toronto and Remington, 2020). In addition, Sutton et al. (2019) distinguished 48 different types of reviews and classified them into seven groups: "traditional reviews, systematic reviews, review of reviews, rapid reviews, qualitative reviews, mixed method reviews and purpose specific reviews" (p. 204). According to Oermann and Knafl (2021) one of those categories is the traditional reviews that include:

- Narrative literature reviews: non-systematic reviews that aim to find and synthesize what has been published, eliminate duplication of investigations, and look for new areas to investigate (Ferrari, 2015).
- Critical reviews: "review the literature and evaluate its quality" (Oermann & Knafl, 2021, p.65).
- Integrative reviews.
- Narrative summaries: "provide an overview of available evidence on a single topic, often produced within a short timeframe" (Oermann & Knafl, 2021, p.65).
- State-Of-The-Art review; "considers mainly the most current research in a given area or concerning a given topic" (Dochy, 2006, p.5).

Overall, all of these reviews have specific purposes and characteristics that distinguish them from each other. The appropriate review to conduct is determined by the topic and the interested research area, as well as the intended audience (Oermann et al., 2018).

The purpose of integrative review is to gain a comprehensive understanding of a specific topic (Toronto, 2020). For this reason, IR utilizes diverse sources to gain information that include empirical, theoretical, and several studies with multiple search designs (Toronto, 2020; Whittemore & Knafl, 2005). To have more clarification, one of the distinguishing features of IR is its ability to provide a holistic understanding of a topic (Hopia et al., 2016). It incorporates both empirical and theoretical literature as well as research with a variety of designs, as opposed to systematic reviews and qualitative synthesis, which are limited to include empirical studies with a particular study design (Oermann & Knafl, 2021).

Benefits of Integrative Review

The current study applied IR as a research method. IR has a distinctive feature, it allows the researcher to identify: (1) the current condition of evidence for a specific phenomenon, (2) the quality of the research evidence, (3) “gaps in the literature” (Toronto, 2020, p.4), and (4) prospective research practices (Russell, 2005). IR method aids in the retention of a current knowledge base in a specific research area (Cooper, 1998). As it stated previously, IR is defined when “past research is summarized by drawing overall conclusions from many studies” (Broome, 2000, p.47).

Following a systematic approach as steps to conduct this review, "a well-prepared integrative review can precisely represent the state of the current research literature" (Russell, 2005, p.8). Also, IR can be used to assess the strength of empirical studies, define gaps in the existing research area, define the necessity of additional investigation, connect several research

areas, recognize core issues in research, formulate a research question, and investigate which research approaches have been used effectively (Cooper, 1998; Russell, 2005).

IR can be used to summarize previously recognized and reviewed studies for a specific area, as well as to draw assumptions about a subject, and to determine suitable practices and future directions for a specific research area (Cooper, 1988; Toronto, 2020; Whittemore & Knafl, 2005). IRs are "useful when an individual is beginning to build knowledge about concepts", and they are expected to make a substantial impact and contribute to a better understanding of research's future directions (Broome, 2000, p.249).

Rationale for The Current Study

For the current study, IR is needed to understand and recognize the current instructional design principles in designing and developing effective worked examples that are appropriate for programming problems. It is a significant technique that uncovered the rich details about the instructional design of worked examples in well-structured programming problems, and it looked at how these principles have promoted computational thinking skills as they are involved in the programming tasks. Additionally, IR is anticipated to build a new knowledge of the most suitable worked examples designs that are built for promoting computational thinking skills. IR would help to provoke the following research questions:

1. Based on the literature, how have worked examples (involving computational thinking skills) been designed?
2. What instructional design principles have been recognized in the literature and need to be considered when designing WEs to promote CT skills?
3. What factors or circumstances might have an impact on the development of WEs for well-structured programming problems?

4. What are some examples of effective WEs for well-structured programming problems that have been identified in the literature?

IR addressed the following areas: instructional design principles within worked examples, and programming domain within computational thinking skills. The first category is instructional design of worked examples, which is a broad category with multiple subcategories, with understandings connected to the instructional design principles for the worked examples as being the primary focus. The second category includes applications of WEs in the programming domain in combination with CT skills.

Steps of Conducting Integrative Review

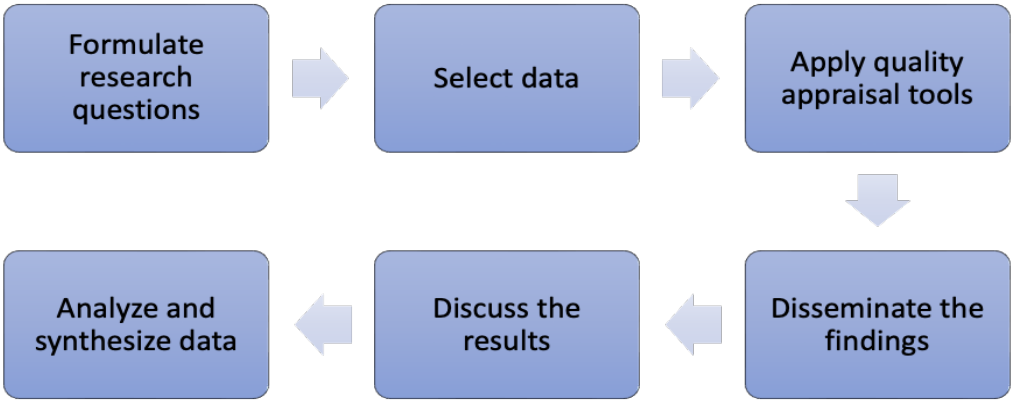
The IR followed a "systematic approach that is transparent and rigorous" (Toronto & Remington, 2020, p.5). There are various frameworks that have been used in conducting IR, like, Souza et al. (2010) six stages process; and Whittemore and Knafl (2005) five stage. All of these frameworks have been built based on Cooper (1984) five stages as the following: (1) problem identification; (2) data collection or literature research; (3) data evaluation; (4) data analysis; (5) results presentation (Russell, 2005).

The IR followed Toronto and Remington (2020) steps in conducting data, and it is also based on Cooper (1984) five stages. Toronto and Remington (2020) six steps are: “1. formulate purpose and/or review question(s) 2. systematically search and select literature 3. quality appraisal 4. analysis and synthesis 5. discussion and conclusion 6. dissemination” (p.5).

The IR followed Toronto and Remington (2020) six steps as it is shown in the next figure (Figure 1).

Figure 1

The six steps of the IR process based on Toronto and Remington (2020) framework.



The following is a brief overview of the steps:

1- formulate purpose and/or review question(s):

The first stage of a review implies that IR is addressing a clearly identified problem (Oermann & Knafl, 2021). According to Torraco (2005), it is essential to highlight the significance of the problem and demonstrate the reasons for choosing integrative review as a methodology that might address this problem. IR begins with Identifying the problems includes identifying a "topic or concept of interest" (Broome, 2000; Toronto & Remington, 2020, p.14).

In addition to identifying the problem, the research needs to identify the conceptual and operational definitions of the variables (Whittemore & Knafl, 2005). It is critical to eliminate any vagueness in the IR by defining the variables and stating how to employ in the integrative review (Russell, 2005). The conceptual definition defines the concept, while the operational definition explains what the concept implies that can be observed and measured (Toronto & Remington, 2020). The main variables of the current study were worked examples, computational thinking,

and well-structured problems; all of these variables have been discussed in the second chapter. Recognizing and understanding the meaning of those variables helped to conduct IR effectively.

In conjunction with identifying the problem, a researcher's research interest should be generated from their curiosity, as they will be spending a significant amount of time researching a particular topic (Beyea & Nichll, 1998). A researcher's interest to conduct the current study derived from the experience background area, in addition to the previous research about the difficulties that have been experienced by novice programmers in studying programming courses and gaining computational thinking skills. This researcher noticed that one key aspect affecting the programmer's progression in developing their computational thinking skills was the increasing cognitive load while completing programming tasks. According to (Moore et al., 2020) "computational thinking requires high cognitive load as students work to manage multiple tasks in their problem-solving environment "(p.19).

There is a substantial risk of increasing the cognitive load as learners gain those skills and solve tasks, which could impede them from meeting their learning objectives (Stachel et al., 2013). To explain, the programming task is regarded as a mechanism or a vehicle to promote computational thinking skills, in which learners engage in solving specific programming problems and employ computational steps to solve those problems (McVeigh-Murphy, 2019). According to Moon et al. (2020), "engaging learners in programming exercises not only improve their skills in a specific programming language, it could enhance their CT competencies" (p.2).

Although the goal of programming is to improve learners' computational thinking skills, there is a high possibility that it will affect and increase cognitive load (Krugel & Hubwieser, 2017; Weese et al., 2016). Yet, if the cognitive load is increased, it may impact their performance and prevent them from meeting their learning objectives.

On the other hand, instructional design provides strategies that help to reduce the cognitive load, and worked-examples are considered as an effective way to meet this purpose (Paas & Van Gog, 2006). Worked examples have been recognized for 30 years as an emerging instructional design technique and it has been used as a strategy in many domains such as mathematics and physics (Atkinson et al., 2000; Spanjers et al., 2012).

Accordingly, efforts have been made to reduce the cognitive load and facilitate learning from problems. Instructional design scholars and researchers have investigated worked examples and developed principles in designing worked examples (Atkinson et al., 2000; Chi et al., 1989; Paas et al., 2003; Sweller, 1988; Sweller, 2002; Ward & Sweller, 1990).

Moreover, developing CT skills began as a fundamental skill to be learned throughout the school curriculum and in a variety of disciplines (Maharani et al., 2019). As CT skills have grown in popularity and are regarded as essential skills required to compete in today's global economy, learners still face difficulties in acquiring those skills, and it is identified as a difficult obstacle (Bower et al., 2017; Grover & Pea, 2013; Lye & Koh, 2014). It is a challenging task since it involves a lot of mental effort, such as following the explanation, thinking about the solution, and solving the programming problem (Morrison et al., 2015). Worked examples in various designs like labeled sub-goals were developed to support and guide learners while they are performing programming tasks and developing their computational thinking skills (Caspersen & Nowack, 2013; Mannila et al., 2014; Weese et al., 2016; Weese, 2017). In addition, interactive worked examples (Webb & Rosson, 2013) and short video examples (Grover, 2017) were considered as additional types of worked examples that have been investigated to promote CT skills in the programming domain.

Overall, with all these attempts, there is a call for providing a deeper understanding on how to support and guide promoting CT skills in programming context. Sanford and Naidu (2016) stressed the importance of providing appropriate support and guidance in order to promote CT skills in a way that meets learners' needs. Despite the importance of CT skills being recognized, the resources that guide promoting and developing them are very limited (Li et al., 2020). Numerous pre-college teachers struggle with planning and implementing specific instructional activities to support and teach CT skills (Denning et al., 2017; Hu, 2011).
It is anticipated that it is beneficial to combine all the resources that discuss the design of worked examples and how they might be employed in specific settings such as programming context.

This study followed IR methodology as a comprehensive method that may help in identifying all the design principles, examples, experiences, and theoretical discussions that are related to the design of worked examples, programming problems and computational thinking skills.

2. Systematically search for literature and select data

The search process for IR indicates "defining in detail all databases, search terms, limiters, eligibility (inclusion/exclusion)" (Toronto & Remington, 2020, p. 22). It also indicates defining the selection criteria and any additional research tools that have been used to provide a clearly documented and comprehensive literature review (Cooper 1982; Whittemore & Knafl, 2005).

Comprehensive IR research seeks to collect as much relevant literature as possible, and researchers should identify some considerations to improve the study's rigor (Evans, 2007; Whittemore, 2007). The comprehensive searching may be associated with employing multiple research strategies like search within several databases and manual searching in journals and

different references (Whittemore & Knafl, 2005). Inability to employ several searching strategies may lead to inadequate or biased search results, affecting the accuracy of the included sources in the collected data (Cooper, 1998).

For this purpose, the current study applied two complementary approaches advocated by Whittemore and Knafl (2005). First, the study collected computerized data by searching several electronic databases. Then, the study used the collected literature's references to conduct a manual screening. The manual screening approach is known as the ancestry approach, and it is useful to ensure that the most extensive body of literature relevant to the topic is collected (Atkinson et al., 2015). Cooper (2017) defines the ancestry approach or backward search as "using the reference lists at the end of research reports to locate other reports that might be relevant to a search" (p. 14).

Using the search methods necessitated searching through a range of databases; this study used the following databases:

- Scopus, (extensive database includes scientific research and social sciences research).
- Web of Science, (international multidisciplinary database).
- ERIC, (educational research electronic library).
- EBSCO, (educational database).
- JSTOR, (provides access to academic journal articles).
- ProQuest, (provides access to educational journals and dissertations).

All of the above databases are accessed online through Virginia Tech's university library. Following, the study used specific search terms as key terms to conduct the data. For key terms, the Boolean operators *AND, OR, NOT,* are employed to expand the search results (Toronto & Remington, 2020). The study used several key terms, for instance: "*worked examples"* AND

"*Design"* AND "*computational thinking"* OR "*programming", "worked examples"* AND *"computational thinking"* OR *"programming", "Worked examples"* AND *"instructional design"* AND *"programming", "worked examples"* AND *"instructional design"* AND *"well-structured problems".*

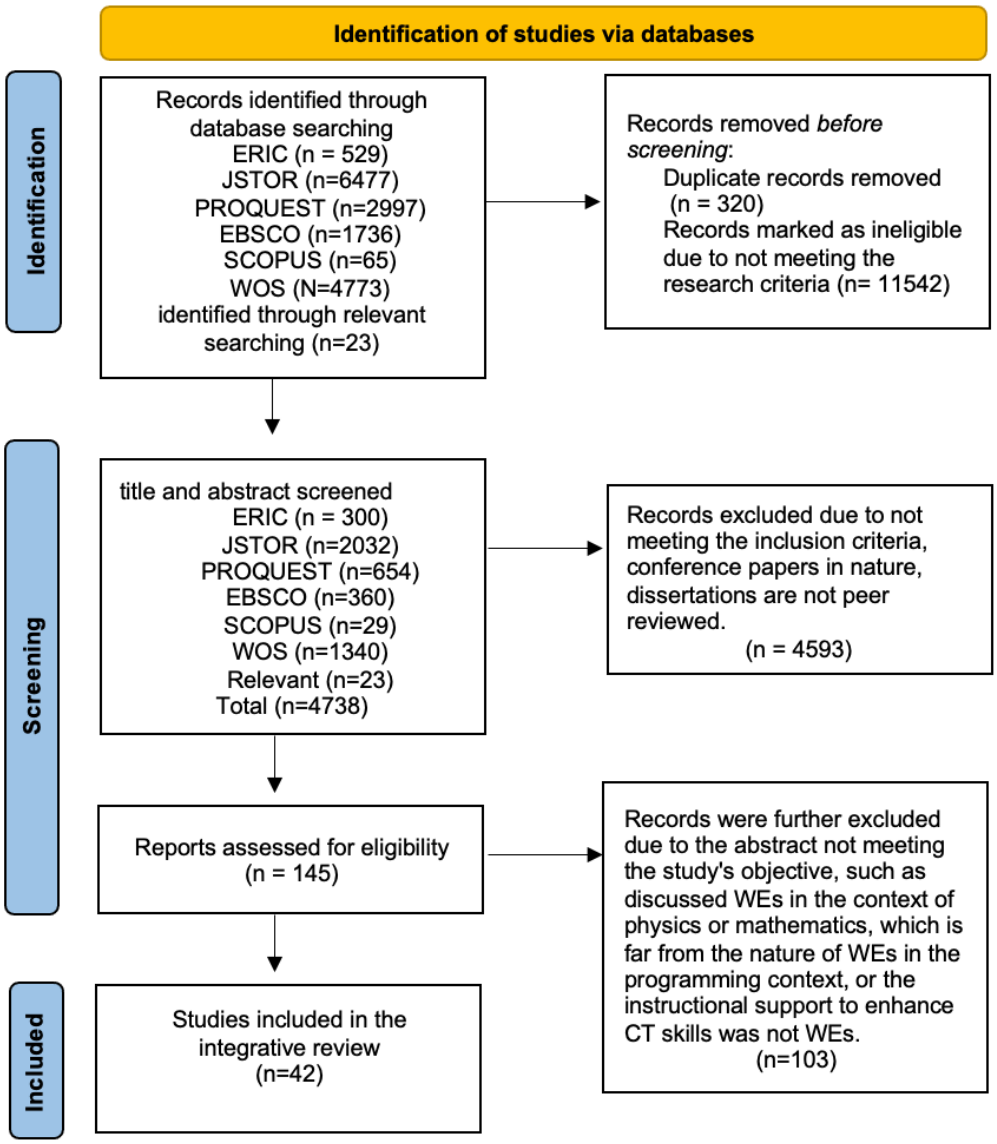
Furthermore, "IRs are characteristically broad in nature", to refine the results more adequately, inclusion and exclusion criteria must be defined (Toronto & Remington, 2020, p.31). The inclusion criteria include qualitative, quantitative research and mixed method, theoretical frameworks and discussion, meta-analysis, systematic reviews, and scoping review studies, peer-reviewed, English language, published in academic journals, and published between 2000 and 2022. All of the collected data focused on the instructional design of worked examples, programming context and computational thinking skills. Further, the exclusion criteria include data that was published before 2000, not peer-reviewed, editorials, abstracts, government reports, gray literature (reports other than peer-reviewed articles), and conference proceedings.

The study employed various search filters (such as peer review, full text, publication date, language, and type of publication) to narrow down the database's results to relevant articles. It also employed field filtering, which included the inclusion of articles about programming context, computer education, and instructional design, in order to achieve the goals of the study.

Furthermore, using inclusion criteria, the initial screening encloses screening titles and abstracts of the relevant studies (Toronto & Remington, 2020). The remaining literature is next assessed to decide whether it should be included. The study used a search flow diagram to document the procedure and the reasons for the exclusion. PRISMA, a search flow diagram, is used for reporting and documenting the selection literature (Page et al., 2021) (Figure 2).

Figure 2

An overview of the data collection results based on the PRISMA flow diagram.



PRISMA flow diagram (adapted from Page et al. (2021))

The data collection method produced a total of 16,600 papers; however, during the identification phase, approximately 11,862 research were eliminated from the further review due to duplication or not meeting the inclusion criteria. About 4593 articles were excluded from the analysis after the titles and abstracts were screened because they were either not peer-reviewed

dissertations or conference papers. After that, 103 records were disregarded for various reasons, such as 1) the abstract examines the worked examples in other contexts, such as physics and mathematics, and 2) the instructional support to improve computational thinking skills is not worked examples. Ultimately, a total of 42 articles complied with all the elements of the search criteria and were used to conduct the review.

In addition, as a procedure for following IR stages and carefully selecting data, the study used EndNote and Excel to collect, organize, and cite records.

3. Quality Appraisal

The quality of studies included in a review determines the strength of its findings (Coughlan & Cronin, 2016). According to Denney and Tewksbury (2013), it is critical to analyze the internal validity of the selected literature to ensure validity when conducting IR. It is crucial to keep in mind that the quality of published papers varies (Toronto & Remington, 2020).

Incorporating low-quality papers into the review could influence the results at the same time; eliminating low-quality studies could bias the study (Evans, 2007). As a result, after the appraisal process, the inclusion and exclusion criteria should determine whether the more inferior studies will be included or excluded (Toronto & Remington, 2020).

Despite the fact that there is no specific way and no universal agreement to evaluate the quality of the selected studies, the literature suggests that any study should be rigorously evaluated (Katrak et al., 2004). Several appraisal tools could be utilized to appraise the quality of the research, and they vary between open questions, closed questions, or statements (Crowe & Sheppar, 2011; Sanderson et al., 2007).

To ensure the rigor of the current research, the study followed several appraisal tools based on the articles' methodologies. The study followed the *Joanna Briggs Institute (JBI) via*

https://jbi.global/critical-appraisal-tools. JBI's critical appraisal tools help to assess the validity, relevance, and results of the published papers, and it indicates several checklists as to appraise tools for randomized controlled trial, Cohort study, Case- controlled study, and Qualitative study. In this study, JBI's critical appraisal tools were used to evaluate Quasi-experimental studies and systematic review. The *JBI* Quasi-experimental checklist includes 9 items with 4 possible responses for each as (Yes, No, Unclear, Not applicable) with indicating (include, not include, seek further info) for overall appraisal (see Appendix F.A). The *JBI* systematic review checklist includes 11 items with 4 possible responses for each as (Yes, No, Unclear, Not applicable) with indicating (include, not include, seek further info) for overall appraisal (see Appendix F.C).

The Mixed Methods Appraisal Tool (MMAT) for mixed methods studies section used to evaluate the mixed methods studies, and it includes two sections: the first section involves two screening questions about the studies, and the second section involves 5 items with 4 possible responses for each as (Yes, No, Can't tell, Comments) (see Appendix F.D).

Furthermore, because IR encompasses a wide range of data, not just qualitative or quantitative investigations, but also theoretical discussions were evaluated (see Appendix F.B).

For this purpose, the study followed Walker and Avant (2019) six steps procedure to guide in evaluating the theoretical discussions:

1. the purpose of the theory.
2. the concepts, statements, and assumptions of theory.
3. the structure, "the ability of the theory to make accurate predictions is examined" (Toronto & Remington, 2020, p.51).
4. theory's usefulness.
5. the degree of generalization.

6. ” Testability refers to how well the theory can be supported by empirical data” (Toronto & Remington, 2020, p.51).

4. Analysis and Synthesis

The purpose of an IR is to improve the knowledge of a specific subject by combining and synthesizing data from several sources (Whittemore & Knafl, 2005). Torraco (2016) stated that generating a new perspective on the topic of interest required IR to analyze and synthesize the incorporation of a large number of existing data. According to Torraco (2016) “synthesis brings together existing ideas with new ideas to create fresh, new ways of thinking about the topic" (p.66).

For that, Toronto and Remington (2020) recommended analyzing the similarities, differences, and recommendations from the extracted data by utilizing data matrix as tables and patterns. The data matrix as tables includes rows and columns that are used to synthesize data from the extracted articles (Tomasic, 2011). This study used Excel spreadsheets as a data matrix to help abstract the extracted data. Examples of the suggested information to be included in the table may include the authors’ name, year, method, sample, quality rating, and study results (Coughlin & Sethares, 2017).

There are no defined procedures for analyzing IR data; nonetheless, constant comparison, content analysis, and thematic analysis are suggested methods that could be utilized to analyze data (Hopia et al., 2016). Whittemore and Knafl (2005) described constant comparison technique as a systematic analytical strategy that researchers might use throughout the data analysis stage. The study employed a comparison method to analyze data. This method includes four steps: data reduction, data display, data comparison, and conclusion drawing and verification (Whittemore & Knafl, 2005). Each step will be described as the following:

Data reduction means the method of choosing, organizing, summarizing, and extracting data from the resources (Toronto & Remington, 2020). It integrates the primary source of data in a manner that the review's findings may be inferred and validated (Miles & Huberman, 1994). The primary sources that are included in IR must be organized into subgroups in some systematic way, such as type of evidence, sample characteristics, and settings (Whittemore & Knafl, 2005).

After, techniques to extract and code data from primary sources should be identified "to simplify, abstract, focus, and organize data into a manageable framework" (Whittemore & Knafl, 2005, p.550). As a result, the primary source is focused on a single matrix, with data from each subgroup category derived from different sources (Miles & Huberman, 1994; Whittemore & Knafl, 2005). The current study classified the primary source of data into theoretical and empirical studies, then they were classified again based on common features into subcategories. In addition, an Excel table was created to summarize and abstract each article into a digestible unit of analysis with similar characteristics for further coding. At first, a table is created to involve all the general information about conducted articles, like 1) Article name, 2) Author(s), 3) Year, 4) Database and journal (see Appendix A, B, C, D, E).

Data display gives a more simplified presentation of the data and makes it easier to conclude (Miles & Huberman, 1994). The data presentation “can be in the form of matrices, graphs, charts, or networks” (Whittemore & Knafl, 2005, p.551). These visualizations assist in representing patterns and relationships across and between primary data sources and in interpreting data (Sandelowski, 1995). The study compiled all the abstracted data into tables for further data comparison and analysis.

Data comparison phase includes identifying patterns, themes, or relationships among the displayed data (Toronto & Remington, 2020; Whittemore & Knafl, 2005). To enhance data comparison, the researcher employed several strategies like “clustering, counting, and making contrasts and comparisons" (Toronto & Remington, 2020, p.65). In the current study, four tables have been created and classified based on each research question, and each table indicates subclassifications or sub-themes regarding the main classification. For instance, the first theme is (Worked examples designs) and from this theme 5 subcategories have been identified as 1) animated WEs, 2) Process-oriented, 3) Subgoal, 4) fading, 5) Sequencing, 6) Tutorials, 7) in-game WEs. The second theme, instructional design principles, involves 1) multimedia, 2) split-attention, 3) completion, 4) self-explanation, 5) fading principle. For the third theme (factors affect the design of WEs), several elements were classified as 1) time of instruction, 2) learners’ engagement, 3) learners’ background, 4) learners’ familiarity with the language, 5) inter, intra, and situational features. The fourth theme, successful examples of WEs, includes 3 main classifications as 1) medium, 2) supporting tools, and 3) techniques (see Appendix A, B, C, D, E)

Conclusion drawing and verification is the last phase of the data analysis, and its goal focuses on shifting the “interpretive effort from the description of patterns and relationships to higher levels of abstraction, subsuming the particulars into the general” (Whittemore & Knafl, 2005, p.551). The verification procedure entails going back to the sample sources to validate the results' accuracy, or it could include confirming the identified patterns, themes, and relationships (Toronto & Remington, 2020; Whittemore & Knafl, 2005). For this purpose, the articles were double-checked several times to avoid the risk of misunderstanding and verify consistency with the primary sources.

5.Discussion and Conclusion

The IR is a unique type of study in which current literature is used to generate a deep and new understanding of a specific topic (Torraco, 2016). These studies often include comparisons and contrasts to indicate findings dealing with background literature, recommendations, implications, and limitations (Toronto & Remington, 2020). The discussion section should explicitly state how the literature gap has been addressed and how the study's findings and conclusions contribute to the body of knowledge on the subject (Flanagan, 2018; Oermann & Hays, 2019). According to Coughlan and Cronin (2016), the discussion section begins by reminding readers of the questions that the review attempted to answer. Following, in summarizing IR findings, the researchers are allowed to express their opinions of the relevance and significance of the findings and indicate comparisons to the current information (Coughlan & Cronin, 2016).

In comparison to background literature, the researcher compared the synthesized theoretical literature in the introduction section and the analysis of the results revealed in the discussion (Souza et al., 2010). In addition, Toronto and Remington (2020) stated that if many research studies provide results that lead to identical findings, then "reviewers draw broader understanding of the phenomenon under investigation" (p.77). Afterwards, the researcher may provide their comments about any implications and recommendations for “research, practice, education, theory, and/or policy as appropriate”, however, this is not applied in all domains of research (Toronto & Remington, 2020, p.78).

In writing the limitations section, the researcher should provide information about IR limitations and not the individual studies limitations (Coughlan & Cronin, 2016). The conclusion section comes at the end of the discussion section, it includes the primary findings without

providing any discussion about new ideas and thoughts, and without indicating any citations (Watson, 2018). The following chapter, Chapter 4 presents the study's findings, and Chapter 5 discusses the conclusion and the limitations of the study.

6.Dissemination

Dissemination is the last step in the IR process, where the researcher aims to share the findings and present them to the intended audience (Sethares, 2020).Toronto and Remington (2020), refered this step to the ability to design a poster or presentation at a professional conference, or peer-reviewed, news, or social media.

Threats to Validity

Validity is a primary consideration in scientific research. When conducting an integrative review, "maintaining scientific integrity" necessitates paying attention to validity threats (Russell, 2005, p.3). Several approaches have been identified to verify the validity; however, several factors might threaten the research and reduce its efficiency (Yu & Ohlund, 2010). Bias can threaten the validity of specific research findings and result in a biased IR, leading to an exaggeration of the study (Sanderson et al., 2007). Aside from bias, a lack of the selected study may also represent a threat; consequently, paying attention to the quality of the study is a must (Whittemore & Knafl, 2005). Cooper (1982) recommended employing rigorous and transparent approaches at each stage of the IR process.

For instance, Cooper (1982) suggested that the operational definitions of the study variables should not be defined too precisely. However, it should not be defined too broadly; instead, it should be balanced in establishing conceptual and operational definitions while paying attention to the review procedures to reduce the threat's risk (Cooper, 1982).

Regarding the literature systematic search and data collection, inadequate literature sampling may threaten the validity of the study (Russell, 2005). As a result, the researcher must specify the inclusion and exclusion criteria and explain why they were excluded from the study (Russell, 2005). Furthermore, Cooper (1998), recommended the following strategies to enhance the validity in this step, (1) conduct an exhaustive data collection; (2) outline sources, years, and keywords; (3) present all selection biases; and (4) summarize demographics of the subjects (if it is included in the samples). For the current study, several key terms, several sources and two complementary approaches for data collection have been identified for this purpose.

To enhance objectivity in the data evaluation step and as it has been shown previously in the Quality Appraisal section, the current study applied *Joanna Briggs Institute (JBI) via https://jbi.global/critical-appraisal-tools***.** JBI's critical appraisal tools help to assess the validity, relevance, and results of the published papers with Quasi-experimental studies and systematic review. In addition to the MMAT tool to evaluate the included mixed method studies, and Walker and Avant (2019) framework to evaluate the included theoretical discussions.

Regarding data analysis threats, the first threat is failing to apply proper inference guidance (Cooper, 1998). Another threat in this step is "inferring causality that is inappropriate when examining research review data" (Russell, 2005, p.5). Cooper (1998) recommended the following methods to maintain the validity:

1. Make assumptions clear when explaining results and inferences.
2. Vital interpretation guidelines should be identified.
3. Evidence that is based on a single study should be completely separated.

The current study paid attention to minimize any misinterpretation by recognizing and differentiating any direct study-based from the indirect studies. At the data interpretation and

presentation step, several threats may occur, like, omitting important details about conducting the IR, and omitting details about the study findings that might be important to other investigators (Copper, 1998). One suggestion for reducing these threats is to pay close attention to all of the study's possible details (Russell, 2005). Regarding the study, the gathered articles were double-checked numerous times to minimize misunderstandings and to ensure that all relevant information was included.

To sum up, the current study aims to provide detailed information in an explicit and clear manner. The extracted comprehensive information aims to guide instructors and instructional designers in recognizing the several approaches and principles in designing and developing WEs to promote CT skills in well-structured programming problems.

Chapter Four

Findings and Discussion

This chapter presents the outcomes of the integrative literature review following iterative comparisons of patterns and distinctions among all of the studies that were considered. This study's findings identified and categorized numerous design forms of worked examples in programming contexts. Furthermore, the data showed several instructional design principles applied to producing worked examples to improve computational thinking skills. It also highlighted elements that may impact the use of effective worked examples, as well as some successful examples of their use. The following are the answers to each of the questions.
This study aims to address the following questions:

1. Based on the literature, how have worked examples (involving computational thinking skills) been designed?
2. What instructional design principles have been recognized in the literature and need to be considered when designing WEs to promote CT skills?
3. What factors or circumstances might have an impact on the development of WEs for well-structured programming problems?
4. What are some examples of effective WEs for well-structured programming problems that have been identified in the literature?

The study developed a theme to answer the research questions; this technique assisted in organizing data collection and adhering to the requirements of conducting the integrative review more clearly and accurately.

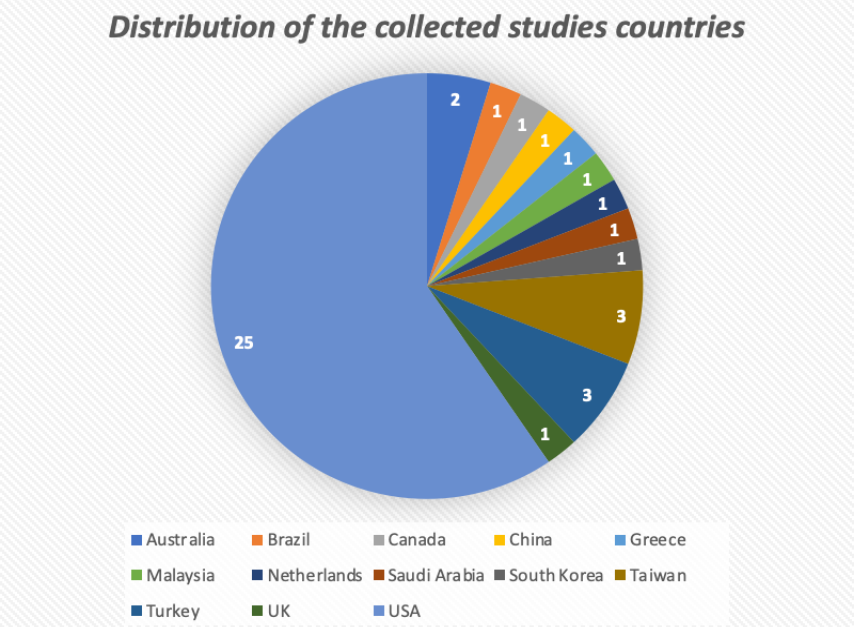
The IR final analysis revealed 42 studies to be included in the review and to answer the research's questions. Between the years 2000 and 2022, these studies were conducted in a variety

of countries with different research methodologies. According to data from this study, the instructional design of worked examples in well-structured programming problems has mainly been studied in the United States with 25 studies, followed by Turkey and Taiwan with three studies, and one study for each of the UK, South Korea, Saudi Arabia, Netherlands, Australia, Brazil, Canada, China, Greece, and Malaysia.

The next figure (Figure 3) shows the distribution of these studies based on countries.

Figure 3

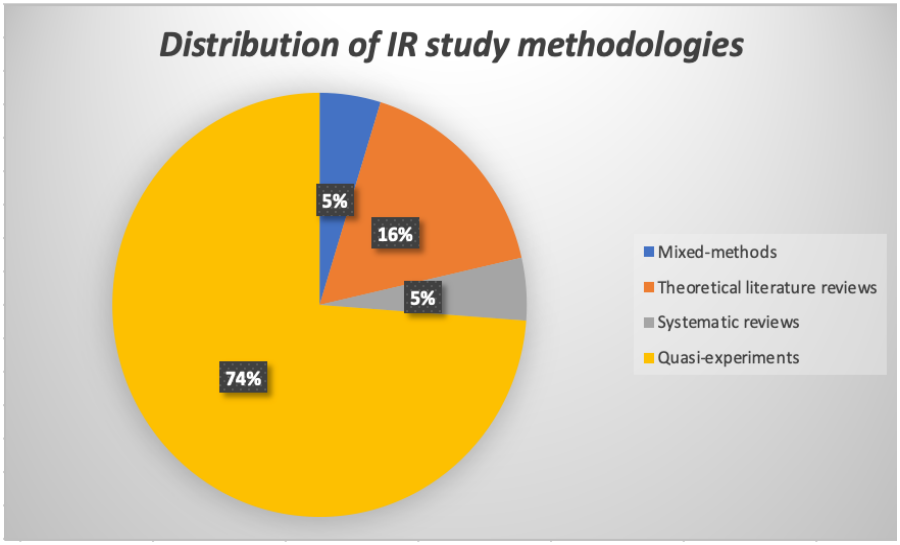
Distribution of the collected studies countries.



In addition, several research methodologies have been used to explore the worked examples in programming settings, including quasi-experiment as the most employed methodologies, and followed by theoretical literature reviews, systematic reviews, and mixed methods studies. The next figure (Figure 4) demonstrates the percentage of the methodologies that have been employed in these studies.

Figure 4

Distribution of IR study methodologies.



Findings of Research Question 1

The first research question "*Based on the literature, how have worked examples (involving computational thinking skills) been designed?*", is seeking information about the several designs of worked examples in programming settings. Approximately 72 % of the gathered studies examined several worked examples designs that have been employed to enhance computational thinking skills in a programming setting. The study categorized the results of this research question into seven categories as the following:

- animated worked examples.
- process-oriented worked examples.
- sequenced worked examples.
- subgoal design.
- tutorial as worked examples.

- faded worked examples.
- in-game worked examples.

Animated worked examples.

The animated worked examples were particularly beneficial for developing learners' complex cognitive skills, specifically when learners are dealing with programmed instructions (Chang et al., 2011). They "were primarily useful for training complicated cognitive skills to learners" (Hsu et al., 2012, p.164). Particularly, when learners are encouraged to learn new programs and required to solve programmed problems with automating them via computers (Hosseini et al., 2020).

In computer science education settings, understanding program dynamics, such as how codes operate in a machine language, is one of the main hurdles in learning computer programming (Cevahir et al., 2022). Learners are more likely to observe the exemplified codes running as a means to enhance their computational skills rather than following static worked examples to understand the algorithm (Hosseini et al., 2020). The animated codes or codes with animations promote the program's dynamic as it is executed and assist in comprehensively understanding the programming process (Sorva et al., 2013). With animated elements, a worked example illustrates and visualizes how the algorithm is built rather than merely providing the line-by-line code (Hosseini et al., 2020).

Animations are visual representations of dynamic concepts that assist learners in understanding, for instance, events that change over time (Paas et al., 2007). Offering animated instructions has various benefits, including 1) making the learning process appealing, 2) increasing motivation for learning, 3) fostering learners' engagement, and 4) fueling learners' imagination (Weiss et al., 2002). Combining two instructional techniques like designing

animations and implementing them with worked examples is one way to promote computational thinking skills and programming skills.

Studies showed that animated worked examples can take several designs and formats. For instance, examples with codes could be implemented to promote “program construction skills” where learners can understand how the algorithm has been implemented and built based on the code statement (Hosseini et al., 2020, p.300). Learners were able to observe the animated outcomes of certain programming tasks by using *Mastery Grid*, which is an interactive programming platform. By using this approach learners were able to interact with items on the screen by clicking on them and interacting with what is shown in the screen (Hosseini et al., 2020).

In Chang et al. (2011) study, an executable program was provided to the learners as a worked example of the programmed instructions. To explain, learners were provided with an executable program as a worked example of the necessary steps to solve the programmatic problems besides another instructional slide that showed a sequence of a step-by-step algorithm to solve the problem. The study revealed that learners would pay close attention to the animated worked examples and grasp programming principles and techniques (Chang et al., 2011). The idea of experiencing an executable program as a worked example in completing an algorithm is to gain the attention of the learners as much as possible simultaneously with instructions. Learners anticipated performing notably better when teaching with instructional slides and executable program examples than when studying with "static structured working examples” for solving programming problems (Chang et al., 2011, p.191).

Moreover, it has been demonstrated that integrating worked examples with animations in programming environments may also serve as instructional material. Researchers in the field

have investigated innovative design to implement effective worked examples that are suitable for fostering computational thinking skills. For instance, Cevahir et al. (2022) investigated the effect of combining animated based worked examples in Augmented Reality (AR) environment to promote programmed instructions. The study focused on comparing the learners' achievements and motivation toward this innovative technique. Learners in the experimental group were provided with animated worked examples as they programmed while the control group relied on using traditional worked examples via paper only. The results showed that learners in the experimental group had increased their motivation toward learning programming as well as their achievements score after the experiment (Cevahir et al., 2022).

In addition, IR analyzed a study with an innovative format of integrating worked examples via animated computational modules using "Virtual Kinetics of Materials Laboratory (VKML)" to promote abstract skill (Vieira et al., 2018, p.330). According to the study's experiment, VKML is based on the Python programming language, and it enables learners to "represent complex thermodynamics phenomena using modeling techniques and simple python programming" (Vieira et al., 2018, p.323). Researchers claimed that integrating worked examples with this computational model via VKML has a great influence in the learning process, learner's computational skills, and learners' programming practices (Vieira et al., 2018).

However, the collected studies recommended paying more attention to the instructional design aspect of those animated worked examples including following message design principles (Cevahir et al., 2022, Chang et al., 2011).

Process-oriented worked examples.

The IR found a design of worked examples that were employed in one study (n=1) to promote computational thinking skills in programming settings, and it is known as process-oriented worked examples. Process-oriented refers to the concept of how experts approach problems and the rationale for adopting a particular approach (Van Gog et al., 2004).

The traditional process-oriented working examples outline the steps necessary to solve the problem and explain why those actions should be taken to achieve the expected conclusion (Si et al., 2014). Explaining each solution step to learners may be beneficial, especially in the early phases of the learning process and problem-solving. In Si et al. (2014) study, the researchers employed the process-oriented worked example format in several stages. Learners are exposed to fully worked examples, traditional problem solving, and one subproblem to answer through interaction with four steps through four phases (Si et al., 2014). In addition, a mental effort scale was evaluated for the entirety of the learning session, and the results were shown by the time the session was completed (Si et al., 2014).

The design of Si et al. (2014) indicated process-oriented worked examples as series or worked examples following with rational reasoning of the problem-solving process. Si et al. (2014) claimed that providing process-oriented worked examples enable an effective schema construction and enhance cognitive skills.

The study found a mixed implementation of this particular design, which combined with numerous instructional design principles such as modality effect and multimedia effect; the response to the third study question will cover this topic.

Sequencing set of worked examples.

Providing sets of multiple examples to introduce programming instructions is one of the essential aspects of worked example design. Research showed that worked examples should be given as a sequence of at least two examples when used for complex domains like computer science (Nainan & Balakrishnan, 2019; Si et al., 2014).

In keeping with the idea of sets of worked examples, there might be different presentation types provided, such as 1) worked examples that cover numerous scenarios, and 2) complete-partial worked examples.

One format that has been analyzed in this IR was used specifically to promote data collection and data set skills in computational thinking is worked example sequencing scenarios. Worked examples offered in this format are a series of multiple scenarios with applying the same level of complexity and they are requiring the same techniques to solve a computational problem (Bunch, 2009).

In addition, a sequence of different quantities of examples could be provided in order to increase learners' skills in solving problems. Providing a sequence of completed worked examples and partial or incomplete worked examples is common in programming settings (Bunch, 2009; Nainan & Balakrishnan, 2019; Si et al., 2014). This could be formatted as a combination of different sequenced examples to increase the efficiency of learning programming.

For instance, Abdul-Rahman and Du Boulay (2014) claimed that two formats could be sequenced simultaneously, following the completion strategy (Van Merriënboer, 1990), and the structure-emphasis strategy (Quilici & Mayer, 2002). The completion strategy (Van Merriënboer, 1990) indicates providing a "well-designed" program by an expert and required

learners to “complete, modify, and extend it” (Chang et al., 2000, p.211).While the structure-emphasis strategy entails abstracting and organizing the underlying structural aspects of an example (structural similarity) to formulate a problem-solving model that aids in developing problem schemas and assists in completing a program (Quilici & Mayer, 2002, p.325).

Moreover, Chang et al. (2000) developed a template based on completion strategy to enhance programming skills. The research’s template consists of a programming problem database, code library (to retrieve any needed code statements), learner model, generator, and evaluator element (Chang et al., 2000). The study showed that learners in the experimental group performed better in programming tasks compared with the control group (Chang et al. 2000). Completion strategy within worked examples (part-complete worked examples) has been studied to decrease learners' cognitive load and to aid in the learners’ efficient programming learning (Abdul-Rahman & Du Boulay, 2014; Chang et al, 2000; Garner, 2007).

However, research showed that more considerations should be taken into account when deciding which part of the programming code might be left to the learner to be completed (Abdul-Rahman & Du Boulay, 2014; Chang et al, 2000). For instance, Garner (2007) conducted a study to explore the effect of part-complete worked examples in association with the scaffolding algorithm in the attached screen. The experiment showed remarkable results in enhancing learners’ programming skills in addition to recommending scaffolding techniques when teaching with part-complete worked examples.

Consequently, several instructional techniques were integrated with worked examples to enhance guiding learners on their way to solve and complete programming problems like hints and clues (Abdul-Rahman & Du Boulay, 2014; Chang et al, 2000; Garner, 2007). Furthermore, the studies that used the completion technique for worked examples found that the germane

cognitive load increased while the extraneous cognitive load decreased (Abdul-Rahman & Du Boulay, 2014; Chang et al, 2000; Garner, 2007).

Subgoal worked examples.

A subgoal signifies the goal of a series of steps. Learners often remember a linear series of steps when learning a technique in order to solve a problem or accomplish a goal (Margulieux et al., 2012). Subgoals "are structural parts of a problem-solving procedure, in which the overall goal is to solve the problem" (Margulieux & Catrambone, 2021, p.503). While worked examples demonstrate a step-by-step solution to solve a specific problem (Margulieux et al., 2018), subgoal labeled worked examples are utilized to improve the learning by capturing the attention of problem solutions' steps (Atkinson et al., 2003; Atkinson & Derry, 2000, Margulieux & Catrambone, 2016), and organizing the structure of the new information (Morrison et al., 2016).

According to various studies, when some learners receive new problems after studying worked examples, they do not follow the guidelines provided by the foremost worked example, instead, they follow the similar surface features shared with the new problems and the original worked example (Margulieux et al., 2020). Therefore, subgoal worked examples have been utilized to deliver instructions in procedural domains like computer science (Morrison et al., 2020).

The current IR investigation revealed that various researchers have studied the use of subgoal worked example designs to improve the computational thinking skills of programmers, particularly beginner programmers. Berssanette and de Francisco (2021) conducted a systematic review and discovered that the significant effect of subgoal worked examples implies decreasing the extraneous load of the learning process. According to the findings, utilizing subgoal labels assists inexperienced learners in learning how to program and establish a mental model during

the early phases of developing the algorithm and solving the programmed problem (Berssanette & de Francisco, 2021).

The design of subgoal worked examples can take many formats. According to Margulieux et al. (2016) study, the researchers used a computer-based environment to deliver the programmed instructions and investigated the effectiveness of subgoal worked examples. In the experiment learners followed subgoal worked examples that are represented in the computer-based environment and had the chance to request any additional assistance. Afterwards, learners were asked to use App Inventor for Android smartphones to develop an algorithm code design by utilizing drop-and-drag codes. According to the study's findings, employing numerous subgoal worked examples improves learners' performance when solving programming problems and completing codes (Margulieux et al., 2016).

Moreover, Margulieux and Catrambone (2021) discovered another design of subgoal worked examples in programming. The study aimed to assess the participants' abilities in solving programming problems following a specific scaffolding approach based on incorporating subgoal worked examples and self-explanation. To explain, in this design learners were provided with the problem statement and followed by subgoal worked examples to assist in solving the problem. Those subgoal worked examples have various features, such as, they were short instructions, and “context-independent explanations” (Margulieux & Catrambone, 2021, p.503). To have more clarification about this design, the worked examples were labeled based on chunking the problem statement into sub-problems, and those subproblems were the subgoals that the learners should pay attention to solve the problem effectively. It has been shown that including subgoals worked examples within drop-and-drag programming environments has a

positive influence on the learners' ability to effectively manage their cognitive load and, as a result, successfully solve the programming problems (Margulieux & Catrambone, 2021).

Furthermore, several suggestions have been made in order to decrease the burden of extraneous cognitive load and to ensure the effective utilization of subgoal worked examples. These suggestions include the indication of self-explanation, code comments, and scaffolding exercises for learners to complete while they are programming (Berssanette & de Francisco, 2021; Margulieux & Catrambone, 2021, Margulieux et al., 2016, Margulieux et al., 2020).

Worked examples as tutorials

In the context of the online environment, it has been shown that tutorials that extensively use worked examples as a mode of instruction are incredibly beneficial (Lee & Hong, 2016). Intelligent tutoring systems that are based on worked examples are claimed to provide well-supported learning and increase problem-solving, as stated by Renkl et al. (2009). These tutorials are acting as a model for the steps involved in problem-solving, and they are laying an emphasis on demonstrating and explaining the stages that are involved in problem-solving (Kale et al. 2018).

Studies have shown that it is practical to indicate tutorials and videos of the actual steps of solving problems and coding accurately to promote computational thinking skills such as decomposition, algorithm design, pattern recognition, abstraction, and analysis (Grover, 2017; Kale et al. 2018; Lee & Hong, 2016). Several initiatives have been established since these skills are crucial for computer programmers. As an example, Grover (2017) assessed the effectiveness of Foundations for Advancing Computational Thinking (FACT), a middle school introductory computing curriculum. The study found that the materials “comprised short Khan Academy-style videos ranging between 1 and 5 min in length that led learners through the thinking involved in

the construction of computational solutions using the Scratch programming environment" (Grover, 2017, p.273).

Also, learners are able to complete learning tasks more effectively and improve their computational thinking skills, particularly abstract, when step-by-step solutions to programming problems are demonstrated and modeled for them as a tutorial (Kale et al., 2018). For instance, a teacher utilized a video clip of the essential stages of accomplishing a programming algorithm in the Scratch environment (Kale et al., 2018). Accordingly, this allowed students to observe the processes and successfully drop-and-drag the blocks successfully (Kale et al., 2018).

Moreover, it is not a unique approach in the field of STEM education to incorporate tutorials within the context of the programming environment. Lee and Hong (2016) employed worked examples as a programming tutoring system to teach JavaScript, a different programming language. Participants in this study were given instructions on how to enter the virtual learning environment, where they would first be required to watch a video tutorial detailing a lesson, then complete a practice exercise, and finally complete the session with an assignment (Lee & Hong, 2016).

While Jocius et al. (2021) incorporated a scaffolding system into the Snap! Programming environment using the storyboard, this system included tutorials, visuals, and written worked examples.

Furthermore, Jennings and Muldner (2021) developed a tutoring system (*Code tracing CT)* in the form of worked examples to promote computational thinking skills. The *Code tracing CT;* has many advantages, like it:

> explicitly scaffold the process by requiring that students perform the code trace in a prescriptive, principled way, (2) provide assistance such as immediate feedback for

correctness, and (3) log student actions that can be subsequently mined to identify patterns of behaviors that are beneficial (or not) for learning (p.790).

The tutoring system *Code tracing CT* includes simulating a computer program in a step-by-step form, discovering any faults in the code, anticipating the output, and enhancing the learning process and engagement (Jennings & Muldner, 2021).

However, research indicated that some instructional design considerations should be addressed to assure the efficacy of developing tutorials as worked examples. It is important to note that worked examples that are not helping in reducing the extraneous cognitive load will not transfer any learning (Renkl et al., 2009). Therefore, instructional designers and instructors who are interested in developing effective and interactive worked examples-like tutorials- may follow several multimedia design principles (Jennings & Muldner, 2021; Lee & Hong, 2017; Renkl et al., 2009). The multimedia design principles may include image, personalization, temporal contiguity, spatial, coherence and redundancy principles (Lee & Hong, 2017). Self-explanation is another essential aspect of instructional design that should be adhered to when developing compelling worked examples (Lee & Hong, 2017; Renkl et al., 2009).

Fading worked examples.

Another approach to design worked examples involves providing the learner with as many performances supports as are required to accomplish a goal, and then gradually removing those supports as the learner becomes more capable of accomplishing the purpose on their own (Bunch, 2009). This approach is called *fading worked examples*, with fading learners at the beginning of problem solving are provided with a full worked example of a solution, after the provided problem is fading gradually into subproblems that need to be solved (Moreno et al., 2009).

According to the literature, two forms of faded worked examples can be implemented, *backward fading (BF)* and *forward fading (FF).* The distinction between the two formats indicates that in *backward fading (BF)*, learners are completing "the last solution step of the first practice problem, the last two solution steps of the second practice problem, and so on, until they solve all steps" (Moreno et al., 2009, p.83). While in *forward fading (FF)*, learners are completing "the first solution step of the first practice problem, then the first two solution steps of the second practice problem, and so on, until all steps are solved" (Moreno et al., 2009, p.83). Moreover, when learners are given fading as an instructional strategy, they are prompted to reflect on what they learned from the prior scaffolds and apply that information to solve the given problem (Moreno et al., 2009; Vieira et al., 2015).

In addition, the faded worked examples approach "has shown positive results in reducing cognitive loads in the domains of mathematics and programming" (Vieira et al., 2015, p.3). In Boldbaatar and Şendurur (2019) study, the researchers investigated the effect of backward fading worked example designs in both block-based and text-based programming environments. The study shows the high effectiveness of backward fading in improving learners' ability in completing algorithms in block-based programming environments, specifically, novice learners (Boldbaatar & Şendurur, 2019).

Furthermore, Si et al. (2014) applied backward fading worked examples as a prior training session to teach the algorithm design of a program code; this was done in the context of teaching the algorithm.

In-game worked examples.

Another idea was motivated by assessing the consequences of playing a programming game with and without the assistance of studying worked examples. The purpose of the research

carried out by Toukiloglou and Xinogalos (2022) is to explore the effect of using worked examples as a support system to assist learners in engaging with a distinctive programming environment. Learners are required to interact with the programming environment to solve specific problems. The appearance of the worked examples guides them through the interaction with the block-based programming environment (Toukiloglou & Xinogalos, 2022). The researchers concluded that the special in-game design with worked examples assists in activating the schemata, which significantly positively affects the learners' performance in programming (Toukiloglou & Xinogalos, 2022).

Summary of research question #1.

Based on the previous studies, seven forms of worked examples were recognized as designs to be included in the programming settings and used to aid in enhancing the multiple computational thinking skills. The designs or forms include animated worked examples, process-oriented worked examples, sequenced worked examples, subgoal design, tutorial as worked examples, faded worked examples and in-game worked examples.

Each design required specific considerations either following multimedia learning principles design or other instructional design principles that may have an impact on managing the cognitive load and enhance the learning process. The following table (Table 2) shows the summary results of the first research question with the related studies.

Table 2

A summary of research question no.1.

WEs Design	Article (by author/s)	Computational skills	Programming language/ course
animated worked examples	Chang et al., 2011; Cevahir et al., 2022; Hosseini et al., 2020; Hsu et al., 2012; Vieira et al., 2018.	Algorithm design, modeling.	Intro to programming, windows programming, Python, Thermodynamics.
Process-oriented worked examples WOE	Si et al., 2014.	Algorithm design.	C programming.
Sequencing set of worked examples	Abdul-Rahman & Du Boulay, 2014; Bunch, 2009; Chang et al, 2000; Garner, 2007; Nainan & Balakrishnan, 2019; Si et al., 2014.	Algorithm design, data collection, data analysis, evaluation.	Intro to programming, C programming.
Subgoal worked examples	Berssanette & de Francisco, 2021; Margulieux & Catrambone, 2021; Margulieux et al., 2012; Margulieux et al., 2018; Morrison et al., 2020.	Algorithm design,	Intro to programming, Maker App
Worked examples as tutorials	Grover, 2017; Jennings & Muldner, 2021; Jocius et al., 2021; Kale et al. 2018; Lee & Hong, 2016;	Decomposition, abstraction, pattern recognition, algorithm design	Intro to programming, Python, Snap! Programming, JavaScript
Fading worked examples	Boldbaatar & Şendurur; 2019; Bunch, 2009; Moreno et al., 2009; Vieira et al., 2015	Algorithm design, data collection, data analysis.	StarLogoBlocks, C programing language, intro to programming
In-game worked examples	Toukiloglou & Xinogalos, 2022	Algorithm design	Blockly

Findings of Research Question 2

The second research question "*What instructional design principles have been recognized in the literature and need to be considered when designing WEs to promote CT skills?*", is seeking information about the most frequently instructional design principles that have been employed to design worked examples in programming settings. Approximately 50 % of the gathered studies explored several instructional design principles that have been used to design effective worked examples.

The design of worked examples played an essential role in providing effective instruction and ensuring successful learning results (Atkinson & Renkl, 2007; Chang et al., 2011; Hsu et al., 2012; Margulieux & Catrambone, 2021; Nainan & Balakrishnan, 2019; Lee & Hong, 2017).

The IR found a theoretical review of empirical studies conducted by Shen and Tsai (2009) this review analyzed 8 instructional design principles for developing worked examples in several science domains including computer science. According to Shen and Tsai (2009) review, designing effective worked examples may follow several instructional designing principles, as the following: 1) imagination principle, 2) completion principle, 3) fading principle, 4) process principle, 5) presentation principle, 6) media principle, 7) timing principle, 8) self-explanation principle.

The next table (Table 3) provided an overview of Shen and Tsai (2009) instructional design principles list and their implications within the current IR studies.

Table 3

Instructional design principles for designing worked examples according to Shen and Tsai (2009) and how it is applied with the results of the current study.

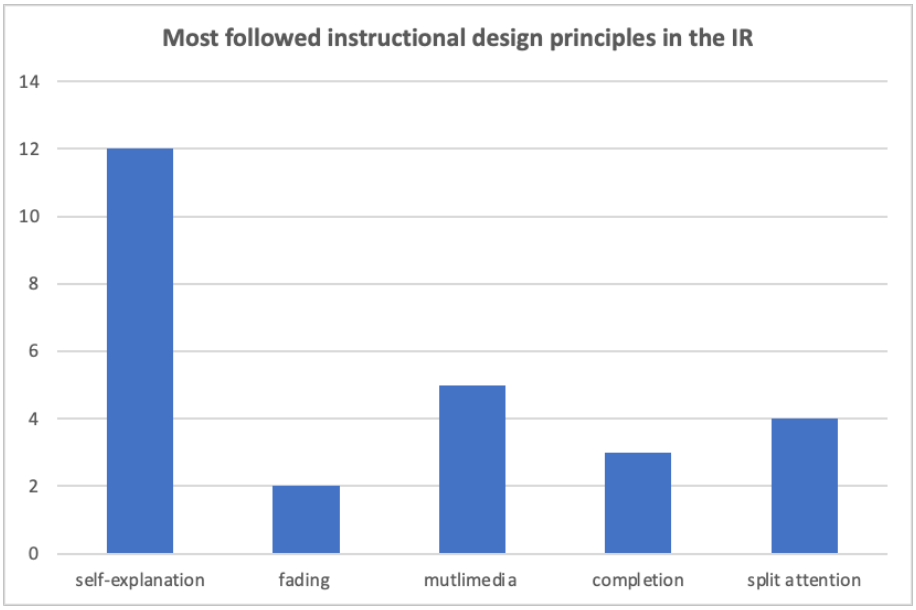
The principle	Description	Applied in programming settings	Related Studies
Imagination principle	Learners who already possessed the necessary schemas would convey that imagining positively impacted their learning more than studying the content itself. On the other hand, less experienced learners would find that imagining had a more detrimental impact on their learning (Shen & Tsai, 2009).	N/A	N/A
Completion principle	A completion of a problem is "is a partial worked example where the learner has to complete some key solution step" (Sweller et al., 2011, p.105).	Yes	(Abdul-Rahman & Du Boulay, 2014; Chang et al. 2000; Davidovic et al., 2003; Garner, 2007).
Fading principle	It is based on the idea that learners will be able to handle increasing demands on their working memory if they gradually get less guidance with solving problems and have to solve more complex problems as they get better at solving problems (Shen & Tsai, 2009; Sweller et al., 2011).	Yes	(Boldbaatar & Şendurur, 2019; Moreno et al., 2009; Vieira et al., 2015).
Process principle	It indicates providing chunks of the problem solutions, adding labels, underlying problem structure, and guiding learners to discover more about the process of solving specific problems (Shen & Tsai, 2009).	Yes	(Berssanette & de Francisco, 2021; Margulieux et al., 2016; Margulieux et., 2018; Margulieux et al., 2020; Margulieux & Catrambone; 2021; Margulieux & Catrambone; 2016; Morrison et al.,

			2016; Si et al., 2014; Van Gog et al., 2004).
Presentation principle	Learners gained more from instructional materials that indicated visuals such as diagrams and graphics as opposed to materials that mainly included text (Shen & Tsai, 2009).	Yes	(Cevahir et al., 2022; Chang et al., 2011; Hosseini et al., 2020; Hsu et al., 2012).
Media principle	It implies incorporating visual, textual, and aural components together with worked examples to attract the learner's attention and to create additional opportunities for engagement with the material (Shen & Tsai, 2009).	Yes	(Grover, 2017; Jennings & Muldner, 2021; Jocius et al., 2021; Kale et al., 2018; Lee & Hong, 2017; Renkl et al., 2009).
Timing principle	It reveals that delivering the worked examples followed by other comparable examples and problems as sequences can significantly influence the learning process than simply offering the worked examples alone (Shen & Tsai, 2009).	Yes	(Bunch, 2009; Davidovic et al., 2003; Mahatanankoon & Wolf, 2021; Nainan & Balakrishnan, 2019).
Self-explanation principle	It is defined as "a mental dialogue that learners have when studying a worked example that helps them understand the example and build a schema from it" (Clark et al., 2011, p. 226).	Yes	(Alhassan, 2017; Atkinson & Renkl, 2007; Berssanette & de Francisco, 2021; Hosseini et al., 2020; Margulieux & Catrambone; 2021; Margulieux & Catrambone; 2016; Margulieux et al., 2016; Nainan & Balakrishnan, 2019; Vieira & Magana, 2015; Vieira et al., 2021; Vieira et al., 2019;Vieira et al., 2017).

In addition, the IR analyzed around 45% of the collected studies that employed various instructional design principles, including self-explanation, fading, multimedia, completion, and split attention. Additionally, according to the IR, it has been found that self-explanation is the most utilized instructional design principle, followed by multimedia, split attention, completion, and then fading. The next figure (Figure 5) illustrates the distribution of the most employed instructional design principles in this IR study.

Figure 5

The most followed ID principles of developing WEs in the IR.



Note. The figure above illustrates the most often used instructional design principles in creating worked examples found in the IR.

Self-explanation effect principle

Explaining and self-explaining have been shown to be crucial cognitive processes that facilitate effective learning and transfer (Alhassan, 2017; Atkinson & Renkl, 2007; Hosseini et al., 2020; Margulieux & Catrambone; 2021; Vieira et al., 2019). In the context of learning

through worked examples, Atkinson and Renkl (2007) defined the self-explanation technique as learners' efforts to understand better how to approach the problem presented in the example.

When it comes to learning computer programming skills, the adoption of self-explanation has demonstrated favorable effects for learners with both low and high levels of prior knowledge (Alhassan, 2017; Berssanette & de Francisco, 2021). The integration of "the worked-examples with the self-explanation strategy and the project structure provided the supportive information in the form of cognitive strategies for problem solving" (Vieira et al., 2021, p.356).

Accordingly, several strategies for facilitating the adoption of worked examples with self-explanation were analyzed in the current IR. For insurance, while providing computational instruction, Vieira et al. (2019) encouraged learners to provide their explanations by using the *in-code* comment element - that is built in the programming environment. Researchers with this strategy found that commenting while coding is a key component of fundamental programming lessons, and it is frequently used to communicate among programmers (Vieira et al., 2019). Researchers of the experiment illustrated that "self-explanations are usually studied following a think-aloud protocol, or as written statements independent from the code" (Vieira et al., 2019, p.202).

According to Alhassan (2017), learners can avoid making the common mistakes that first-time programmers make with the aid of self-explanation and worked examples. Integrating self-explanation and worked examples showed how the commands should be used in various scenarios, allowing the learner to create new code with fewer mistakes (Alhassan, 2017). To illustrate, indicating writing comments in the code, would increase the awareness of the example and the steps of solving the programming problem (Berssanette & de Francisco, 2021).

Furthermore, Hosseini et al. (2020) discovered that demonstrating worked examples with self-explanation in the coding process may motivate learners to develop the mental process of understanding the algorithm of the code.

Moreover, Nainan and Balakrishnan (2019) claimed that incorporating explanations into various parts of the solution in the form of descriptive labels is a viable strategy that exceeds expectations for fostering self-explanation. Therefore, the prospect of indicating the subgoal worked examples format to support the information processing in a way that generates long-term learning arises from this technique (Margulieux et al., 2016). Providing a support system based on self-explanation and worked examples at the initial problem-solving process would ultimately improve performance in subsequent problem-solving attempts (Margulieux & Catrambone; 2021).

However, to ensure the effectiveness of self-explanation and worked examples, Vieira and Magana (2015) stated that:

a self-explanation should contain four aspects that depict an understanding:

(1) the conditions of application of the actions.

(2) the consequences of actions.

(3) the relationship of actions to goals; and

(4) the relationship of goals and actions to natural laws and other principles (pp.12-13).

Furthermore, the literature showed that in designing self-explanation, certain aspects of these explanations, such as the fact that they link to fundamental principles, could be the key to gaining a deeper comprehension of the phenomena (Atkinson & Renkl, 2007; Vieira et al., 2019).

The Multimedia learning principle

According to Renkl (2014), "Learners gain deep understanding in multimedia-based learning environments when they receive worked examples in initial cognitive skill acquisition" (p.391).

Following an overview of the principles of multimedia that are applied in the field of instructional design, it has been found that learning with "words and pictures" is shown to be more successful than learning using text alone (Butcher, 2014, p.174; Mayer, 2006). Whereas it is essential to include visuals to support the procedures in worked examples, also it is imperative to deal with complicated problems involving a more significant cognitive mental load (Chang et al., 2011; Hsu et al., 2012; Renkl, 2014; Si et al., 2014).

As a result, it is essential to pay special attention to the process of developing worked examples while interacting with instruction in a digital environment such as a programming context.

In Chang et al. (2011) study, the researchers followed the multimedia learning principles as they developed an executable program that demonstrated the necessary steps to design the suitable algorithm and to promote computational thinking skills. The researchers followed a specific principle of multimedia principles, known as (temporal contiguity). Mayer (2001) described temporal contiguity as the technique when learners “learn better when corresponding words and pictures are presented simultaneously rather than successively“(p.96). It was discovered that adhering to the idea of temporal contiguity while developing the executable program improved the learner's ability to make connections between the textual descriptions and the relevant visual elements (Chang et al., 2011).

Furthermore, (Modality principle) is another multimedia design principle that has been identified in the current IR. Mayer (2011) described the modality principle as the technique when learners "learn better from animation and narration than from animation and on-screen text; that is, students learn better when words in a multimedia message are presented as spoken text rather than printed text"(p.134). Si et al. (2014), followed the modality principle in designing the experiment materials by developing two types of worked examples in conjunctions with multimedia; the bimodal process-oriented WEs contained visual and audio, and unimodal process-oriented WEs contained visuals only. The results indicated that the bimodal process-oriented WEs had a significant positive effect in controlling the learning cognitive load as well as successful programming performance (Si et al., 2014).

Moreover, Hsu et al. (2012), followed the multimedia principle in developing the worked examples to promote computational thinking skills. Multimedia principles indicate that learners "learn better from words and pictures than from words alone" (Mayer, 2011, p.63). The researchers concluded that providing multimedia and worked examples had a positive effect in controlling the cognitive load enhancing the programming skills as well (Hsu et al., 2012).

In addition, Lee and Hong (2017) applied several multimedia learning principles when they developed worked examples as a tutoring system. Spatial, temporal contiguity, coherence, redundancy, image, and personalization principles were used in Lee and Hong (2017) study. The following table (Table 4) provides an overview of Lee and Hong (2017) design principles and their description.

Table 4

Description of multimedia learning principles in (Lee & Hong, 2017) study.

The multimedia principles	Description
spatial	Learners "learn better when corresponding words and pictures are presented near rather than far from each other on the page or screen" (Mayer, 2011, p.135).
temporal contiguity	Learners "learn better when corresponding words and pictures are presented simultaneously rather than successively"(Mayer, 2011, p.96).
coherence	Learners "learn better when extraneous material is excluded rather than included" (Mayer, 2011, p.89).
redundancy	Learners "learn better from graphics and narration than some graphics, narration, and printed text" (Mayer, 2011, p.118).
image	Learners "do not necessarily learn better when the speaker's image is added to the screen" (Mayer, 2011, p.242).
personalization principles	Learners "learn better from multimedia presentations when words are in conversational style rather than formal style" (Mayer, 2011, p.242).

The study resulted in a significant effect of the multimedia design of the tutoring system on enhancing the learner's computational skills as well as their motivation towards programming (Lee & Hong, 2017).

Split Attention effect principle

According to the multimedia learning principles, split attention indicates that it is preferable to steer clear of the sources that force learners to divide their attention and cognitively integrate data from several sources (Ayres & Sweller, 2014). Consequently, instruction should be designed in such a way that diverse information sources are visually and chronologically merged,

eliminating the necessity for learners to participate in the mental fusion (Ayres & Sweller, 2014; Chang et al., 2011; Hsu et al. 2012).

Furthermore, studies showed that following the split-attention method implies positive effects regarding using a dual screen in decreasing the cognitive load, in contrast to using the single screen (Berssanette & de Francisco, 2021; Chang et al., 2011; Vieira et al., 2015).

Fading effect principle

Cognitive load theory (CLT) studies have demonstrated that providing worked examples first, followed by problems to be addressed, is the most effective way to promote cognitive skill learning in well-structured domains (Atkinson et al., 2000; Sweller et al., 2003). However, Sweller et al. (2011) claimed that "instructional designs and techniques that are relatively effective for novice learners can lose their effectiveness and even have negative consequences with increasing levels of expertise" (p.171). This implies that providing the learners with comprehensive guidance could impede their ability to enhance their learning skills. Therefore, the fading effect used to be useful in designing worked example series (Hosseini et al., 2020).

The current IR analyzed several studies that followed the fading technique in developing worked examples to promote computational thinking skills. For instance, Boldbaatar and Şendurur (2019) applied the fading backward techniques when developing worked examples to enhance programming skills. The research showed that faded worked examples were beneficial in aiding learners to remember the processes that are needed to accomplish the algorithm design of the programmed problem (Boldbaatar & Şendurur, 2019). In a similar manner, the fading feature was utilized by Vieira et al. (2021) while they were developing scaffolding worked examples that gradually faded away from the algorithmic code. Also, Hosseini et al. (2020)

followed the principles of fading when they developed interactive worked examples to enhance learning Python programming language.

However, the complexity level of the proposed problem plays a role in determining whether or not the fading effect should be used when constructing worked examples (Moreno et al., 2009).

Completion effect principle

The IR analysis shows that the effective design of worked examples is associated with several other instructional techniques. The completion effect with worked example "is a partial worked example where the learner has to complete some key solution steps" (Sweller et al., p.105). Learners enhanced their learning skills by first reviewing completed worked examples and then applying what they had learned to solve a portion of the problem (Abdul-Rahman & Du Boulay, 2014; Berssanette & de Francisco, 2021; Van Gog et al., 2004).

Summary of research question #2.

Based on the previous studies, several instructional design principles were being followed when developing worked examples to promote computational thinking skills. For instance, multimedia principles, fading effect, split attention effect, self-explanation effect, completion effect. In addition, it has been shown that the self-explanation effect is one of the most important principles that could be considered when designing worked examples in programming settings.

Furthermore, IR discovered that algorithm design, decomposition, and computational modeling were the most targeted computational thinking skills when answering the second research question. Besides, those studies investigated their experiments in several programming courses. The following table (Table 5) shows the summary results of the second research question with the related studies.

Table 5

A summary of research question no.2.

IDT Design	Article (by author/s)	Computational skills	Programming language/ course
Self-explanation	Alhassan, 2017; Berssanette & de Francisco, 2021; Hosseini et al., 2020; Margulieux & Catrambone; 2021; Vieira et al., 2019	Algorithm design, Decomposition.	Intro to programming, StarLogoBlocks, Python, Maker App, MATLAB.
Multimedia	Chang et al., 2011; Hsu et al., 2012; Lee & Hong, 2017; Si et al., 2014.	Algorithm design.	Windows programming, intro to programming, JavaScript, C programming.
Split attention	Berssanette & de Francisco, 2021; Chang et al., 2011; Hsu et al. 2012; Vieira et al., 2015	Algorithm design.	Intro to programming, StarLogoBlocks, Windows programming, C programming
Fading	Boldbaatar & Şendurur, 2019; Moreno et al., 2009; Vieira et al.,2021	Algorithm design, computational modeling	Intro to programming, StarLogoBlocks, MATLAB
Completion	Abdul-Rahman & Du Boulay, 2014; Berssanette & de Francisco, 2021, Chang et al., 2000; Garner, 2007.	algorithm design.	Intro to programming.

Findings of Research Question 3

The third research question "*What factors or circumstances might have an impact on the development of WEs for well-structured programming problems*?", is seeking information about factors that are affecting the design worked examples in programming settings. Identifying such factors might have led to a more effective learning experience. Twenty-nine (n=29) articles were

included in the analysis for this question, and those studies discussed several factors that have an impact on the designing worked examples.

Utilizing worked examples as part of an instructional strategy is one method that has the potential to improve problem-solving in highly structured disciplines such as physics, mathematics, and computer programming (Abdul-Rahman & Du Boulay, 2014; Alhassan, 2017; Vieira et al., 2015). Under some circumstances, Ward and Sweller (1990) proposed that "worked examples are no more successful, and potentially less effective, than solving problems"(p.1). The ineffective design or worked examples may affect the learning process negatively because it "might burden the student's working memory" (Atkinson et al., 2000, p.186). Therefore, structuring worked examples is essential in order to reduce any cognitive load that could arise throughout the process of learning how to program and might have an effect on the learning outcomes (Chang et al. 2011; Margulieux & Catrambone; 2021).

For this question, the data analysis of peer-reviewed journal articles revealed two general themes. The first theme is connected with the moderating factors influencing the design of working examples identified by Atkinson et al. (2000), and the second with numerous external components involved with the learning process.

As an effort to increase the effectiveness of worked examples, pioneers of the field of instructional design indicated that the effectiveness of worked examples is determined based on several factors, and they classified them as:

a) intra-example features,

b) inter-example features, and

c) environmental / situational aspects (Atkinson & Renkl, 2007; Atkinson et al., 2000).

For more explanation:

***Intra-example* features**

Intra-example features are associated with the approach in which the worked example is designed, as well as the way in which the stages of the worked example are presented (Atkinson et al., 2000). One key to increase the effective design of worked examples lies in the manner in which different kinds of elements, such as text and diagrams, visual and aural information, are integrated (Atkinson & Renkl, 2007; Atkinson et al., 2000; Renkl et al., 2009). Various intra-example designs may provide more or less information about the problems' solutions (Abdul-Rahman & Du Boulay, 2014; Atkinson et al., 2000; Chang et al., 2000; Garner, 2007; Renkl et al., 2009), providing verbal explanations within WEs (Abdul-Rahman & Du Boulay, 2014; Hsu et al., 2012), several modality representations of the problem's solutions (Grover, 2017; Jennings & Muldner, 2021; Jocius et al., 2021; Kale et al., 2018; Lee & Hong, 2017; Chang et al., 2011).

In terms of probable contributing elements, the current IR found that intra-example features are the most prominent factor in this research. Approximately 61% of the analyzed studies in this section cover various approaches to presenting worked examples, each with the implications for how the solution processes should be displayed. For instance, to encourage algorithm construction in an introductory programming course, Abdul-Rahman and Du Boulay (2014) studied the impact of two web-based interfaces for delivering worked examples with varying levels of information about the problem's solution. In their study, Abdul-Rahman and Du Boulay (2014) utilized CORT (Code Restructuring Tool) - created by Garner (2007) - to deliver programming worked examples that were designed as incomplete. This interface has two parts: one describes the incomplete design of a worked example, while the other provides users with the necessary code to complete the programming task (Abdul-Rahman & Du Boulay, 2014).

According to Renkl et al. (2009), providing learners with an incomplete worked example design has an effect on developing their ability to think, solve a problem, and enhance their self-explanation abilities. Additionally, Chang et al. (2000) utilized complete and partial worked examples through three phases of exercises. These exercises involved engaging with the material, practicing, and filling in the missing areas of the programmed codes (Chang et al., 2000).

In addition, providing verbal explanations within the worked examples design has an impact on the learning process (Renkl et al., 2009). As per Abdul-Rahman and Du Boulay (2014), a textual explanation is an option that can be used when designing worked examples. This textual explanation option involves the use of interactive bullets that give the learner the ability to expand and discover additional information regarding the programmed problems (Abdul-Rahman & Du Boulay, 2014). Moreover, the verbal explanation of the solutions' phases toward the correct algorithm design of C programming codes was included by Vieira et al. (2015) into a flowchart design of the solution's processes.

Furthermore, the intra-example feature, in addition to the previous implementations, includes several modality representations of the problem's solutions. This involves integrating worked example text with visuals, audio, animations, or a combination of these elements (Atkinson et al., 2000; Renkl et al., 2009). For instance, Chang et al. (2011) developed an executable program of a worked example solution that simulates the essential procedures to solve a programming problem. Additionally, several of the gathered studies benefited from interactive tutorials that aimed to improve learners computational thinking skills. Several researchers have made use of the access of video tutorials that imitate the algorithm design processes of solving programming problems (Grover, 2017; Jennings & Muldner, 2021; Jocius et

al., 2021; Kale et al., 2018; Lee & Hong, 2017). The integration of animations with worked examples was another way to help learners learn how to code. These animations might act as guides or simulations of the programming processes (Cevahir et al., 2022; Chang et al., 2011; Hosseini et al., Hsu et al., 2012; 2020; Vieira et al., 2018).

Furthermore, presenting chunks of the problems to be solved as subgoals is seen as a crucial component that is associated with the intra-example feature of the effective design of worked examples (Atkinson & Renkl, 2007; Atkinson et al., 2000). Subgoals of the programming problems have been used in several studies, for example, Margulieux and Catrambone (2016) employed subgoal design within the worked examples in a programming environment. The study suggested that providing subgoal explanation of the programming problems is essential and it makes the "learning process passive and dictated by experts rather than by learners" (Margulieux & Catrambone, 2016, p. 503). Berssanette and de Francisco (2021) recognized the importance of designing worked examples and how it may affect the learning process in their systematic review. The findings about subgoals are consistent with the reverse expertise effect, and they implied that as learners get more proficient at solving problems, the subgoal labeling strategy becomes less helpful (Berssanette & de Francisco, 2021).

***Inter-example* features**

Inter-example features are related to the relationship between the provided worked examples (Abdul-Rahman & Du Boulay, 2014). It is crucial to evaluate how the worked examples are sequenced, as well as how they compare to one another and how they are connected to one another, in order to produce an effective design for the worked examples (Hsu et al., 2012; Vieira et al., 2015).

According to Atkinson et al. (2000), it is important to consider:

(1) the number of examples to present during instruction,

(2) how and whether examples should be varied within a lesson,

(3) how themes or "surface stories" might be varied to instructional advantage, and

(4) how practice and examples should be intermingled (p.191).

The inter-example feature offers numerous worked example designs, including the presentation of more or less worked examples (Abdul-Rahman & Du Boulay, 2014; Hsu et al., 2012, Vieira et al., 2015), and it is also related to the presentation of several problem forms (Bunch, 2009). The IR assessed about 19% of studies in this section that covered various approaches to consider while building a variety of worked examples.

Hsu et al. (2012), designed an interactive programming material, including the corresponding worked examples in several slides, and each example provided a simulation of the algorithm design of the programming language. Moreover, to determine the impact of structure-emphasizing worked examples, Abdul-Rahman and Du Boulay (2014) presented pairs of worked examples of programming problems and allowed the learner to explore additional matched problems that were attached as external references. While Vieira et al. (2015) presented three activities, each consisted of a pair of programming worked examples, which was followed by a third exercise to evaluate achieving the instructional goals.

In a similar manner, the IR discovered one article (n=1), which explored the effectiveness of offering various scenarios as several types of problems within the worked example design (Bunch, 2009).

The environmental/situational features

Environmental or situational features are associated with other elements that may affect learning from worked examples (Abdul-Rahman & Du Boulay, 2014; Vieira et al., 2015). This specifically deals with "the ways in which examples are used by the problem solver, particularly the practice of explaining examples to one's self and to others" (Atkinson et al., 2000, p.195). The current IR discovered that the environmental/ situational aspects are less discussed than the previous factors with assessing only two elements related to these features, the demo-practicing, and self-explanation. Only one study (n=1) focused on providing the pre-training as a demo-practice to strengthen the effectiveness of worked example design in a programming domain (Lee & Hong, 2017).

While the most common environmental or situational feature that has an influence on designing worked examples is providing the self-explanation. Self-explanation itself has been mentioned as an important element in approximately ten studies (n=10), revealing the importance of integrating self-explanation when developing worked examples. The way of embedding self-explanation took many forms in the collected studies. Approximately 66% of the analyzed studies in this section, integrated the concept of self-explanation into programming courses through the use of worked examples and required the learners' participation in graded self-explanation assignments (Alhassan, 2017; Hosseini et al., 2020; Margulieux & Catrambone; 2021; Margulieux & Catrambone; 2016; Margulieux et al., 2016; Nainan & Balakrishnan, 2019; Vieira et al., 2021). On the other half, two studies (n=2), employed (in code) comment feature to allow participating in providing self-explanation comments about the programming problems (Vieira et al., 2019; Vieira et al., 2017).

The following table (Table 6*)* will provide an overview of the factors that moderating the design of effective worked examples based on Atkinson et al. (2000) factors, in conjunction with examples of their implementation in the analyzed studies.

Table 6

The integration between the factors affecting the design of worked examples by Atkinson et al. (2000) and their implementation in the IR studies.

The factors/ features	**Description**	**IR studies implementation**
intra-example	-Focusing on how the worked examples are designed and how the process solutions are presented (Atkinson et al., 2000). -how to integrate text and diagrams effectively within the worked examples (Atkinson et al., 2000). -how to integrate aural and visual information (Atkinson et al., 2000). -"Use of multiple modalities (aural, visual, etc.)" (Atkinson et al. 2000, p.203). -how to integrate example's parts (Atkinson et al., 2000). -subgoals design (Atkinson et al. 2000). - completeness of examples (Atkinson et al. 2000).	Worked examples included several representations like -visual, flowcharts, algorithm descriptions, and verbal explanations of the solution (Hsu et al., 2012; Vieira et al., 2015). -An executable program simulates the algorithm design (Chang et al., 2011). -interactive tutorials (Grover, 2017; Jennings & Muldner, 2021; Jocius et al., 2021; Kale et al., 2018; Lee & Hong, 2017). -integration animations within worked examples (Cevahir et al., 2022; Chang et al., 2011; Hosseini et al.,2020; Hsu et al., 2012; Vieira et al., 2018). -subgoal worked examples (Berssanette & de Francisco, 2021; Margulieux et al., 2016; Margulieux & Catrambone, 2016). -partial/complete worked examples (Abdul-Rahman & Du Boulay, 2014; Chang et al., 2000; Garner, 2007).

inter-example	-Focusing on the relationship between the examples and the problem's solution (Atkinson et al. 2000). -the variety of problems within the instructions (Atkinson et al. 2000). -"Surface features that encourage search for deep structure" (Atkinson et al., 2000, p.203). -several numbers, forms, or problem's types (Atkinson et al. 2000).	-Multiple visual slides of the algorithmic worked examples (Abdul-Rahman & Du Boulay, 2014; Hsu et al., 2012;Vieira et al., 2015). -multiple problems' scenarios embedded within worked examples (Bunch, 2009).
environmental /situational features	-pre-training, social interaction, self-explanation. (Atkinson et al. 2000).	-demo practicing (Lee & Hong, 2017). -self explanation (Alhassan, 2017; Hosseini et al., 2020; Margulieux & Catrambone; 2021; Margulieux & Catrambone; 2016; Margulieux et al., 2016; Nainan & Balakrishnan, 2019; Vieira & Magana, 2015; Vieira et al., 2021; Vieira et al., 2019; Vieira et al., 2017).

In addition to the previously mentioned factors, nearly 51% of the studies examined in this part, are investigated various aspects and circumstances that influenced the quality of worked example designs.

The programming language regarding content specific has an impact on transferring knowledge and developing learning materials (Kale et al., 2018). For this reason, studies urge paying attention to teacher development programs, notably ones that focus on developing innovative methods to increase computational thinking skills, integrating technology, and teaching digital skills (Grover, 2017; Kale et al., 2018; Uysal, 2014). In addition, "teaching style" (p.202) and subject goals are considered as important factors when designing learning

support materials including worked examples to promote computational thinking skills (Uysal, 2014).

Furthermore, understanding the learners' differences before creating worked examples aids in selecting the best programming domain or language to achieve the educational aims. For example, the Drag-and-Drop programming language is an appropriate option for offering programming instruction to novice programmers (Grover & Pea, 2013). For this purpose, Margulieux and Catrambone (2021) used Drag-and-Drop programming language in association with worked examples to provide the appropriate guidance for novice programmers while solving programming problems.

In addition to selecting the appropriate programming language, it is a crucial step before designing worked examples to understand the variations amongst learners. This assists in determining the learners' time management in relation to completing algorithmic tasks (Hosseini et al., 2020). Consequently, recognizing learner differences leads to assessing the comprehensive support system that is appropriate for the targeted learners, inspiring the successful design criteria for worked examples, and offering any further assistance required, such as hints and clues (Jennings & Muldner, 2021; Margulieux & Catrambone, 2021).

Recognizing learners' differences is an important factor that affects the design of worked examples (Uysal, 2014). Learners may be differentiated from one another based on their prior knowledge of a particular topic, which characterizes them as either novices or experts (Buitrago Flórez et al., 2017). Accordingly, this element is very important to recognize when designing worked examples, specially, that programming is difficult for novice learners (Lee & Hong, 2017). Knowledge is acquired and generalized more successfully when presented to novice learners in the form of explicit worked examples that are meant to assist them in solving

problems (Hosseini et al., 2020). However, designing worked examples in an explicit way is important because novice learners tend to "organize and encode procedural information based on surface features of the problem" (Margulieux et al., 2018).

Furthermore, Vieira et al. (2019) discovered that the learner's ability to elaborate and comment on the coding problem and to justify their decisions at each stage of algorithm construction is a significant component that may influence the creation of worked examples. Learners' ability to explain and comment is associated with self-explanation, an important situational factor (Atkinson et al. 2000). Self-explanation prompts are considered as an important factor that affect the design of worked examples, specifically in algorithmic domains like computer science (Alhassan, 2017; Renkl et al., 2009; Vieira & Magana, 2015; Vieira et al., 2021). In designing backward and forward worked examples, researchers implied associating self-explanations within the examples to promote a deeper understanding (Moreno et al., 2009). Commenting is advantageous not just to the learner who has engaged with the worked examples, but also to other learners who might benefit from sharing peer comments on the algorithmic task (Wang & Hwang, 2017).

Besides, another element that influences the design of worked examples is the average time necessary to construct an algorithm design by emulating the steps of a previous example, especially if worked examples have the engaging feature (Jennings & Muldner, 2021; Si et al., 2014). Concurrently, there is considerable consensus that engagement is a component that plays a vital role in the effectiveness of the learning process (Abdul-Rahman & Du Boulay, 2014; Hosseini et al., 2020; Uysal, 2014). Therefore, engaging the learners and ensuring that they remain involved throughout the process of following the worked examples is essential

component that contributes to the success of using worked examples (Boldbaatar & Şendurur, 2019; Hsu et al., 2012; Nainan & Balakrishnan, 2019).

Summary of research question #3.

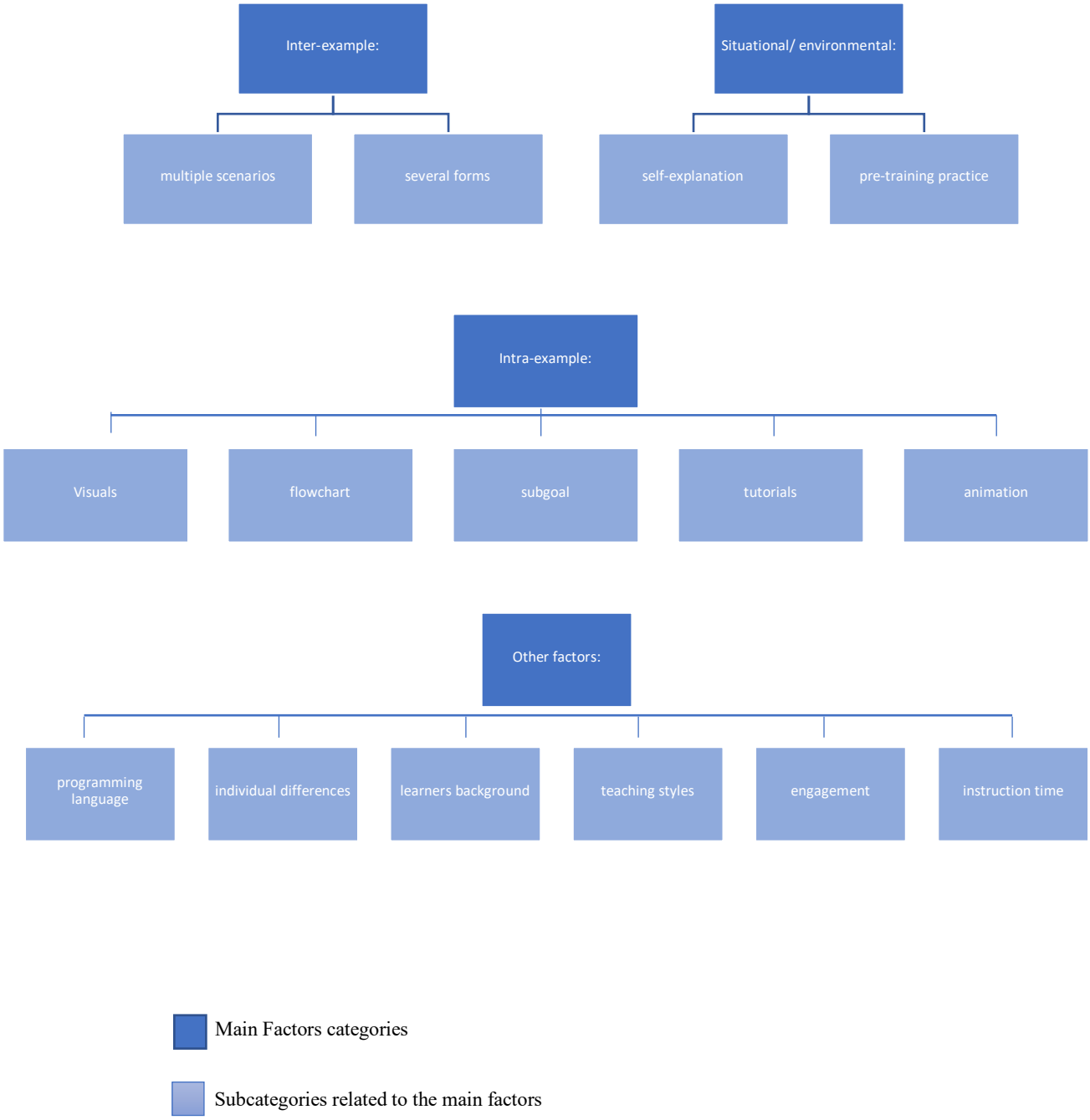
The third research question revealed several factors have been identified in a various number of studies. All these factors are important to consider when designing worked examples, specifically for promoting computational thinking skills in programming settings. When considering designing effective worked examples, the instructional designer or the instructor should pay particular attention to the design features of the worked examples.

When planning to design worked examples and considered the way of presenting those examples, it is crucial to take into account the following three intra-example features: combining text and graphics, integrating auditory and visual data, and incorporating phases and subgoals (Abdul-Rahman & Du Boulay, 2014; Hsu et al., 2012; Vieira et al., 2015). Whereas if the instructional designer or the instructor is planning to design a sequence of worked examples, it is important to consider the inter-example features that are concerned on the way that the examples should be presented during instruction (Hsu et al., 2012; Vieira et al., 2018). In addition, there might be a need to consider additional environmental factors that may have an impact on the design; these factors, for the most part, are not directly connected to the worked examples more than the learners, environment, and all other elements that are related to the learning process (Nainan & Balakrishnan, 2019; Vieira et al., 2017). However, other elements associated with learning process such as learners' differences, programming languages preferable, teaching style, instruction time, and learning engagement (Alhassan, 2017; Grover, 2017; Jennings & Muldner, 2021; Renkl et al., 2009; Uysal, 2014; Vieira & Magana, 2015; Vieira et al., 2021).

The next figure (Figure 6) is a mind map that illustrates an overview of these factors.

Figure 6

Mind map of the most important factors that affect the design of WEs in the IR.



Findings of Research Question 4

The fourth research question "*What are some examples of effective WEs for well-structured programming problems that have been identified in the literature*?", is seeking information about successful examples of employing worked examples to promote computational thinking skills. The IR found that approximately 78 % of the studies reported numerous successful experiences of employing worked examples in programming settings.

The study categorized the successful examples of employing worked examples to promote computational thinking skills into three subcategories:

a) *Medium* concentrates on the delivery system where the worked examples are presented.
b) *Supportive tools* concentrate on the utilized tools that presented or provided additional support with the worked examples.
c) *Techniques* concentrate on strategies that have been integrated with worked examples to promote the computational thinking skills.

Medium: Online learning environments

Within the scope of this analysis, an online learning environment and web pages served as the vehicles for delivering the programming instruction and its accompanying worked examples. Traditional worked examples via worksheets followed by the textbook programming instruction as well as digital worked examples via digital slides have been utilized in addition to practicing the programming tasks in computer labs (Alhassan, 2017; Bunch, 2009; Chang et al., 2011, Grover, 2017).

Wang and Hwang (2017) were able to develop a model for online team-based learning that is centered on generating problems and collaborating to solve them by integrating the Wiki functionality module into the Moodle learning management system. Researchers indicated that the learning environments allow learners to "edit" (p.1659), and participate by comments (Wang & Hwang, 2017). Also, Moodle environment provided the learners with the opportunity to record each version of the modified outcomes, which enabled them to compare and contrast the differences that have occurred among the various codes (Wang & Hwang, 2017). Furthermore, Vieira et al. (2021) used Blackboard to provide a number of resources, such as video lessons, worked examples, and code examples, to encourage learners to demonstrate a variety of skills, such as mathematical and computational thinking skills. Throughout the experiment, the learners were given access to various scaffolds. First, in the classroom format, learners could access the video lessons whenever they wanted. Then, learners may review the accompanying lecture slides; and end the learning session with the MATLAB® solutions that were shared in Blackboard.

In addition, Si et al. (2014) utilized an online environment to explore the efficacy of adaptation instructions based on two designs. These designs were bimodal process-oriented worked examples and unimodal process-oriented worked examples. Both of these designs were based on worked examples. The subject matter covered was the C programming language, and the primary topics covered were algorithm design in terms of task completion, if statements, and compound if statements (Si et al., 2014). Results showed that the adaptive instruction technique and the bimodal process-oriented WEs effectively managed cognitive load, resulting in schema creation and successful and efficient automation (Si et al., 2014).

While Boldbaatar and Endurur (2019) explored the influence of worked examples in

providing guidance for creating 3D games using a block-based programming environment. In this experiment learners were encouraged to create programmed 3D games via StarLogo TNG-a game-making tool- and following a series of guiding worked examples in solving problems. The integration of worked examples, completion examples, and full practice had a positive impact on learners by enhancing their computational thinking skills (Boldbaatar & Endurur, 2019).

In addition, Davidovic et al. (2003) used Stanford University's OpenEdX platform to facilitate learning experiences and provided programming instruction, including worked examples and other supplemental learning resources. The outcomes of the study showed that it was evident that learners benefitted from being provided an opportunity to demonstrate their programming knowledge in a variety of different supported methods (Davidovic et al., 2003).

Moreover, Hsu et al. (2012) presented a learning environment with two different screens so that numerous learning items could be displayed at the same time. The purpose of this research is to compare the cognitive demands placed on learners within interacting with single- and dual-screen learning environments in terms of how well they are able to retain information (Hsu et al., 2012). Results showed that utilizing worked examples was shown to have a considerable influence on decreasing the cognitive load experienced by learners in learning programming via interactive environments (Hsu et al., 2012).

Medium: Web pages

To promote computational thinking skills, Nainan and Balakrishnan (2019) designed worked examples via "web technology (HTML5, CSS3 and JavaScript)" (p.33). The interactive web page contained four sections of problems and worked examples. The websites utilized highlighting as a way to change the background color of different portions of the information "whenever the user moves the mouse over a subproblem in the analysis section" (Nainan &

Balakrishnan, 2019, p.34).

In addition, in Abdul-Rahman and Du Boulay (2014) study, researchers applied two formats when developing worked examples: a completion strategy and a structure-emphasizing strategy. They used the CORT interface that consists of two sections; left and the right. The right section has the worked examples as a structure-emphasizing strategy, and the left section has lines of missing code algorithms to be completed.

Supportive tools: learning Applications

The study conducted by Margulieux et al. (2016) is another example of the effective involvement of worked examples to promote computational thinking skills. In this study, the researchers employed a subgoal design of worked examples to enhance programming skills in the process of creating apps via App Inventor. The findings demonstrated that the subgoal intervention might be beneficial in an advanced field like programming, in an interactive context like an online environment, and for K-12 teachers (Margulieux et al., 2016).

In parallel to the previous study, Margulieux and Catrambone (2021) investigated whether or not subgoal labels followed by self-explanation of worked examples could be used to support early problem-solving in order to increase performance when addressing newer problems. The researchers used Android App Inventor- a drop-and-drag programming language that can be utilized to create applications (apps) for Android devices.

Supportive tools: Tutorials

Lee and Hong (2017) designed interactive worked examples as tutorials following multimedia learning principles in association with applying self-explanation principles to enhance learner's programming skills. While in Cevahir et al. (2022) study, researchers examined the impact of using animation-based worked examples (ARAWEs) tutorials designed

with Augmented Reality (AR) technology as opposed to employing "traditional paper worked examples (TWEs)" (p.226) on "academic achievement, motivation, and attitude" in a programming course (p.226). Similarly, Chang et al. (2000) developed a programming tutorial system to offer worked examples based on the completion technique in order to improve computational thinking skills.

Supportive tools: specific designed tools

PCEX was developed by Hosseini et al. (2020), and it is a tool that has been specifically built to teach Python programming through the use of both interactive and non-interactive learning material, as well as worked examples. The findings of this study pointed to the beneficial effects of interactive worked examples on the levels of engagement, problem-solving skills, and overall learning that learners experience. While Garner (2007) used a code restructuring tool, CORT - a support system - based on worked examples to enhance coding and problem-solving skills in a programming context. The results showed a positive impact of using *CORT* as a support system to enhance learners' programming skills.

On the other hand, Jennings and Muldner (2021) designed and investigated the effect of using a Code Tracing (CT)-Tutor, a tool that combined providing coding instruction and tracing the learning process simultaneously. Three levels of scaffolding based on worked examples were provided in this study, low, intermediate, and high level of assistance (Jennings & Muldner, 2021). The findings showed that learners spent less time reading the worked examples that CT-tutor provided than their programming performance time (Jennings & Muldner, 2021). Therefore, the study suggested that future intervention should encourage the learners to be more active with what they learn from the worked examples, like indicating self-explanation (Jennings & Muldner, 2021).

Due to meet the necessity of keeping learners engaged in the programming process, Toukiloglou and Xinogalos (2022) created a series of programming games, and they indicated worked examples in textual form to offer the required assistance. The findings demonstrated that the integration of worked examples and games had a substantial influence on the performance of novice programmers (Toukiloglou & Xinogalos, 2022).

Techniques

In addition to the medium of delivering the worked examples, and the supportive tools; the IR analyzed several studies discussing techniques integrated with worked examples. These approaches have been proved to be successful in the adoption of worked examples, and some of them have been described in the answers to questions 1, 2 and 3. The researcher classified the techniques themes into: application (the way of applying and implementing the WEs, visual design (integrating visuals with WEs), and worked example design (developing WEs in certain forms).

Techniques: application. The IR analyzed one study (n=1) that discussed the application of Jonassen's (1997) instructional design model for well-structured problems. Uysal (2014) followed an instructional design model that is specially focused on designing instruction in well-structured problems. In this investigation, seven steps have been implemented and it included presenting worked examples in the fourth steps, and it followed by practice to enhance the learning process (Uysal, 2014). The findings of following this approach resulted in a positive impact on the academic performance in addition to a significant problem solving differences between the two groups of the quasi-experiment (Uysal, 2014). It is essential to point out that the methodology utilized in this investigation was based on Jonasssen's (1997) model for the delivery of programming instruction. This procedure included the presentation of worked

examples as a component of the assistance provided for participants in the process of learning programming (Uysal, 2014). The instructional design model of well-structured problems is associated with offering instruction only in the context of well-structured problems (Jonasssen, 1997).

Techniques: visual design. The IR analyzed several studies that discussed the impact of integrating visuals either diagrams, images, or animations with worked examples in promoting computational thinking skills. Several studies have already been examined in response to questions 1, 2 and 3, addressing the possible designs of worked examples, the nature of these studies' relationship with instructional design principles, and the factors that determine the effectiveness of worked examples. For instance, using images and animations in the process of presenting worked examples had a strong effect on decreasing the cognitive load associated with learning how to solve programming problems effectively (Cevahir et al., 2022; Chang et al., 2011; Hsu et al., 2012; Hosseini et al., 2020; Vieira et al., 2018). In addition, presenting visual designs with worked examples had a positive influence on the attitude toward learning programming courses as well as high motivation, and improved problem-solving skills (Cevahir et al., 2022; Chang et al., 2011; Hsu et al., 2012; Hosseini et al., 2020; Vieira et al., 2018). In addition to animations, the employment of flowcharts has been shown to improve learners' abilities to design and construct effective algorithms for the goal of solving programming problems (Vieira., 2015).

Techniques: worked example design. Various studies examined numerous designs of worked examples that demonstrated their efficacy in parallel to some designs that were addressed before while answering the first question. For instance, several design formats have

been employed to present effective worked examples including subgoals, fading, sequences of WEs, and completion WEs.

The subgoal design of worked examples implies demonstrating the phases (as subgoals) of addressing the solution of a particular problem (Abdul-Rahman & Du Boulay, 2014; Berssanette & de Francisco, 2021; Nainan & Balakrishnan, 2019). According to the literature, subgoal design has been shown to be useful in offering instruction and improving learning outcomes, particularly in computer-based contexts (Catrambone, 1995; Margulieux et al., 2016). The current IR analyzed several studies where subgoal design of worked examples were experienced to promote computational thinking skills and enhance the programming performance (Bunch, 2009; Chang et al., 2000; Davidovic & Trichina, 2003; Margulieux & Catrambone, 2021; Moreno, 2009).

Moreover, the IR analyzed another classification of effective techniques where studies discussed the impact of employing the design of subgoal worked examples on learning programming. Renkl et al. (2004) stated that:

> In order to facilitate the transition from learning from worked examples in earlier stages of skill acquisition to problem solving in later stages, it is effective to successively fade out worked solution steps – in comparison to the traditional method (p.59).

Additionally, the design of faded worked examples has many advantages, for instance, the purpose of the faded worked example is to assist learners in remembering what they have learned in earlier stages so that they may apply it to complete a task (Garner, 2007; Margulieux et al., 2020; Si et al., 2014). Likewise, the probability of making fewer mistakes is one of the many benefits of fading (Boldbaatar & Şendurur, 2019).

Sequencing worked examples and presenting them in pair forms is another effective method to present worked examples. The current IR analyzed two sequencing sets that have been utilized in designing several quantities of worked examples, 1) worked examples with multiple scenarios, and 2) complete-partial worked examples. Studies showed that presenting varied amounts of examples could have a positive impact on developing problem solving skills as well as improving programming performance (Abdul-Rahman & Du Boulay, 2014; Bunch, 2009; Nainan & Balakrishnan, 2019; Si et al., 2014). Furthermore, it is a common practice in programming environments to provide a sequence of worked examples to enhance learners' computational thinking skills (Bunch, 2009; Nainan & Balakrishnan, 2019; Si et al., 2014).

Overall, designing worked examples in these formats resulted in a positive impact in learning programming, enhancing computational thinking skills, and reducing the cognitive load.

The following table (Table 7) provides an overview of these studies and their applications.

Table 7

Examples of effective techniques integrated with worked examples.

The technique	Theme	The study	Description	Programming area/ computational thinking skills	Programming language	The effect
Jonassen's (1997) Instructional Design Model for Well-Structured Problems.	Application	(Uysal, 2014).	Researcher followed Jonassen's (1997) IDT model for well-structured programming instruction. Embedding worked examples to enhance the programming skills in C programming course.	-algorithm design.	-C programming	Positive effect on the learner's performance in the treatment group.
Animation with worked examples	Visual design	(Cevahir et al., 2022; Chang et al., 2011; Hsu et al., 2012; Hosseini et al., 2020; Vieira et al., 2018).	Researchers integrated animations' elements within the construction of worked examples.	-Programming skills -algorithm design. -abstract computational thinking skill.	-C/C++ advanced programming -Fundamental programming course. -Python.	-Positive effect on the learner's performance in the treatment group. -Fewer intrinsic and extraneous cognitive loads. -Positive attitudes towards programming. -High motivation. -Increased learners' engagement. -Increased problem-solving skills.

Visuals and flowchart	Visual design	(Vieira., 2015).	Researchers used a mix of worked examples of coding solutions in association with a flowchart of an algorithm design.	-algorithm design.	-C programming	-Significant impact on novice programmers' performance.
Multiple formats of WEs (subgoals, fading, sequences of WEs, completion WEs)	WEs design	(Abdul-Rahman & Du Boulay, 2014; Berssanette & de Francisco, 2021; Boldbaatar & Şendurur, 2019; Bunch, 2009; Chang et al., 2000, Davidovic & Trichina, 2003; Garner, 2007; Si et al., 2014; Nainan & Balakrishnan, 2019; Margulieux et al., 2020; Margulieux & Catrambone, 2021; Moreno, 2009)	Researchers had several designs of worked examples that proved their effectiveness in providing programming instruction and enhancing computational thinking skills. Like, process-oriented worked examples, sequencing several forms of problem scenarios, completion WEs, subgoal WEs, fading.	-algorithm design. -data collection. -data analysis. -Evaluation skill. -abstract skill. -decomposition.	-Fundamental programming course. -C programming -introductory database. -BASIC Language -Drag-and-drop languages	-Positive effect on the learner's performance in the treatment group. -Fewer intrinsic and extraneous cognitive loads.

Summary of research question #4.

Based on the IR, around 78% of the collected studies provided several successful examples of employing worked examples to promote computational thinking skills. The researcher classified the examples into several categories including: 1) medium, 2) supportive tools, and 3) techniques. Medium concentrates on the delivery system where the worked examples are presented, and it includes subcategories as online learning environments and web pages. While the supportive tools concentrate on the utilized tools that presented or provided additional support with the worked examples, and it includes subcategories as learning applications, tutorials, and specific designed tools. In addition, the Techniques concentrate on strategies that have been integrated with worked examples to promote the computational thinking skills, and it includes subcategories as faded worked examples, subgoal design of worked examples, and sequencing of worked examples as completed and partial.

The context where those studies applied was varied including several courses like, C programming, C++ advanced, Fundamentals of programming course, BASIC language, introductory to database, Python programming. In addition to the variations of the courses, several skills were promoted like algorithm design, data collection, data analysis, abstract, decomposition skill. Beside improving learner's programming performance, reducing the intrinsic and extrinsic cognitive load was one major impact of utilizing worked examples in programming settings (Abdul-Rahman & Du Boulay, 2014; Nainan & Balakrishnan, 2019; Margulieux et al., 2020; Margulieux & Catrambone, 2021).

Summary of chapter four

The current IR seeks information regarding the effective design of worked examples in programming settings, the most suitable instructional design principle to employ in the design, factors and circumstances that may have an influence on the effectiveness of the design, and to find experiences of emptying worked examples in programming settings. This IR aims to provide information that may assist instructional designers and educators to provide the appropriate design of worked examples to enhance learner's computational thinking skills.

The IR reviewed 42 articles drawn from a variety of sources and employed a variety of research approaches. The findings demonstrated that the majority of the gathered data, which accounted for around 72% of the total, discussed the various forms and styles of worked examples. Animated worked examples, process-oriented worked examples, sequenced worked examples, subgoal design, tutorials as worked examples, fading worked examples, and in-game worked examples are all included in the design and formats.

While 50% of the collected studies discussed the most appropriate instructional design principles to design effective worked examples in well-structured programming problems. The instructional design principles vary in their nature and include multimedia learning principles, fading effect, completion effect, split-attention effect, and self-explanation effect.

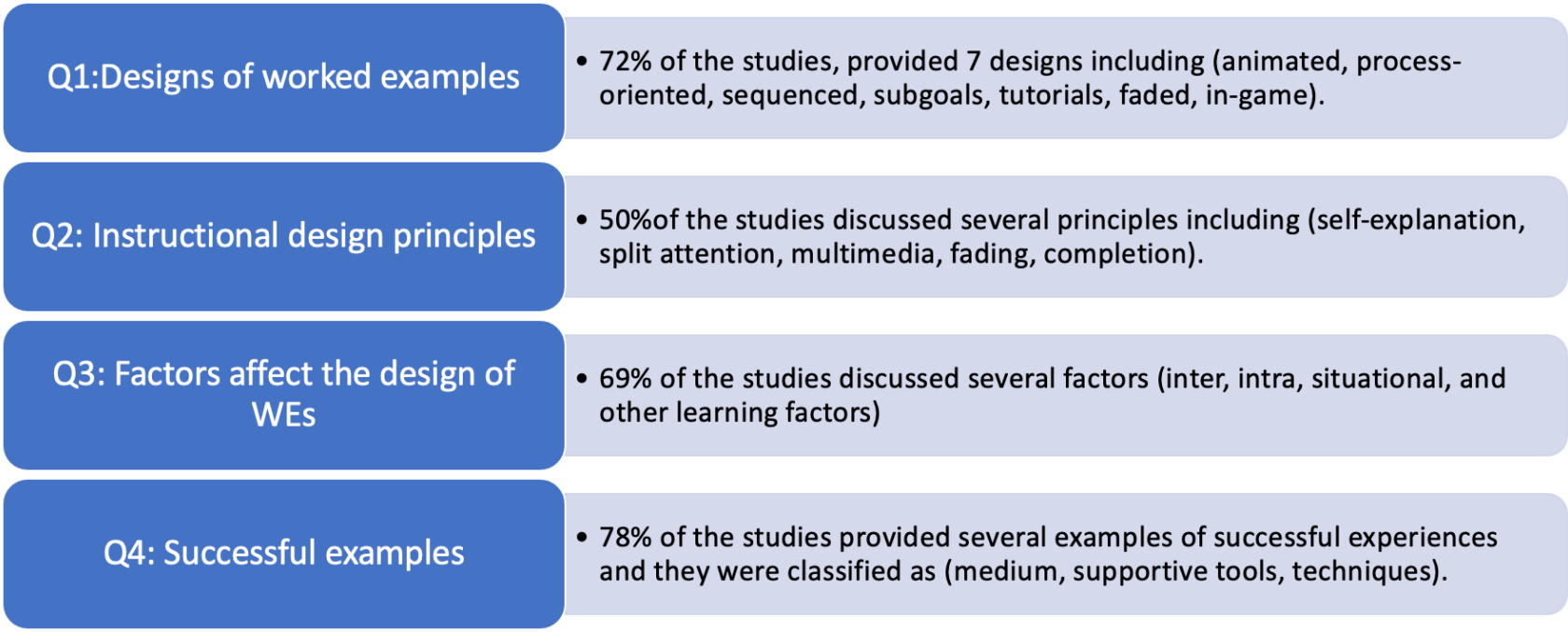
In regard to the factors and circumstances that may influence the design of worked examples, the IR analyzed 69% of the studies that have discussed those aspects. Accordingly, surface features, several forms of WEs, the use of multiple modalities, interaction with WEs, learners' learning style and background, teaching style, and the subject nature are all comprised to be important factors that influenced the effectiveness of worked examples.

The last research questions revealed information regarding the successful implementation of worked examples in enhancing computational thinking skills. 78% of the studies implemented worked examples in programming settings effectively with several applications including developing worked examples via online environments, delivering worked examples via web pages, integrating support systems by Apps, using tutorials, using visualizations, using innovative tools. In addition to the association with several techniques to design worked examples in effective forms, like following instructional design models, following visual design aspects, and designing several forms of worked examples.

The researcher designed the following figure (Figure 7) to provide a comprehensive view of the fourth chapter discussion.

Figure 7

An overview of the research questions discussions.



Chapter Five

Discussion and Conclusion

This chapter includes a discussion of the findings of the existing integrative literature review. This study was conducted with the intention of providing a comprehensive analysis of how worked examples may be effectively designed to enhance computational thinking skills within the context of a programming environment. The study followed the integrative literature review for this purpose. According to Russell (2005) integrative review has several advantages entail:

- evaluating the strength of the scientific evidence,
- identifying gaps in current research,
- identifying the need for future research,
- bridging between related areas of work,
- identifying central issues in an area,
- generating a research question,
- identifying a theoretical or conceptual framework,
- and exploring which research methods have been used successfully (p.1).

The rapid rise of technologies and scientific discoveries, alongside the amount of data created each day, has grown exponentially, revealing the need to focus more on how to accommodate our knowledge in this digital era (Israel-Fishelson & Hershkovitz, 2022). Besides, the ability to think computationally (CT) is now seen as crucial for success in today's modern world (Bocconi et al., 2016; DeSchryver & Yadav, 2015). As a result, worked examples are taken into consideration as one of the support techniques that may be utilized to develop

computational thinking skills within the context of programming (Falkner & Falkner, 2018, Jacob et al., 2020, Jocius et al., 2020).

All worked examples include a problem statement followed by the corresponding solution, often accompanied by detailed explanations of the processes to reach that conclusion (Renkl, 2011). Furthermore, worked examples are regarded as effective instructional strategies for "well-structured "subject areas (Renkl, 2012, p.1197).

Accordingly, the current research by following integrative review aimed to finding answers to the following questions:

1. Based on the literature, how have worked examples (involving computational thinking skills) been designed?
2. What instructional design principles have been recognized in the literature and need to be considered when designing WEs to promote CT skills?
3. What factors or circumstances might have an impact on the development of WEs for well-structured programming problems?
4. What are some examples of effective WEs for well-structured programming problems that have been identified in the literature?

Teaching students how to code is one of the most difficult aspects of computer science education (Mangaroska et al., 2021). As a consequence, the development of a support system that guides learners in developing and improving their computational thinking skills is essential (Seneviratne, 2017). From instructional design perspective, a worked example is one strategy that may assist learners in gaining skills in well-structured domains as a programming domain (Moore et al., 2020).

Furthermore, to ensure that worked examples are utilized well as a supportive technique, we have to examine how to develop them appropriately (Nainan & Balakrishnan, 2019).

Therefore, this study aimed to provide a comprehensive overview of how to design effective worked examples to promote computational thinking skill in well-structured programming problems. The purpose of this study also includes, providing information about the appropriate instructional design principles to follow when designing worked examples in programming context. In addition, it discussed the factors that may influence the effectiveness of worked examples when prompting computational thinking skills. It also referred to the literature with successful examples of the application of worked examples in enhancing computational thinking skills.

Discussion of Findings

The findings of the integrative literature analysis conducted for the current study revealed a total of 42 previous investigations conducted in various countries and utilizing several research methodologies. The majority of the studies were conducted in the USA, while Quasi-experiments were the most employed research methodology. Studies were also conducted in the UK, Turkey, Taiwan, China, Netherlands, South Korea, Greece, Canada, Australia, Malaysia, Brazil, and Saudi Arabia. Besides quasi-experiments, mixed methods, systematic reviews, and literature review discussions were parts of the employed methodologies.

According to the findings, several programming languages and courses were studied as a part of the interventions to promote computational thinking skills, including Windows programming, C programming language, introduction to programming languages, StarLogo Blocks, JavaScript, MATLAB, Python, Snap!, thermodynamics course, and Blockly.

The first research question is "*Based on the literature, how have worked examples (involving computational thinking skills) been designed?"*. This research question aims to discover methods for creating worked examples in a programming context that are distinct from those used in other disciplines and require simply paper and a pen. Even though programming necessitated interaction with computer devices, the research advocated exploring suitable designs that could be used in digital contexts. The IR showed several designs of worked examples that are employed to promote computational thinking skills in programming context. The designs include animated worked examples, process-oriented worked examples, sequenced worked examples, subgoal design, tutorial as worked examples, faded worked examples and in-game worked examples.

Each design incorporates certain elements or features, such as animated worked examples that are constructed with the use of animations (Cevahir et al., 2022; Chang et al., 2011; Hsu et al., 2012). In this design, worked examples with animated components demonstrate and visualize how the algorithm is developed (Hosseini et al., 2020). Process-oriented is another unique design of worked examples, and it has been found with one study. The research showed that worked examples in this design consist of a series or several examples followed by additional explanations of the process (Si et al., 2014). While sequence worked examples consist of providing several numbers of worked examples. According to the literature, it is suggested to present this design form of worked examples in disciplines such as computer science to increase programming skills more effectively (Nainan & Balakrishnan, 2019; Si et al., 2014).

As part of sequencing worked examples, research offered another design that based on providing sub-problems of the problematic task. The IR found that subgoals were widely employed to improve computational thinking skills (Berssanette & de Francisco, 2021;

Margulieux et al., 2020). Also, worked examples as tutorials appeared frequently as an example of designing worked examples. It has been noticed that several studies utilized "video modeling" - either live video or recorded video- as a way to demonstrate the coding process technique and considered this technique as providing an example of the programmed solution (Kale et al., 2018; Grover, 2017).

Moreover, a fading worked example was another design that has been found in the literature. At the beginning of applying this design, learners receive a completed worked example solution that is gradually fading to encourage learners to think about the solution (Moreno et al., 2009). Also, the research found another feature that helps significantly in block-based programming environments in merging the gaming environment with offering guidance of worked examples (Toukiloglou & Xinogalos, 2022).

Yet, it has been discovered that developing worked examples and implementing them through a programming environment necessitates paying close attention to how they are designed. As a result, the researchers suggest that numerous instructional design principles be addressed while constructing worked examples. For instance, the IR found that studies with the application of animated worked examples recommended more considerations of instructional message design principles (Cevahir et al., 2022, Chang et al., 2011).

Besides, indicating the use of visuals or animations required paying attention to the design of those visuals in addition to considering the effective design of worked examples as part of the instructional strategy. Following process-oriented design is particularly required considering message design principles and the modality effect (Si et al., 2014). Research also found recommendations associated with providing additional assistance while presenting worked examples, like indicating the use of hints and clues (Abdul-Rahman and Du Boulay, 2014;

Chang et al, 2000; Garner,2007). Furthermore, the studies focused on enhancing the algorithm design and the ability to complete the code successfully with the presence of worked examples.

In addition, the first research question and its recommendations served as an efficient guide for investigating the second research question, which was linked to the instructional design principles.

The second research question is "*What instructional design principles have been recognized in the literature and need to be considered when designing WEs to promote CT skills?"*. The second research question seeks information on the most common instructional design principles used to create worked examples in programming environments. The research looked into the techniques and how each study adopted each instructional design principle. Several instructional design principles that have been followed in developing worked examples to promote computational thinking. Principles include completion, fading, process, presentation, media, timing, self-explanation (Shen & Tsa, 2009), in addition to split-attention.

It has been observed that the majority of designed worked examples adhere to self-explanation principles. In programming context, the suitable application of following self-explanation principles is to indicate the code-comment feature where some programming context allowed for this adoption (Vieira et al., 2019). However, for this implementation other studies incorporate designing a space for adding comments and explanations in their programming environments (Nainan & Balakrishnan, 2019).

Multimedia learning principles are another important principle to be followed in designing worked examples, specifically in programming context. The research discussed temporal contiguity, modality, spatial, coherence, redundancy, image, personalization principles (Hsu et al., 2012; Lee & Hong, 2017; Si et al.., 2014). Integrating animations, visuals or tutorials

required paying attention to the multimedia learning principles in addition to split-attention principles. The idea of split-attention concentrates on the manner in which several items are constructed as well as the relationships between them. The research found that the nature of the studies that followed split-attention in their design, relied on the adoption of several instructional methods in providing instruction. Specifically, when comparing the effectiveness of worked examples with visuals/animations or tutorial and traditional worked examples with words only (Berssanette & de Francisco, 2021; Chang et al., 2011; Vieira et al., 2015).

In addition, fading is associated with the same technique of faded worked examples considered as a significant design principle. According to the research findings, there are two distinct forms of fading: forward fading and backward fading. Both of these methods have been proven to be beneficial in terms of improving computational thinking skills (Boldbaatar & endurur, 2019; Vieira et al., 2021). It has been shown that the success of developing worked examples in improving learning outcomes may be influenced by the degree of programmed problem complexity.

Additionally, completion principle which is focusing on how to design uncompleted or completed parts of the worked examples and enabling the learner to interact with the instruction and complete the missing parts.

The research showed a variety of instructional design principles within worked examples to promote computational thinking skills with different applications. Most of the studies associated the instructional design principles with similar design format categories. For instance, fading design principle with faded worked examples, and visualized worked examples with multimedia learning principles. This consideration enhanced the importance of instructional design as a theoretical foundation to develop several forms of worked examples.

The third research question is "*What factors or circumstances might have an impact on the development of WEs for well-structured programming problems?".* The purpose of this research question was to explore a range of factors or circumstances that have the potential to impact the efficacy of worked examples within a programming setting. The IR showed a specific category of factors that influenced the effectiveness of worked examples including a) intra-example features, b) inter-example features, and c) environmental aspects (Atkinson & Renkl, 2007; Atkinson et al., 2000). According to the findings, the impact of intra-example features is associated with the integration of input data with diagrams or aural content, labeling problems, subgoals, and fading solution phases (Atkinson et al., 2000). It was seen with the adoption of using multiple forms of worked examples representations, like visuals, flowcharts, descriptions, simulations, tutorials, and several forms of the problems (Bunch, 2009; Chang et al., 2011; Hsu et al., 2012; Lee & Hong, 2017; Si et al., 2014; Vieira et al., 2015).

While the impact of inter-example features is associated with the relations between various examples and problems' solutions (Atkinson et al., 2000). It was seen through the implementation of employing multiple designs at once. For instance, using multiple visual slides, complete/incomplete, subgoals worked examples (Hsu et al., 2012). The third category is associated with situational features that may influence the design worked examples, such as pre-training, social interaction, self-explanation. (Atkinson et al. 2000).

In addition, the research encountered several additional circumstances that were anticipated to have an impact on the creation of worked examples. Prior knowledge of the learner, preferences of the learner's learning style, and preferences of the learner's programming language are regarded to be essential aspects and design circumstances that are impacting the efficacy of worked examples.

The last research question is "*What are some examples of effective WEs for well-structured programming problems that have been identified in the literature?*". The purpose of this research question is to learn more about examples of the valuable utilization of worked examples in fostering computational thinking skills. The IR classified the findings into several categories, including successful examples of using medium, supportive tools, techniques.

The medium elements for delivering worked examples of programming instruction vary between online learning environments (Moodle, Blackboard, 3D game environment), webpages (Boldbaatar & Endurur, 2019; Si et al., 2014; Vieira et al., 2021; Wang & Hwang, 2017).

The supportive tools include App inventor, COde Restructuring Tool (CORT), and tutorials (Abdul-Rahman and Du Boulay, 2014; Cevahir et al., 2022; Garner, 2007; Lee and Hong, 2017; Margulieux & Catrambone, 2021; Margulieux et al.; 2016; Nainan & Balakrishnan, 2019).

The techniques contain the application of various themes like following the application of IDT model *Jonassen's (1997) Instructional Design Model for Well-Structured Problems*, in designing a full environment to provide programming instructions with indicating worked examples online (Uysal, 2014). In addition, the adoption of integrating worked examples with visual design that include using animations and visuals with worked examples (Cevahir et al., 2022; Chang et al., 2011; Hsu et al., 2012; Hosseini et al., 2020; Vieira et al., 2018).

Besides, worked examples designs include presenting multiple formats of worked examples as subgoals, faded, several sequences of worked examples, and completion worked examples (Abdul-Rahman & Du Boulay, 2014; Berssanette & de Francisco, 2021; Boldbaatar & Şendurur, 2019; Bunch, 2009; Chang et al., 2000, Davidovic & Trichina, 2003; Garner, 2007; Si

et al., 2014; Nainan & Balakrishnan, 2019; Margulieux et al., 2020; Margulieux & Catrambone, 2021; Moreno, 2009).

The study highlighted the numerous consequences of constructing worked examples and their beneficial effects on the learning process. It also revealed that the efficient use of worked examples resulted in a considerable reduction in intrinsic and extrinsic cognitive load. The study also aimed to illustrate the practical consequences by including screenshots of several experiences, demonstrating the wide range of valuable applications and designs.

Furthermore, the answer to the fourth research question provides a comprehensive wrap-up of all the results of the previous questions. This wrap-up begins with the design of worked examples, proceeds with the discussion of important factors and instructional design principles, and concludes with referring to a significant application of either a particular experiment or a combination of successful designs.

Implications of the Study

The findings of this research may be significant to computer science instructors and instructional designers who are interested in fostering computational thinking skills and improving programming skills among their students. In light of the nature of the environment in which programming occurs, it is vital to consider how learners might be supported and guided while they are interacting with the instruction that has been written digitally. Providing the focus of how to develop worked examples for programming purpose, may benefit not only the instructors and their learners but also instructional designers.

In terms of designing effective worked examples, the study discussed several designs and applications of worked examples in programming context. The study includes several examples of worked examples designing forms that have empirical proof of its effectiveness. As an

outcome, CS instructors and instructional designers may be encouraged to create and build other designs in addition to those presented in this study, or to investigate the efficacy of other alternative designs. In addition, the findings of the study can provide instructors and instructional designers with information on the appropriate design features and context to enable proper utilization of these worked examples.

Furthermore, following instructional design principles played a significant role in creating effective learning experiences. From an instructional design perspective, adhering to the theoretical components of design is extremely necessary to ensure that the learning goals will be accomplished. The study discussed several instructional design principles that have been employed to design worked examples for programming purposes. That may benefit instructional designers and instructors while they create the desired materials, and it may also motivate them to improve their knowledge via the application of other possible principles.

In order to build effective worked examples, it is necessary to follow the principles of instructional design, which require designers to take into account the significance of the aspects and conditions that may influence their creation. According to the findings of the study, there are a number of factors and conditions that have an impact on the design. Whether such factors are connected to the surface design of the worked examples, the relationship among several examples, or other contextual aspects such as learner knowledge, the designer must identify them to achieve the intended design.

This study could be relevant for instructional designers and instructors who are ready to adopt the successful application of worked examples in programming context. Specifically with observing the multiple successful examples of implementing those worked examples to promote computational thinking skills. The study identified numerous prior experiences with a wide

variety of methods for the efficient adoption of worked examples in programming contexts. These findings may encourage instructors and instructional designers to either continue utilizing the same strategies or come up with new interventions.

Limitations of the Study

Even though the present integrative literature review involves a comprehensive examination of designing worked examples in well-structured programming problems to promote computational thinking skills, it has several limitations. First, some innovative and effective interventions were published as conference papers, which exceeded the data collection's inclusion criteria, resulting in a limitation in the diversity of interventions.

Second, although multiple databases were utilized to gather the data, certain publications have a separate subscription with the academic institutions. As a consequence, the search for individual articles and the verification of whether or not they match the inclusion requirements required extra time and effort.

Third, due to the fact that the full text was accessible, a certain minimum number of the articles needed to have a particular request made through the library loan system. This, once again, required extra time and effort in order to incorporate those articles into the data collection.

Fourth, the present integrative literature review primarily focuses on the programming domain. Rather than focusing on the programming domain, several studies and interventions have investigated the process of developing worked examples in computerized settings. Examining some of these studies with the aim of future intervention might be beneficial to the design idea in general.

Recommendations for Future Research

To enhance computational thinking skills and to meet the demands of the modern era, educators should pay attention to how to provide the appropriate guidance and assistance to ensure reaching the desired learning outcomes.

The findings of this integrative literature review can be used to guide the adoption of several interventions in the future, including the exploration of various programming languages and diverse contexts. Additionally, a variety of other instructional design principles may be investigated with reference to how effectively they are applied in the worked examples design.

Moreover, instructional designers and instructors are able to research the prerequisites for such interventions and investigate the efficacy of merging several designing forms into a single model design. Also, automated worked examples are becoming increasingly popular. Therefore, it is essential for programming domains to investigate technological tools for designing and implementing worked examples in digital environments. As well as it is necessary for any future study interested in the adoption of computerized worked examples.

References

Abdul-Rahman, S. S., & Du Boulay, B. (2014). Learning programming via worked-examples: Relation of learning styles to cognitive load. *Computers in Human Behavior*, *30*, 286-298. https://doi.org/10.1016/j.chb.2013.09.007

Aggarwal, A., Touretzky, D. S., Gardner-Mccune, C., & 48th ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2017 48 2017 03 08 - 2017 03 11. (2017). Evaluating the effect of using physical manipulatives to foster computational thinking in elementary school. *Proceedings of the Conference on Integrating Technology into Computer Science Education, Iticse*, *9-14*, 9–14. https://doi.org/10.1145/3017680.3017791

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal*, *55*(7), 832–835. https://doi.org/10.1093/comjnl/bxs074

Akcaoglu, M. (2014). Learning problem-solving through making games at the game design and learning summer program. *Educational Technology Research and Development*, *62*(5), 583–600. https://doi.org/10.1007/s11423-014-9347-4

Alhassan, R. (2017). The Effect of Employing Self-Explanation Strategy with Worked Examples on Acquiring Computer Programing Skills. *Journal of Education and Practice*, *8*(6), 186-196.

Allaire, J. C., & Marsiske, M. (2002). Well- and ill-defined measures of everyday cognition: relationship to older adults' intellectual ability and functional status. *Psychology and aging*, *17*(1), 101–115. https://doi.org/10.1037/0882-7974.17.1.101

Altybayeva, A. A., & Wood, K. L. (2019). Application of designette practice in solving the challenges of design education in aerospace engineering. *AIAA Aviation 2019 Forum*, 3328. https://doi.org/10.2514/6.2019-3328

Alvarez, C., Rojas, L. A., & de Dios Valenzuela, J. (2022, June 26 – July 1). *Design and Evaluation of a Programming Tutor Based on an Instant Messaging Interface*. International Conference on Human-Computer Interaction, Virtual. https://doi.org/10.1007/978-3-031-05064-0_1

Anderson, J. R. (2015). *Cognitive psychology and its implications* (8th ed). Worth.

Angeli, C., & Giannakos, M. (2020). Computational thinking education: Issues and challenges. *Computers in Human Behavior*, *105*, 106185. https://doi.org/10.1016/j.chb.2019.106185

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A k-6 computational thinking curriculum framework: implications for teacher knowledge. *Journal of Educational Technology & Society*, *19*(3), 47–57. http://www.jstor.org/stable/jeductechsoci.19.3.47

Angevine, C. (2017, December 6). *Advancing computational thinking across K-12 education*. Digital Promise. https://digitalpromise.org/2017/12/06/advancing-computational-thinking-across-k-12-education.

Araujo, A. L. S. O., Santos, J. S., Andrade, W. L., Guerrero, D. D. S., & Dagienė, V. (2017, October). *Exploring computational thinking assessment in introductory programming courses*. 2017 IEEE Frontiers in Education Conference (FIE), Indianapolis, IN, United States. https://doi.org/10.1109/FIE.2017.8190652

Arlin, P. K. (1989). Problem solving and problem finding in young artists and young scientists. In M. L. Commons, J. D. Sinnott, F. A. Richards, & C. Armon (Eds.), *Adult development, Vol. 1. Comparisons and applications of developmental models*(pp. 197–216). Praeger Publishers.

Armoni, M. (2013). Designing a K--12 computing curriculum: the questions. *ACM Inroads*, *4*(2), 34-35. https://doi.org/10.1145/2465085.2465095

Atkinson, K. M., Koenka, A. C., Sanchez, C. E., Moshontz, H., & Cooper, H. (2015). Reporting standards for literature searches and report inclusion criteria: making research syntheses more transparent and easy to replicate. *Research synthesis methods*, *6*(1), 87-95. https://doi.org/10.1002/jrsm.1127.

Atkinson, R. K., & Derry, S. J. (2000, May). *Computer-Based Examples Designed to Encourage Optimal Example Processing: A Study Examining the Impact of Sequentially Presented, Subgoal-Oriented Worked Examples1*. Fourth international conference of the learning sciences, New York, NY, United States. https://doi.org/10.4324/9780203763865.

Atkinson, R. K., & Renkl, A. (2007). Interactive example-based learning environments: Using interactive elements to encourage effective processing of worked examples. *Educational Psychology Review*, *19*(3), 375-386. https://doi.org/10.1007/s10648-007-9055-2

Atkinson, R. K., Catrambone, R., & Merrill, M. M. (2003). Aiding Transfer in Statistics: Examining the Use of Conceptually Oriented Equations and Elaborations During Subgoal Learning. *Journal of Educational Psychology*, *95*(4), 762. https://doi.org/10.1037/0022-0663.95.4.762

Atkinson, R. K., Derry, S. J., Renkl, A., & Wortham, D. (2000). Learning from examples: Instructional principles from the worked examples research. *Review of educational research*, *70*(2), 181-214. https://doi.org/10.3102/00346543070002181

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems*, *75*, 661-670. https://doi.org/10.1016/j.robot.2015.10.008

Austerlitz, H. (2002). *Data acquisition techniques using PCs*. Academic press.

Ayres P. (2012) Worked Example Effect. In: Seel N.M. (eds) *Encyclopedia of the Sciences of Learning*. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-1428-6_20

Ayres, P., & Sweller, J. (2014). The split-attention principle in multimedia learning. In R. E. Mayer (Ed.), *The Cambridge handbook of multimedia learning* (pp. 206–226). Cambridge University Press. https://doi.org/10.1017/CBO9781139547369.011

Badeau, R., White, D. R., Ibrahim, B., Ding, L., & Heckler, A. F. (2017). What works with worked examples: Extending self-explanation and analogical comparison to synthesis problems. *Physical Review Physics Education Research*,*13*(2), 020112. https://doi.org/10.1103/PhysRevPhysEducRes.13.020112

Bakar, M. A., Mukhtar, M., & Khalid, F. (2019). The development of a visual output approach for programming via the application of cognitive load theory and constructivism. *Development*, *10*(11), 305-312. https://doi.org/10.14569/IJACSA.2019.0101142

Balanskat, A., & Engelhardt, K. (2014). *Computing our future: Computer programming and coding-Priorities, school curricula and initiatives across Europe*. European Schoolnet.

Bandura, A. (1986). *Social foundations of thought and action: Selected passages. A social cognitive theory*. Prentice Hall.

Baratè, A., Formica, A., Ludovico, L. A., & Malchiodi, D. (2017, April 21-23). *Fostering computational thinking in secondary school through music-an educational experience based on google blockly* [Paper presentation]. International Conference on Computer Supported, Porto, Portugal. https://doi.org/10.5220/0006313001170124

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community?. *Acm Inroads*, *2*(1), 48-54. https://doi.org/10.1145/1929887.1929905

Basogain, X., Olabe, M. A., Olabe, J. C., Ramírez, R., Del Rosario, M., & Garcia, J. (2016, September 13-15). *PC-01: Introduction to computational thinking: Educational technology in primary and secondary education* [Paper presentation]. 2016 International Symposium on Computers in Education (SIIE), Salamanca, Spain. https://doi.org/ 10.1109/SIIE.2016.7751816

Basu, S., Biswas, G., & Kinnebrew, J. S. (2017). Learner modeling for adaptive scaffolding in a computational thinking-based science learning environment. *User Modeling and User-Adapted Interaction*, *27*(1), 5-53. https://doi.org/10.1007/s11257-017-9187-0

Bautista, R. G. (2013). The Students' Procedural Fluency and Written Mathematical Explanation on Constructed Response Tasks in Physics. *Journal of Technology and Science Education*, *3*(1), 49-56.

BBC. (2018). *BBC—Introduction to computational thinking*. https://www.bbc.co. uk/education/guides/zp92mp3/revision/1.

Beege, M., Schneider, S., Nebel, S., Zimm, J., Windisch, S., & Rey, G. D. (2021). Learning programming from erroneous worked-examples. Which type of error is beneficial for learning?. *Learning and Instruction*, *75*, 101497. https://doi.org/10.1016/j.learninstruc.2021.101497

Berland, M., & Lee, V. R. (2011). Collaborative strategic board games as a site for distributed computational thinking. *International Journal of Game-Based Learning (IJGBL)*, *1*(2), 65-81. https://doi.org/10.4018/ijgbl.2011040105

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, *72*, 145-157. https://doi.org/10.1016/j.compedu.2013.10.020

Berssanette, J. H., & de Francisco, A. C. (2021). Cognitive Load Theory in the Context of Teaching and Learning Computer Programming: A Systematic Literature Review. *IEEE Transactions on Education*, *65*(3),440-449. https://doi.org/10.1109/TE.2021.3127215

Beyea, S., & Nichll, L. H. (1998). Writing an integrative review. *AORN journal*, *67*(4), 877-881. https://doi.org/10.1016/s0001-2092(06)62653-7

Binkley, M., Erstad, O., Herman, J., Raizen, S., Ripley, M., Miller-Ricci, M., & Rumble, M. (2012). Defining twenty-first century skills. In *Assessment and teaching of 21st century skills* (pp. 17-66). Springer, Dordrecht.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K., Kampylis, P., & Punie, Y. (2016). Developing computational thinking in compulsory education. *European Commission, JRC Science for Policy Report*, *68*. https://doi.org/10.2791/792158

Boldbaatar, N., & Şendurur, E. (2019). Developing Educational 3D Games With StarLogo: The Role of Backwards Fading in the Transfer of Programming Experience. *Journal of Educational Computing Research*, *57*(6), 1468–1494. https://doi.org/10.1177/0735633118806747

Bower, M., Wood, L. N., Howe, C., & Lister, R. (2017). Improving the Computational Thinking Pedagogical Capabilities of School Teachers. *Australian Journal of Teacher Education, 42*(3), 53–72. https://doi.org/10.14221/ajte.2017v42n3.4

Brabeck, M., & Wood, P. K. (1990). Cross-sectional and longitudinal evidence for differences between ill-structured and well-structured problem solving abilities. In *Adult development--volume 2: Models and methods in the study of adolescent and adult thought*. Praeger.

Branch, R. M., & Kopcha, T. J. (2014). Instructional design models. In *Handbook of research on educational communications and technology* (pp. 77-87). Springer, New York, NY.

Bråting, K., & Kilhamn, C. (2021). Exploring the intersection of algebraic and computational thinking. *Mathematical Thinking and Learning*, *23*(2), 170-185. https://doi.org/10.1080/10986065.2020.1779012

Brennan, K., & Resnick, M. (2012, April 13-17). *New frameworks for studying and assessing the development of computational thinking* [Paper presentation]. The 2012 annual meeting of the American educational research association, Vancouver, Canada.

Broome M.E. (1993). Integrative literature reviews for the development of concepts. In B.L. Rogers & K. Knafl (Eds.), *Concept Development in Nursing*, (2nd ed., pp. 231-250). W.B. Saunders Co.

Brown, N. C. C., Kölling, M., Crick, T., Peyton Jones, S., Humphreys, S., & Sentance, S. (2013, March 3-9). *Bringing computer science back into schools: Lessons from the UK* [Paper presentation]. The 44th ACM technical symposium on Computer science education, Denver, CO, United States.

Brusilovsky, P., & Yudelson, M. V. (2008). From webex to navex: Interactive access to annotated program examples. *Proceedings of the IEEE*, *96*(6), 990-999. https://doi.org/10.1109/JPROC.2008.921611

Bucci, P., Long, T. J., & Weide, B. W. (2001). Do we really teach abstraction?. *ACM SIGCSE Bulletin*, *33*(1), 26-30. https://doi.org/10.1145/366413.364531

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a generation's way of thinking: Teaching computational thinking through programming. *Review of Educational Research*, *87*(4), 834-860. https://doi.org/10.3102/0034654317710096

Bunch, J. M. (2009). Teaching Tip: An Approach to Reducing Cognitive Load in the Teaching of Introductory Database Concepts. *Journal of Information Systems Education*, *20*(3), 277.

Burke, Q., Bailey, C. S., & Ruiz, P. (2019). CIRCL Primer: Assessing Computational Thinking. In *CIRCL Primer Series*. http://circlcenter.org/assessing-computational-thinking.

Butcher, K. (2014). The Multimedia Principle. In R. Mayer (Ed.), *The Cambridge Handbook of Multimedia Learning* (Cambridge Handbooks in Psychology, pp. 174-205). Cambridge: Cambridge University Press. doi:10.1017/CBO9781139547369.010.

Butler, M., & Morgan, M. (2007, December 2-5). *Learning challenges faced by novice programming students studying high level and low feedback concepts* [Paper presentation]. Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education 2007, Nanyang Singapore, Singapore.

Cansu, F. K., & Cansu, S. K. (2019). An Overview of Computational Thinking. *International Journal of Computer Science Education in Schools*, *3*(1), 17–30. https://doi.org/10.21585/ijcses.v3i1.53

Casakin, H. (2002, September 5-7). *Well-defined vs. ill-defined design problem solving: The use of visual analogy* [Paper presentation]. *Common Ground - DRS International Conference*, London, United Kingdom. https://dl.designresearchsociety.org/drs

Caskurlu, S., Yadav, A., Dunbar, K., & Santo, R. (2021). Professional development as a bridge between teacher competencies and computational thinking integration. In *Computational Thinking in Education* (pp. 136-150). Routledge.

Caspersen, M. E., & Bennedsen, J. (2007, September 15-17). *Instructional design of a programming course: a learning theoretic approach* [Paper presentation]. The third international workshop on Computing education research, Atlanta, GA, United States. https://doi.org/10.1145/1288580.1288595

Caspersen, M. E., & Nowack, P. (2013, January). *Computational thinking and practice: A generic approach to computing in Danish high schools* [Paper presentation]. The Fifteenth Australasian Computing Education Conference, Adelaide, South Australia. https://doi.org/10.1145/1288580.1288595

Catrambone, R. (1995). Aiding subgoal learning: Effects on transfer. *Journal of educational psychology*, *87*(1), 5. https://doi.org/10.1037/0022-0663.87.1.5

Catrambone, R. (1996). Generalizing solution procedures learned from examples. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *22*(4), 1020-1031. https://doi.org/10.1037/0278-7393.22.4.1020

Catrambone, R. (1998). The subgoal learning model: Creating better examples so that students can solve novel problems. *Journal of experimental psychology: General*, *127*(4), 355-376. https://doi.org/10.1037/0096-3445.127.4.355

Cevahir, H., Özdemir, M., & Baturay, M. H. (2022). The Effect of Animation-Based Worked Examples Supported with Augmented Reality on the Academic Achievement, Attitude and Motivation of Students towards Learning Programming. *Participatory Educational Research*, *9*(3), 226-247. https://doi.org/10.1007/s10639-021-10625-w

Ch'ng, S. I., Low, Y. C., Lee, Y. L., Chia, W. C., & Yeong, L. S. (2019). Video games: A potential vehicle for teaching computational thinking. In *Computational Thinking Education*(pp. 247-260). Springer, Singapore.

Chandler, P., & Sweller, J. (1991). Cognitive load theory and the format of instruction. *Cognition and instruction*, *8*(4), 293-332. https://doi.org/10.1207/s1532690xci0804_2

Chang, K. E., Chiao, B. C., Chen, S. W., & Hsiao, R. S. (2000). A programming learning system for beginners-A completion strategy approach. *IEEE Transactions on Education*, *43*(2), 211-220.

Chang, T. W., Hsu, J. M., & Yu, P. T. (2011). A comparison of single-and dual-screen environment in programming language: Cognitive loads and learning effects. *Journal of Educational Technology & Society*, *14*(2), 188-200.

Chen, O., & Kalyuga, S. (2020). Cognitive load theory, spacing effect, and working memory resources depletion: Implications for instructional design. In *Form, function, and style in instructional design: Emerging research and opportunities* (pp. 1-26). IGI Global.

Chen, O., Kalyuga, S., & Sweller, J. (2016). When instructional guidance is needed. *The Educational and Developmental Psychologist*, *33*(2), 149-162. https://doi.org/10.1017/edp.2016.16

Chen, O., Retnowati, E., & Kalyuga, S. (2020). Element interactivity as a factor influencing the effectiveness of worked example–problem solving and problem solving–worked example sequences. *British Journal of Educational Psychology*, *90*, 210-223. https://doi.org/10.1111/bjep.12317

Chen, X., Mitrovic, A., & Mathews, M. (2019). Learning from worked examples, erroneous examples, and problem solving: Toward adaptive selection of learning activities. *IEEE Transactions on Learning Technologies*, *13*(1), 135-149.

Chi, M. T., Bassok, M., Lewis, M. W., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive science*, *13*(2), 145-182. https://doi.org/10.1207/s15516709cog1302_1

Chi, M. T. H., & Glaser, R. (1985). Problem-Solving Ability. *In R. J. Sternberg (Ed.), Human Abilities: An Information-Processing Approach* (pp. 227-257). San Francisco, CA: W H Freeman & Co. (Sd).

Chin, C., & Chia, L. G. (2006). Problem-based learning: Using ill-structured problems in biology project work. *Science Education*, *90*(1), 44-67. https://doi.org/10.1002/sce.20097

Chong, T. S. (2005). Recent Advances Cognitive Load Theory Research: Implications for the Instructional Designers. Recent Advances in Cognitive Load Theory Research. *Malaysian Online Journal of Instructional Technology (MOJIT)*, *2*(3), 106-117.

Chu, S. K. W., Reynolds, R. B., Tavares, N. J., Notari, M., & Lee, C. W. Y. (2021). *21st century skills development through inquiry-based learning from theory to practice*. Springer International Publishing.

Clark, R. C., & Mayer, R. E. (2016). *E-learning and the science of instruction: Proven guidelines for consumers and designers of multimedia learning*. john Wiley & sons.

Clark, R. C., Nguyen, F., & Sweller, J. (2011). *Efficiency in learning: Evidence-based guidelines to manage cognitive load*. John Wiley & Sons.

Clement, C., & Gentner, D. (1991). Systematicity as a selection constraint in analogical mapping. *Cognitive Science, 15*, 89-132. https://doi.org/10.1016/0364-0213(91)80014-V

Collins, A., Brown, J. S., & Newman, S. E. (1988). Cognitive apprenticeship: Teaching the craft of reading, writing and mathematics. *Thinking: The journal of philosophy for children*, *8*(1), 2-10.

Computer Science Teachers Association, & International Society for Technology in Education. (2011). *Computational Thinking: Leadership Toolkit (1st ed).* http://www.csta.acm.org/Curriculum/sub/CurrFiles/471.11CTLeadershiptToolkit-SP-vF.pdf

Cooper, G. (1990). Cognitive load theory as an aid for instructional design. *Australasian Journal of Educational Technology*,*6*(2). https://doi.org/10.14742/ajet.2322

Cooper, G., & Sweller, J. (1987). Effects of schema acquisition and rule automation on mathematical problem-solving transfer. *Journal of educational psychology*, *79*(4), 347-362. https://doi.org/10.1037/0022-0663.79.4.347

Cooper, H. (1998). Synth*esizing Research: A Guide for Literature Reviews*. Sage.

Cooper, H. (2015). *Research synthesis and meta-analysis: A step-by-step approach* (Vol. 2). Sage.

Cooper, H. (2017). Step 2: searching the literature. *In Research synthesis and meta-analysis* (pp. 61-109). SAGE Publications, Inc, https://dx.doi.org/10.4135/9781071878644.n3

Cooper, H. M. (1982). Scientific guidelines for conducting integrative research reviews. *Review of educational research*, *52*(2), 291-302. https://doi.org/10.2307/1170314

Cooper, H. M. (1988). Organizing knowledge syntheses: A taxonomy of literature reviews. *Knowledge in society*, *1*(1), 104-126. https://doi.org/10.1007/BF03177550

Coughlan, M., & Cronin, P. (2016). *Doing a literature review in nursing, health and social care*. Sage.

Coughlin, M. B., & Sethares, K. A. (2017). Chronic sorrow in parents of children with a chronic illness or disability: An integrative literature review. *Journal of Pediatric Nursing*, *37*, 108-116. https://doi.org/10.1016/j.pedn.2017.06.011

Crippen, K. J., & Earl, B. L. (2007). The impact of web-based worked examples and self-explanation on performance, problem solving, and self-efficacy. *Computers & Education*, *49*(3), 809-821. https://doi.org/10.1016/j.compedu.2005.11.018

Cross, N. (2021). *Engineering design methods: strategies for product design*. John Wiley & Sons.

Crowe, M., & Sheppard, L. (2011). A review of critical appraisal tools show they lack rigor: alternative tool structure is proposed. *Journal of clinical epidemiology*, *64*(1), 79-89. https://doi.org/10.1016/j.jclinepi.2010.02.008

Czerkawski, B. (2015, October). Computational thinking in virtual learning environments. In *E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education* (pp. 1227-1231). Association for the Advancement of Computing in Education (AACE).

D'Alba, A., & Huett, K. C. (2017). Learning computational skills in uCode@ UWG: challenges and recommendations. In *Emerging research, practice, and policy on computational thinking* (pp. 3-20). Springer, Cham.

Dabbagh, N. (2002). Assessing complex problem-solving skills and knowledge assembly using web-based hypermedia design. *Journal of educational multimedia and hypermedia*, *11*(4), 291-322.

Dart, S., Pickering, E., & Dawes, L. (2020). Worked example videos for blended learning in undergraduate engineering. *AEE Journal*, *8*(2).

Dasgupta, A., & Rynearson, A. M., & Purzer, S., & Ehsan, H., & Cardella, M. E. (2017, June), *Computational Thinking in K-2 Classrooms: Evidence from Student Artifacts (Fundamental)* [Paper presentation]. 2017 ASEE Annual Conference & Exposition, Columbus, Ohio. https://doi.org/10.18260/1-2—28062

Davenport, C. E. (2019). Using worked examples to improve student understanding of atmospheric dynamics. *Bulletin of the American Meteorological Society*, *100*(9), 1653-1664. https://doi.org/10.1175/BAMS-D-18-0226.1

Davidovic, A., Warren, J., & Trichina, E. (2003). Learning benefits of structural example-based adaptive tutoring systems. *IEEE Transactions on Education*, *46*(2), 241-251. https://doi.org/10.1109/TE.2002.808240.

Denney, A. S., & Tewksbury, R. (2013). How to write a literature review. *Journal of criminal justice education*, *24*(2), 218-234. https://doi.org/10.1080/10511253.2012.730617

Denning, P. J. (2009). The profession of IT Beyond computational thinking. *Communications of the ACM*, *52*(6), 28-30. https://doi.org/10.1145/1516046.1516054

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33-39. https://doi.org/10.1145/2998438

DeSchryver, M. D., & Yadav, A. (2015). Creative and computational thinking in the context of new literacies: Working with teachers to scaffold complex technology-mediated approaches to teaching and learning. *Journal of Technology and Teacher Education*, *23*(3), 411-431.

Dick, W., Carey, L., & Carey, J. O. (2001). *The systematic design of instruction* (6th ed). Longman.

Dijkstra, E. W. (1974). Programming as a discipline of mathematical nature. *The American Mathematical Monthly*, *81*(6), 608-612. https://doi.org/10.1080/00029890.1974.11993624

Dillon, S. M. (2002). *Understanding decision problem structuring by executives* [Thesis, Doctor of Philosophy (Ph.D.), The University of Waikato, Hamilton, New Zealand]. University of Waikato Achieve. https://hdl.handle.net/10289/14020

Dochy, F. (2006). A guide for writing scholarly articles or reviews for the Educational Research Review. *Educational Research Review*, *4*(1-2), 1-21.

Dondi, M., Klier, J., Panier, F., & Schubert, J. (2021, July 5). *Defining the skills citizens will need in the future world of work*. McKinsey & Company. https://www.mckinsey.com/industries/public-and-social-sector/our-insights/defining-the-skills-citizens-will-need-in-the-future-world-of-work

Driscoll, M. P. (2005). *Psychology of learning for instruction* (3rd ed.). Pearson Allyn and Bacon.

Duch, B. J. (2001). Writing problems for deeper understanding. *The power of problem-based learning*, 47-58. https://doi.org/10.12691/education-4-9-10

Dunbar, K. (1998). Problem solving. *In W. Bechtel, & G. Graham (Eds.). A companion to Cognitive Science*. London, England: Blackwell, pp 289-298.

Durak, H. Y. (2020). The effects of using different tools in programming teaching of secondary school students on engagement, computational thinking and reflective thinking skills for problem solving. *Technology, Knowledge and Learning*, *25*(1), 179-195.

Durak, H. Y., Yilmaz, F. G. K., & Yilmaz, R. (2019). Computational thinking, programming self-efficacy, problem solving and experiences in the programming process conducted with robotic activities. *Contemporary Educational Technology*, *10*(2), 173-197. https://doi.org/10.30935/cet.554493

Dyer, J. O., Hudon, A., Montpetit-Tourangeau, K., Charlin, B., Mamede, S., & Van Gog, T. (2015). Example-based learning: comparing the effects of additionally providing three different integrative learning activities on physiotherapy intervention knowledge. *BMC medical education*, *15*(1), 1-16. https://doi.org/10.1186/s12909-015-0308-3

Evans, D. (2007). Overview of methods. In Webb, C., & Roe, B. (Eds.). (2008). *Reviewing research evidence for nursing practice: Systematic reviews*. (pp 135-148). John Wiley & Sons.

Faber, H. H., Wierdsma, M. D. M., Doornbos, R. P., & van der Ven, J. S. (2017). Teaching computational thinking to primary school students via unplugged programming lessons. *Journal of the European Teacher Education Network, 12,* 13–24. https://etenjournal.com/2020/02/07/teaching-computational-thinking-to-primary-school-students-via-unplugged-programming-lessons/

Falkner, K., Vivian, R., & Falkner, N. (2018, April. 19-22). *Supporting computational thinking development in K-6* [Paper presentation]. 2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE), Auckland, New Zealand. https://doi.org/10.1109/LaTICE.2018.00-15

Fernández, J. M., Zúñiga, M. E., Rosas, M. V., & Guerrero, R. A. (2018). Experiences in learning problem-solving through computational thinking. *Journal of Computer Science and Technology*, *18*(2), 136-142. https://doi.org/10.24215/16666038.18.e15

Ferrari, R. (2015). Writing narrative style literature reviews. *Medical Writing*, *24*(4), 230-235. https://doi.org/10.1179/2047480615Z.000000000329

Ferreira, M. M., & Trudel, A. R. (2012). The impact of problem-based learning (PBL) on student attitudes toward science, problem-solving skills, and sense of community in the classroom. *Journal of classroom interaction*, *47*(1), 23-30. https://doi.org/10.12691/education-5-2-11

Figueiredo, J. A. Q. (2017). How to Improve Computational Thinking: a Case Study. [Cómo mejorar el pensamiento computacional: un estudio de caso] *Education in the Knowledge Society, 18*(4), 35-51. https://doi.org/10.14201/eks20171843551

Flanagan, J. (2018). The integrative review. *International Journal of Nursing Knowledge*, *29*(2), 81-81. https://doi.org/10.1111/2047-3095.12208

Fletcher, G. H., & Lu, J. J. (2009). Education Human computing skills: rethinking the K-12 experience. *Communications of the ACM*, *52*(2), 23-25. https://doi.org/10.1145/1461928.1461938

FLOYD, S. (2020, August 19-21). *Computational Thinkers: Contemporary Approaches and Directions in Computational Thinking for K-12 Education* [Paper presentation]. International Conference on Computational Thinking Education 2020, Hong Kong.

Forrest, S., & Mitchell, M. (2016). Adaptive computation: The multidisciplinary legacy of John H. Holland. *Communications of the ACM*, *59*(8), 58-63. https://doi.org/10.1145/2964342

French Academy of Sciences. (2013). *Teaching computer science in France, tomorrow can't wait*. http://www.academie-sciences.fr/pdf/rapport/ rads_0513gb.pdf

Gagné, R. M. (1985). *Conditions of learning and theory of instruction*. Holt, Rinehart and Winston.

Gagné, R. M., & Briggs, L. J. (1974). *Principles of instructional design* (2nd ed.). Holt, Rinehart and Winston.

Ganong, L. H. (1987). Integrative reviews of nursing research. *Research in nursing & health*, *10*(1), 1-11. https://doi.org/10.1002/nur.4770100103

Garces, S., Vieira, C., Ravai, G., & Magana, A. J. (2022). Engaging students in active exploration of programming worked examples. *Education and Information Technologies*, 1-18. https://doi.org/10.1007/s10639-022-11247-6

Garcia-Penalvo, F. J. (2016, September 14-16). *A brief introduction to TACCLE 3—coding European project* [Paper presentation]. The 2016 International Symposium on Computers in Education (SIIE), the University of Salamanca, Spain.

Garneli, V., Giannakos, M. N., & Chorianopoulos, K. (2015, March 18-20). *Computing education in K-12 schools: A review of the literature* [Paper presentation]. The 2015 IEEE Global Engineering Education Conference (EDUCON), Tallinn, Estonia.

Garner, S. (2002). *Reducing the cognitive load on novice programmers* (pp. 578-583). Association for the Advancement of Computing in Education (AACE).

Garner, S. (2007). An Exploration of How a Technology-Facilitated Part-Complete Solution Method Supports the Learning of Computer Programming. *Issues in Informing Science & Information Technology*, *4*. https://doi.org/10.28945/3127.

Gaweda, A. M., Lynch, C. F., Seamon, N., Silva de Oliveira, G., & Deliwa, A. (2020, February 3-7). *Typing exercises as interactive worked examples for deliberate practice in cs courses* [Paper presentation]. The Twenty-Second Australasian Computing Education Conference, Melbourne, Australia.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive science*, *7*(2), 155-170. https://doi.org/10.1016/S0364-0213(83)80009-3

Giannakos, M. N., Pappas, I. O., Jaccheri, L., & Sampson, D. G. (2017). Understanding student retention in computer science education: The role of environment, gains, barriers and usefulness. *Education and Information Technologies*, *22*(5), 2365-2382. https://doi.org/10.1007/s10639-016-9538-1

Gick, M. L. (1986). Problem-solving strategies. *Educational psychologist*, *21*(1-2), 99-120. https://doi.org/10.1207/s15326985ep2101&2_6

Gick, M. L., & Holyoak, K. J. (1980). Analogical problem solving. *Cognitive psychology*, *12*(3), 306-355. https://doi.org/10.1016/0010-0285(80)90013-4

Goel, V. (1992, July 29- August 1). *Ill-structured representations for ill-structured problems* [Paper presentation]. The fourteenth annual conference of the cognitive science society, Bloomington, IN, United States.

Goode, J., Chapman, G., & Margolis, J. (2012). Beyond curriculum: The exploring computer science program. *ACM Inroads*, *3*(2), 47-53. https://doi.org/10.1145/2189835.2189851

Google. (2018). *Google for education: Exploring computational thinking*. https://edu.google.com/resources/programs/exploring-computational-thinking/#!ct- overview.

Gorschek, T., Mendez, D. (2021). Solving Problems or Enabling Problem-Solving? from Purity in Empirical Software Engineering to Effective Co-production (Invited Keynote). *In: Winkler, D., Biffl, S., Mendez, D., Wimmer, M., Bergsmann, J. (eds) Software Quality: Future Perspectives on Software Engineering Quality.* SWQD 2021. Lecture Notes in Business Information Processing, vol 404. Springer, Cham. https://doi.org/10.1007/978-3-030-65854-0_9

Gouws, L. A., Bradshaw, K., & Wentworth, P. (2013, July 1-3). *Computational thinking in educational activities: an evaluation of the educational game light-bot* [Paper presentation]. The 18th ACM conference on Innovation and technology in computer science education, Canterbury England, UK.

Gralla, L., Tenbrink, T., Siebers, M., & Schmid, U. (2012). Analogical problem solving: Insights from verbal reports. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, *34*(34), 396-401.

Greeno, J. G. (1976). Indefinite goals in well-structured problems. *Psychological Review*, *83*(6), 479. https://doi.org/10.1037/0033-295X.83.6.479

Greeno, J. G., Collins, A. M., & Resnick, L. B. (1996). In DC Berliner, & RC Calfee (Eds.), *Cognition and learning. Handbook of educational psychology* (pp. 15–46). Macmillan Library Reference USA; Prentice Hall International.

Gretter, S., & Yadav, A. (2016). Computational thinking and media & information literacy: An integrated approach to teaching twenty-first century skills. *TechTrends*, *60*(5), 510-516. https://doi.org/10.1007/s11528-016-0098-4

Grover, S. (2017). Assessing algorithmic and computational thinking in K-12: Lessons from a middle school classroom. In *Emerging research, practice, and policy on computational thinking* (pp. 269-288). Springer, Cham.

Grover, S., Cooper, S., & Pea, R. (2014, June 21-25). *Assessing computational learning in K-12* [Paper presentation]. The 2014 conference on Innovation & technology in computer science education, New York, NY, United States. https://doi.org/10.1145/2591708.2591713

Grover, S., & Pea, R. (2013). Computational thinking in K–12 a review of the state of the field. *Educational Researcher, 42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Grover, S., & Pea, R. (2018). Computational Thinking: A Competency Whose Time Has Come. In S. Sentance, E. Barendsen, & C. Schulte (Eds.), *Computer Science Education: Perspectives on Teaching and Learning in School* (p. 20-38). Bloomsbury Publishing.

Güney, Z. (2019). Four-Component Instructional Design (4C/ID) Model Approach for Teaching Programming Skills. *International Journal of Progressive Education*, *15*(4), 142-156. https://doi.org/10.29329/ijpe.2019.203.11

Gursoy, A. (2006). Education programs and teaching. *Procedia - Social and Behavioral Sciences*, *152*, 137-142. https://doi.org/10.1016/j.sbspro.2014.09.170.

Guzdial, M. (2008). Education paving the way for computational thinking. *Communications of the ACM*, *51*(8), 25-27. https://doi.org/10.1145/1378704.1378713

Han, A., Kim, J., & Wohn, K. (2015, September 7-11*). Entry: visual programming to enhance children's computational thinking* [Paper presentation]. The 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, Osaka, Japan. https://doi.org/10.1145/2800835.2800871

Harangus, K., & Kátai, Z. (2020). Computational thinking in secondary and higher education. *Procedia Manufacturing*, *46*, 615-622. https://doi.org/10.1016/j.promfg.2020.03.088

Hartmann, W., Nievergelt, J., & Reichert, R. (2001, September 5-7). *Kara, finite state machines, and the case for programming as part of general education* [Paper presentation]. The 2001 IEEE Symposia on Human-Centric Computing Languages and Environments, Stresa, Italy. https://doi.org/10.1109/HCC.2001

Hazzan, O., Ragonis, N., & Lapidot, T. (2020). Computational Thinking. In *Guide to Teaching Computer Science* (pp. 57-74). Springer, Cham.

Heintz, F., Mannila, L., & Färnqvist, T. (2016, October 12-15). *A review of models for introducing computational thinking, computer science and computing in K-12 education* [Paper presentation]. The 2016 IEEE Frontiers in Education Conference (FIE), Eire, PA, United States. https://doi.org/10.1109/FIE.2016.7757410

Hesser, T. L., & Gregory, J. L. (2015). Exploring the Use of Faded Worked Examples as a Problem Solving Approach for Underprepared Students. *Higher Education Studies*, *5*(6), 36-46. https://doi.org/10.5539/hes.v5n6p36

Hiemstra, R., & Sisco, B. (1990). *Individualizing Instruction. Making Learning Personal, Empowering, and Successful*. Jossey-Bass.

Hoogerheide, V., & Roelle, J. (2020). Example-based learning: New theoretical perspectives and use-inspired advances to a contemporary instructional approach. *Applied Cognitive Psychology*, *34*(4), 787-792. https://doi.org/10.1002/acp.3706

Hopia, H., Latvala, E., & Liimatainen, L. (2016). Reviewing the methodology of an integrative review. *Scandinavian journal of caring sciences*, *30*(4), 662-669. https://doi.org/10.1111/scs.12327

Hoskinson, A. M., Caballero, M. D., & Knight, J. K. (2013). How can we improve problem solving in undergraduate biology? Applying lessons from 30 years of physics education research. *CBE—Life Sciences Education*, *12*(2), 153-161. https://doi.org/10.1187/cbe.12-09-0149

Hosseini, R., Akhuseyinoglu, K., Brusilovsky, P., Malmi, L., Pollari-Malmi, K., Schunn, C., & Sirkiä, T. (2020). Improving engagement in program construction examples for learning Python programming. *International Journal of Artificial Intelligence in Education*, *30*(2), 299-336. https://doi.org/10.1007/s40593-020-00197-0

Hoyles, C., & Noss, R. (2015). Revisiting programming to enhance mathematics learning [Paper presentation]. In *Math+ coding symposium. Western University, London*

Hsu, J. M., Chang, T. W., & Yu, P. T. (2012). Learning Effectiveness and Cognitive Loads in Instructional Materials of Programming Language on Single and Dual Screens. *Turkish Online Journal of Educational Technology-TOJET*, *11*(2), 156-166.

Hsu, T. C., Chang, S. C., & Hung, Y. T. (2018). How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education*, *126*, 296-310. https://doi.org/10.1016/j.compedu.2018.07.004

Hu, C. (2011, June 27-29). *Computational thinking: what it might mean and what we might do about it* [Paper presentation]. The 16th annual joint conference on Innovation and technology in computer science education, Darmstadt, Germany. https://doi.org/10.1145/1999747.1999811

IEA (2016). *The IEA's international computer and information literacy study (ICILS) 2018*. What's next for IEA's ICILS in 2018?. http://www.iea.nl/fileadmin/user_upload/Studies/ICILS_2018/IEA_ICILS_2018_Computational_Thinking_Leaflet.pdf.

Ismail, M. N., Ngah, N. A., & Umar, I. N. (2010). Instructional strategy in the teaching of computer programming: a need assessment analyses. *TOJET: The Turkish Online Journal of Educational Technology*, *9*(2).

Israel-Fishelson, R., & Hershkovitz, A. (2022). Studying interrelations of computational thinking and creativity: A scoping review (2011–2020). *Computers & Education*, *176*, 104353. https://doi.org/10.1016/j.compedu.2021.104353

ISTE. (2011). *Teacher Resources*. https://www.iste.org/explore/articledetail?articl eid=152

Jackson, G. B. (1980). Methods for integrative reviews. *Review of educational research*, *50*(3), 438-460. https://doi.org/10.3102/0034654305000343

Jacob, S., Nguyen, H., Garcia, L., Richardson, D., & Warschauer, M. (2020, March 10-11). *Teaching Computational Thinking to Multilingual Students through Inquiry-based Learning* [Paper presentation]. The 2020 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT), Portlans, OR, United States.

Jacobs, A. E., Dolmans, D. H., Wolfhagen, I. H., & Scherpbier, A. J. (2003). Validation of a short questionnaire to assess the degree of complexity and structuredness of PBL problems. *Medical Education*, *37*(11), 1001-1007. https://doi.org/10.1046/j.1365-2923.2003.01630.x

Jennings, J., & Muldner, K. (2021). When does scaffolding provide too much assistance? a code-tracing tutor investigation. *International Journal of Artificial Intelligence in Education*, *31*(4), 784-819. https://doi.org/10.1145/2462476.2462507

Jensen, S. H., Møller, A., & Thiemann, P. (2009, August). Type analysis for JavaScript. In *International Static Analysis Symposium* (pp. 238-255). Springer, Berlin, Heidelberg.

Jocius, R., Joshi, D., Dong, Y., Robinson, R., Cateté, V., Barnes, T., ... & Lytle, N. (2020, March 11-14). *Code, connect, create: The 3c professional development model to support computational thinking infusion* [Paper presentation]. The 51st ACM Technical Symposium on Computer Science Education, Portland, OR, United States. https://doi.org/10.1145/3328778.3366797

Jocius, R., O'Byrne, W. I., Albert, J., Joshi, D., Robinson, R., & Andrews, A. (2021). Infusing Computational Thinking into STEM Teaching. *Educational Technology & Society*, *24*(4), 166-179. https://www.jstor.org/stable/48629253

Jonassen, D. H. (1997). Instructional design models for well-structured and III-structured problem-solving learning outcomes. *Educational technology research and development*, *45*(1), 65-94. https://doi.org/10.1007/BF02299613

Jonassen, D. H. (2000). Toward a design theory of problem solving. *Educational technology research and development*, *48*(4), 63-85. https://doi.org/10.1007/BF02300500

Jonassen, D. H. (2010). *Learning to solve problems: A handbook for designing problem-solving learning environments*. Routledge.

Jones, P., & Davis, R. (2008). Instructional design methods integrating instructional technology. In *Handbook of research on instructional systems and technology* (pp. 15-27). IGI Global.

Kafai, Y. B., & Burke, Q. (2013). Computer programming goes back to school. *Phi Delta Kappan*, *95*(1), 61-65. https://doi.org/10.1177/003172171309500111

Kafura, D., & Tatar, D. (2011, March 9-12). *Initial experience with a computational thinking course for computer science students* [Paper presentation]. The 42nd ACM Technical Symposium on Computer Science Education, Dallas, TX, United States. https://doi.org/10.1145/1953163.1953242

Kale, U., Akcaoglu, M., Cullen, T., Goh, D., Devine, L., Calvert, N., & Grise, K. (2018). Computational what? Relating computational thinking to teaching. *TechTrends*, *62*(6), 574-584. https://doi.org/10.1007/s11528-018-0290-9

Kale, U., & Yuan, J. (2021). Still a new kid on the block? Computational thinking as problem solving in Code. org. *Journal of Educational Computing Research*, *59*(4), 620-644. https://doi.org/10.1177/073563312097205

Kalyuga, S., Ayres, P., Chandler, P., Sweller, J. (2003). The expertise reversal effect. *Educational Psychologist, 38*(1), 23–31. https://doi.org/10.1207/S15326985EP3801_4

Katrak, P., Bialocerkowski, A. E., Massy-Westropp, N., Kumar, V. S., & Grimmer, K. A. (2004). A systematic review of the content of critical appraisal tools. *BMC medical research methodology*, *4*(1), 1-11. https://doi.org/10.1186/1471-2288-4-22

Katz, D. L. (Ed.). (1960). Conference report on the use of computers in engineering classroom instruction. *Communications of the ACM*, *3*(10), 522-527. https://doi.org/10.1145/3098997

Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2012). Learning programming at the computational thinking level via digital game-play. *Procedia Computer Science*, *9*, 522-531. https://doi.org/10.1016/j.procs.2012.04.056

Kilpeläinen, P. (2010). Do all roads lead to Rome?(Or reductions for dummy travelers). *Computer Science Education*, *20*(3), 181-199. https://doi.org/10.1080/08993408.2010.501226

Kim, M. C., & Hannafin, M. J. (2011). Scaffolding problem solving in technology-enhanced learning environments (TELEs): Bridging research and theory with practice. *Computers & Education*, *56*(2), 403-417. https://doi.org/10.1016/j.compedu.2010.08.024

Kirschner, P.A., Sweller, J., Clark, R.E., (2006). Why minimal guidance during instruction does not work: an analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educ. Psychol. 41*(2), 75–86. https://doi.org/10.1207/s15326985ep4102_1

Kitchener, K. S. (1983). Cognition, metacognition, and epistemic cognition. *Human development*, *26*(4), 222-232. https://doi.org/10.1159/000272885

Klepsch, M., & Seufert, T. (2020). Understanding instructional design effects by differentiated measurement of intrinsic, extraneous, and germane cognitive load. *Instructional Science*, *48*(1), 45-77. https://doi.org/10.1007/s11251-020-09502-9

Knoblich, G., Ohlsson, S., Haider, H., & Rhenius, D. (1999). Constraint relaxation and chunk decomposition in insight problem solving. *Journal of Experimental Psychology: Learning, memory, and cognition*, *25*(6), 1534. https://doi.org/10.1037/0278-7393.25.6.1534

Knuth, D. E. (1974). Computer science and its relation to mathematics. *The American Mathematical Monthly*, *81*(4), 323-343. https://doi.org/10.1080/00029890.1974.11993556

Kölling, M. (1999). The problem of teaching object-oriented programming, Part 1: Languages. *Journal of Object-oriented programming*, *11*(8), 8-15.

Kong, S. C. (2019). Components and methods of evaluating computational thinking for fostering creative problem-solvers in senior primary school education. In *Computational thinking education* (pp. 119-141). Springer, Singapore.

Kong, S. C., Chiu, M. M., & Lai, M. (2018). A study of primary school students' interest, collaboration attitude, and programming empowerment in computational thinking education. *Computers & education*, *127*, 178-189. https://doi.org/10.3390/ijerph19106005

Koulouri, T., Lauria, S., & Macredie, R. D. (2014). Teaching introductory programming: A quantitative evaluation of different approaches. *ACM Transactions on Computing Education (TOCE)*, *14*(4), 1-28. https://doi.org/10.1145/2662412

Krugel, J., & Hubwieser, P. (2017, April 26-28). *Computational thinking as springboard for learning object-oriented programming in an interactive MOOC* [Paper presentation]. IEEE Global

Engineering Education Conference, Athens, Greece. https://doi.org/10.1109/EDUCON.2017.7943079.

Kumar, A. N. (2016, June). Using Cloze Procedure Questions in Worked Examples in a Programming Tutor. In *International Conference on Intelligent Tutoring Systems* (pp. 416-422). Springer, Cham.

Labusch, A., Eickelmann, B., & Vennemann, M. (2019). Computational thinking processes and their congruence with problem-solving and information processing. In *Computational thinking education* (pp. 65-78). Springer, Singapore.

Lange, C., Almusharraf, N., Koreshnikova, Y., & Costley, J. (2021). The effects of example-free instruction and worked examples on problem-solving. *Heliyon*, *7*(8), e07785. https://doi.org/10.1016/j.heliyon.2021.e07785

Laxman, K. (2010). A conceptual framework mapping the application of information search strategies to well and ill-structured problem solving. *Computers & Education*, *55*(2), 513-526. https://doi.org/10.1016/j.compedu.2010.02.014

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., ... & Werner, L. (2011). Computational thinking for youth in practice. *Acm Inroads*, *2*(1), 32-37. https://doi.org/10.1145/1929887.1929902

Lee, N., & Hong, E. (2017). Examining the Effects of Two Computer Programming Learning Strategies: Self-Explanation versus Reading Questions and Answers. *IAFOR Journal of Education*, *5*, 69-88. https://doi.org/10.22492/ije.5.si.03

Lee, T. Y., Mauriello, M. L., Ahn, J., & Bederson, B. B. (2014). CTArcade: Computational thinking with games in school age children. *International Journal of Child-Computer Interaction*, *2*(1), 26-33. http://dx.doi.org/10.1016/j.ijcci.2014.06.003

Lee, Y. (2004). *Student perceptions of problems' structuredness, complexity, situatedness, and information richness and their effects on problem-solving performance* (Publication No. 3160668) [Doctoral dissertation, The Florida State University]. ProQuest Dissertations Publishing.

Li, Y., Schoenfeld, A. H., diSessa, A. A., Graesser, A. C., Benson, L. C., English, L. D., & Duschl, R. A. (2020). On computational thinking and STEM education. *Journal for STEM Education Research*, *3*(2), 147-166. https://doi.org/10.1007/s41979-020-00044-w

Lieberman, H. (1986). An example based environment for beginning programmers. *Instructional Science*, *14*(3-4), 277-292. https://doi.org/10.1007/BF00051824

Lintern G., Moon B., Klein G., and Hoffman R., (2018). Chap. 11, Eliciting and Representing the Knowledge of Experts, in Ericsson, K.A., Hoffman, R.R., Kozbelt, A., & Williams, A.A., (Eds.). *The Cambridge handbook of expertise and expert performance, 2nd ed*. Cambridge University Press

Looi, C. K., How, M. L., Longkai, W., Seow, P., & Liu, L. (2018). Analysis of linkages between an unplugged activity and the development of computational thinking. *Computer Science Education*, *28*(3), 255-279. https://doi.org/10.1080/08993408.2018.1533297

Luszcz, M. A. (1989). Theoretical models of everyday problem solving in adulthood. In J. D. Sinnott (Ed.), *Everyday problem solving: Theory and applications* (pp. 24–39). Praeger Publishers.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12?. *Computers in Human Behavior*, *41*, 51-61. https://doi.org/10.1016/j.chb.2014.09.012

Lyon, J. A., & J. Magana, A. (2020). Computational thinking in higher education: A review of the literature. *Computer Applications in Engineering Education*, *28*(5), 1174-1189. https://doi.org/10.1002/cae.22295

Lyon, J. A., & Magana, A. J. (2021). The use of engineering model-building activities to elicit computational thinking: A design-based research study. *Journal of Engineering Education*, *110*(1), 184-206. https://doi.org/10.1002/jee.20372

Maharani, S., Kholid, M. N., Pradana, L. N., & Nusantara, T. (2019). Problem solving in the context of computational thinking. *Infinity Journal*, *8*(2), 109-116. https://doi.org/10.22460/infinity.v8i2.p109-116

Mahatanankoon, P., & Wolf, J. (2021). Cognitive Learning Strategies in an Introductory Computer Programming Course. *Information Systems Education Journal*, *19*(3), 11-20. https://doi.org/ 10.3390/ijerph19106005

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, *10*(4), 1-15. https://doi.org/10.1145/1868358.1868363

Mangaroska, K., Martinez-Maldonado, R., Vesin, B., & Gašević, D. (2021). Challenges and opportunities of multimodal data in human learning: The computer science students' perspective. *Journal of Computer Assisted Learning*, *37*(4), 1030-1047. https://doi.org/10.1111/jcal.12542

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014, June 23-25*). Computational thinking in K-9 education* [Paper presentation]. TICSE '14: Innovation and Technology in Computer Science Education Conference 2014, Uppsala, Sweden. https://doi.org/10.1145/2713609.2713610

Margulieux, L. E., & Catrambone, R. (2016). Improving problem solving with subgoal labels in expository text and worked examples. *Learning and Instruction*, *42*, 58-71. http://dx.doi.org/10.1016/j.learninstruc.2015.12.002

Margulieux, L. E., & Catrambone, R. (2021). Scaffolding problem solving with learners' own self explanations of subgoals. *Journal of Computing in Higher Education*, *33*(2), 499-523. https://doi.org/10.1007/s12528-021-09275-1

Margulieux, L. E., Catrambone, R., & Guzdial, M. (2016). Employing subgoals in computer programming education. *Computer Science Education*, *26*(1), 44-67. https://scholarworks.gsu.edu/ltd_facpub/3

Margulieux, L. E., Catrambone, R., & Schaeffer, L. M. (2018). Varying effects of subgoal labeled expository text in programming, chemistry, and statistics. *Instructional Science*, *46*(5), 707-722. https://scholarworks.gsu.edu/ltd_facpub/12

Margulieux, L. E., Guzdial, M., & Catrambone, R. (2012, September 9-11). *Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications* [Paper presentation]. International Computing Education Research Conference, Auckland, New Zealand. https://doi.org/10.1145/2361276.2361291

Margulieux, L. E., Morrison, B. B., & Decker, A. (2020). Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples. *International Journal of STEM Education*, *7*(1), 1-16. https://doi.org/10.1186/s40594-020-00222-7

Martin, F. (2011). Instructional design and the importance of instructional alignment. *Community College Journal of Research and Practice*, *35*(12), 955-972. https://doi.org/10.1080/10668920802466483

Mason, S. (2019). *Examining the Nature of Collaborative Learning in the Context of Problem Solving in First-Year and Second-Year Computing Education, a Naturalistic Exploratory Case Study* (Publication No. 13808099) [Doctoral dissertation, State University of New York at Buffalo]. ProQuest Dissertations Publishing.

Mayer, R. (2001). Temporal Contiguity Principle. In *Multimedia Learning* (pp. 96-112). Cambridge: Cambridge University Press. doi:10.1017/CBO9781139164603.007.

Mayer, R. E. (2006). Ten research-based principles of multimedia learning. In H. F. O'Neil Jr. & R. S. Perez (Eds.), *Web-based learning: Theory, research, and practice* (pp. 371–390). Mahwah, NJ: Lawrence Erlbaum. Associates.

Mayer R. E. (2011). Multimedia learning and games. In Tobias S., Fletcher J. D. (Eds.), *Computer games and instruction* (pp. 281–305). Charlotte, NC: Information Age.

Mayer, R. E. (2020). Advances in designing instruction based on examples. *Applied Cognitive Psychology*, *34*(4), 912-915. https://doi.org/10.1002/acp.3701

Mayer, R. E., & Wittrock, M. C. (2006). Problem solving. In P. A. Alexander & P. H. Winne (Eds.), *Handbook of educational psychology* (pp. 287–303). Lawrence Erlbaum Associates Publishers.

McLaren, B. M., & Isotani, S. (2011, June). When is it best to learn with all worked examples?. In *International conference on artificial intelligence in education* (pp. 222-229). Springer, Berlin, Heidelberg.

McVeigh-Murphy, A. (2019, August 19). *Computational Thinking and Computer Science*. equip. https://equip.learning.com/computer-science-computational-thinking-coding

Meacham, J. A., & Emont, N. C. (1989). The interpersonal basis of everyday problem solving. In J. D. Sinnott (Ed.), *Everyday problem solving: Theory and applications* (pp. 7–23). Praeger Publishers.

Melton, J., & Simon, A. R. (1993). *Understanding the new SQL: a complete guide*. Morgan Kaufmann.

Mentz, E. (Ed.). (2013). *Empowering IT & CAT Teachers* (1st ed.). African Sun Media.

Michaelson, G. (2018). Microworlds, objects first, computational thinking and programming. In *Computational thinking in the STEM disciplines* (pp. 31-48). Springer, Cham.

Miles, M. B., & Huberman, A. M. (1994). *Qualitative data analysis. 2*. Aufl., Thousand Oaks et al.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, *63*(2), 81. https://doi.org/10.1037/h0043158

Minsky, M. (1961). Steps toward artificial intelligence. *Proceedings of the IRE*, *49*(1), 8-30. https://doi.org/10.1109/JRPROC.1961.287775

Moon, J., Do, J., Lee, D., & Choi, G. W. (2020). A conceptual framework for teaching computational thinking in personalized OERs. *Smart Learning Environments*, *7*(1), 1-19. https://doi.org/10.1186/s40561-019-0108-z

Moore, T. J., Brophy, S. P., Tank, K. M., Lopez, R. D., Johnston, A. C., Hynes, M. M., & Gajdzik, E. (2020). Multiple representations in computational thinking tasks: a clinical study of second-grade students. *Journal of Science Education and Technology*, *29*(1), 19-34. https://doi.org/10.1007/s10956-020-09812-0

Moreno-León, J., Robles, G., & Román-González, M. (2015). Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, (46), 1-23

Moreno, R. (2006). When worked examples don't work: Is cognitive load theory at an impasse?. *Learning and Instruction*, *16*(2), 170-181. https://doi.org/10.1016/j.learninstruc.2006.02.006

Moreno, R., Reisslein, M., & Ozogul, G. (2009). Optimizing worked-example instruction in electrical engineering: The role of fading and feedback during problem-solving practice. *Journal of Engineering Education*, *98*(1), 83-92. https://doi.org/10.1002/j.2168-9830.2009.tb01007.x

Morrison, B. B., Decker, A., & Margulieux, L. E. (2016, September 8-12*). Learning loops: A replication study illuminates impact of HS courses* [Paper presentation]. The 2016 ACM conference on international computing education research, Melbourne, VIC, Australia. http://dx.doi.org/10.1145/2960310.2960330 221

Morrison, B. B., Margulieux, L. E., & Decker, A. (2020). The curious case of loops. *Computer Science Education*, *30*(2), 127-154. https://doi.org/10.1080/08993408.2019.1707544

Morrison, B. B., Margulieux, L. E., & Guzdial, M. (2015, July 9-13). *Subgoals, context, and worked examples in learning computing problem solving* [Paper presentation International Computing Education Research Conference, Omaha, Nebraska, United States. https://doi.org/10.1145/2787622.2787733

Nainan, M., & Balakrishnan, B. (2019). Design and Evaluation of Worked Examples for Teaching and Learning Introductory Programming at Tertiary Level. *Malaysian Online Journal of Educational Technology*, *7*(4), 30-44. https://doi.org/10.17220/mojet.2019.04.003

Najar, A. S., & Mitrovic, A. (2013, July). Examples and tutored problems: How can self-explanation make a difference to learning?. In *International Conference on Artificial Intelligence in Education* (pp. 339-348). Springer, Berlin, Heidelberg.

National Research Council. (2011). *Successful K-12 STEM education: Identifying effective approaches in science, technology, engineering, and mathematics*. National Academies Press.

National Research Council (NRC). (2012a). 5 Problem Solving, Spatial Thinking, and the Use of Representations in Science and Engineering. *Discipline-Based Education Research: Understanding and Improving Learning in Undergraduate Science and Engineering*, pp. 75–118. https://doi.org/10.17226/13362

National Research Council. (2012). *Education for life and work: Developing transferable knowledge and skills in the 21st century*. The National Academies Press. https://doi.org/10.17226/13398

Neelakandan, N. (2021, May 12). *5 main reasons why instructional design matters in Elearning*. eLearning Industry. https://elearningindustry.com/reasons-instructional-design-matters-elearning.

Neelen, M., & Kirschner, P. A. (2021, July 4). *Designing winning worked examples 2 – Inter example features*. 3-STAR LEARNING EXPERIENCES. https://3starlearningexperiences.wordpress.com/2021/07/06/designing-winning-worked-examples-2-inter-example-features/.

Newell, A., & Simon, H. A. (1972). *Human problem solving* (Vol. 104, No. 9). Englewood Cliffs, NJ: Prentice-hall.

Nouri, J., Zhang, L., Mannila, L., & Norén, E. (2020). Development of computational thinking, digital competence and 21st century skills when learning programming in K-9. *Education Inquiry*, *11*(1), 1-17. https://doi.org/10.1080/20004508.2019.1627844

Oermann, M. H., & Hays, J. C. (2019). *Writing for Publication in Nursing (4th ed.)*. New York, NY: Springer.

Oermann, M. H., & Knafl, K. A. (2021). Strategies for completing a successful integrative review. *Nurse Author & Editor*, *31*(3-4), 65-68. https://doi.org/10.1111/nae2.30

Oermann, M. H., Nicoll, L. H., Chinn, P. L., Conklin, J. L., McCarty, M., & Amarasekara, S. (2018). Quality of author guidelines in nursing journals. *Journal of Nursing Scholarship*, *50*(3), 333-340. https://doi.org/10.1111/jnu.12383

Oliveira, E. C., Bittencourt, R. A., & Trindade, R. P. (2019, October 16-19). *Introduction to computational thinking for k-12 educators through distance learning* [Conference Paper]. 2019 IEEE Frontiers in Education Conference (FIE), Covington, KY, United States. https://doi.org/10.1109/FIE43999.2019.9028492

Ouyang, Y., Hayden, K. L., & Remold, J. (2018, February 21-24). *Introducing computational thinking through non-programming science activities* [Paper presentation]. The 49th ACM Technical Symposium on Computer Science Education, Baltimore, Maryland, United States. https://doi.org/10.1145/3159450.3159520

Paas, F. G. (1992). Training strategies for attaining transfer of problem-solving skill in statistics: a cognitive-load approach. *Journal of educational psychology*, *84*(4), 429. https://doi.org/10.1037/0022-0663.84.4.429

Paas, F. G., & Van Merriënboer, J. J. (1994). Variability of worked examples and transfer of geometrical problem-solving skills: A cognitive-load approach. *Journal of educational psychology*, *86*(1), 122. https://doi.org/10.1037/0022-0663.86.1.122

Paas, F., Renkl, A., & Sweller, J. (2003). Cognitive load theory and instructional design: Recent developments. *Educational psychologist*, *38*(1), 1-4. https://doi.org/10.1207/S15326985EP3801_1

Paas, F., Van Gerven, P. W., & Wouters, P. (2007). Instructional efficiency of animation: Effects of interactivity through mental reconstruction of static key frames. *Applied Cognitive Psychology, 21*(6), 783-793. https://doi.org/10.1002/acp.1349

Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E., Chou, R., Glanville, J., Grimshaw, J. M., Hróbjartsson, A., Lalu, M. M., Li, T., Loder, E. W., Mayo-Wilson, E., McDonald, S., … Moher, D. (2021). The prisma 2020 statement: an updated guideline for reporting systematic reviews. *International Journal of Surgery (London, England)*, *88.* https://doi.org/10.1136/bmj.n71

Palts, T., & Pedaste, M. (2020). A model for developing computational thinking skills. *Informatics in Education*, *19*(1),113-128.

Papadakis, S., Kalogiannakis, M., Orfanakis, V., & Zaranis, N. (2014, June 9). *Novice programming environments. Scratch & app inventor: a first comparison* [Paper presentation]. The 2014 workshop on interaction design in educational environments, Albacete, Spain. https://doi.org/10.1145/2643604.2643613

Papert, S. (1980). *Mindstorms: children, computers, and powerful ideas Basic Books*. Basic Books.

Papert, S. (1993). *The children's machine: Rethinking school in the age of the computer*. Basic Books.

Park, S. Y., Song, K. S., & Kim, S. H. (2015, April 23-25). *EEG analysis for computational thinking based education effect on the learners' cognitive load* [Paper presentation]. The Applied Computer and Applied Computational Science (ACACOS'15), Kuala Lumpur, Malaysia.

Park, Y., & Shin, Y. (2019). Comparing the effectiveness of scratch and app inventor with regard to learning computational thinking concepts. *Electronics*, *8*(11), 1269. https://doi.org/10.3390/electronics8111269

Pirolli, P. L., & Anderson, J. R. (1985). The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology/Revue canadienne de psychologie*, *39*(2), 240. https://doi.org/10.1037/h0080061

Potvin, G. (2019, March 20-22). *Problem-Based Lab Education: Redesign of a Senior Year Chemical Engineering Lab Course to Promote Autonomy, Critical Thinking, and Problem-Solving Skills* [Paper presentation]. 2019 ASEE PNW Section Conference, Corvallis, Oregon, United States. https://peer.asee.org/31887

Probes. Center for computational Thinking, Carnegie Mellon. (n.d.). https://www.cs.cmu.edu/~CompThink/probes.html.

Pulgar, J. A. (2019). *Performance on Well and Ill-Structured Problems in University Physics: The role of Instruction in Cooperative Learning*. University of California, Santa Barbara.

Qualls, J. A., & Sherrell, L. B. (2010). Why computational thinking should be integrated into the curriculum. *Journal of Computing Sciences in Colleges*, *25*(5), 66-71.

Quilici, J.L. & Mayer, R.E. (2002). Teaching students to recognize structural similarities between statistics word problems. Applied Cognitive Psychology, 16, 325-342. https://doi.org/10.1002/acp.796

Rahman, M. M. (2019). 21st Century Skill "Problem Solving": Defining the Concept. *Asian Journal of Interdisciplinary Research*, *2*(1), 64-74. https://doi.org/10.34256/ajir191

Raj, A. G. S., Patel, J., & Halverson, R. (2018, April 19-22). *Is More Active Always Better for Teaching Introductory Programming?* [Paper presentation]. The International Conference on Learning and Teaching in Computing and Engineering (LaTICE), Auckland, New Zealand. https://doi.org/10.1109/LaTICE.2018.00006

Rajaravivarma, R. (2005). A games-based approach for teaching the introductory programming course. *ACM SIGCSE Bulletin*, *37*(4), 98-102. https://doi.org/10.1145/1113847.1113886

Reed, S. K. (2016). The structure of ill-structured (and well-structured) problems revisited. *Educational Psychology Review*, *28*(4), 691-716. https://doi.org/10.1007/s10648-015-9343-1

Reed, S. K., Willis, D., & Guarino, J. (1994). Selecting examples for solving word problems. *Journal of Educational Psychology*, *86*(3), 380. https://doi.org/10.1037/0022-0663.86.3.380

Renkl, A. (2011). Instruction based on examples. In *Handbook of research on learning and instruction* (pp. 286-309). Routledge.

Renkl A. (2012) Example-Based Learning. In: Seel N.M. (eds) *Encyclopedia of the Sciences of Learning*. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-1428-6_1370

Renkl, A. (2012). Example-Based Learning. In: Seel, N.M. (eds) *Encyclopedia of the Sciences of Learning.* Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-1428-6_1370.

Renkl, A. (2014a). The Worked Examples Principle in Multimedia Learning. In R. Mayer (Ed.), *The Cambridge Handbook of Multimedia Learning* (Cambridge Handbooks in Psychology, pp. 391-412). Cambridge: Cambridge University Press. doi:10.1017/CBO9781139547369.020

Renkl, A. (2014b). Toward an instructionally oriented theory of example-based learning. *Cognitive science*, *38*(1), 1-37.

Renkl, A., Atkinson, R. K., & Große, C. S. (2004). How fading worked solution steps works–a cognitive load perspective. *Instructional science*, *32*(1), 59-82. https://doi.org/10.1023/B:TRUC.0000021815.74806.f6

Renkl, A., Hilbert, T., & Schworm, S. (2009). Example-based learning in heuristic domains: A cognitive load theory account. *Educational Psychology Review*, *21*(1), 67-78. https://doi.org/10.1007/s10648-008-9093-4

Resnick, M. (2014, August 19-23). *Give P'sa chance: Projects, peers, passion, play* [Opening keynote]. *Constructionism 2014: Constructionism and Creativity. Theory and practice of constructionist learning with due respect to creativity, Vienna, Austria. http://constructionism2014.ifs.tuwien.ac.at/home*

Resnick, M. (2017). The seeds that Seymour sowed. *International Journal of Child-Computer*. https://web.media.mit.edu/~mres/papers/IJCCI-seeds-seymour-sowed.pdf

Retnowati, E. (2017). Faded-example as a Tool to Acquire and Automate Mathematics Knowledge. *Journal of Physics: Conference Series, 824*(1). https://doi.org/10.1088/1742-6596/824/1/012054

Retnowati, E., Ayres, P., & Sweller, J. (2017). Can collaborative learning improve the effectiveness of worked examples in learning mathematics?. *Journal of educational psychology*, *109*(5), 666–679. https://doi.org/10.1037/edu0000167

Richey, R. C., & Klein, J. D. (2005). Developmental research methods: Creating knowledge from instructional design and development practice. *Journal of Computing in higher Education*, *16*(2), 23-38. https://doi.org/10.1007/BF02961473

Richey, R., & Klein, J. (2007). *Design and development research: methods, strategies, and issues.* Lawrence Erlbaum Associates.

Richey, R. C., & Klein, J. D. (2014). Design and development research. In *Handbook of research on educational communications and technology* (pp. 141-150). Springer.

Richey, R. C., Klein, J. D., & Tracey, M. W. (2011). The instructional design knowledge base. *Theory, research, and practice*. Routledge.

Rohlfing, K. J., Rehm, M., & Goecke, K. U. (2003). Situatedness: The interplay between context (s) and situation. *Journal of Cognition and Culture*, *3*(2), 132-156. https://doi.org/10.1163/156853703322148516

Romero, M., Lepage, A., & Lille, B. (2017). Computational thinking development through creative programming in higher education. *International Journal of Educational Technology in Higher Education*, *14*(1), 1-15. https://doi.org/10.1186/s41239-017-0080-z

Royal Society (Great Britain). (2012). *Shut down or restart?: The way forward for computing in UK schools*. Royal Society.

Rumelhart, D. E. (2017). *Schemata: The building blocks of cognition* (pp. 33-58). Routledge.

Rummel, N., Spada, H., & Hauser, S. (2009). Learning to collaborate while being scripted or by observing a model. *International Journal of Computer-Supported Collaborative Learning*, *4*(1), 69-92. https://doi.org/10.1007/s11412-008-9054-4

Russell, C. L. (2005). An overview of the integrative research review. *Progress in transplantation*, *15*(1), 8-13. https://doi.org/10.1177/152692480501500102

Sanabria, M. L. B., & Pulido, L. H. O. (2009). Critical review of problem solving processes traditional theoretical models. *International Journal of Psychological Research*, *2*(1), 67-72. https://doi.org/10.21500/20112084.879

Sandelowski, M. (1995). Qualitative analysis: What it is and how to begin. *Research in nursing & health*, *18*(4), 371-375. https://doi.org/10.1002/nur.4770180411

Sanderson, S., Tatt, I. D., & Higgins, J. (2007). Tools for assessing quality and susceptibility to bias in observational studies in epidemiology: a systematic review and annotated bibliography. *International journal of epidemiology*, *36*(3), 666-676. https://doi.org/10.1093/ije/dym018

Sanford, J. F., & Naidu, J. T. (2016). Computational thinking concepts for grade school. *Contemporary Issues in Education Research, 9*(1), 23–32

Santos, S. C., Tedesco, P. A., Borba, M., & Brito, M. (2020, May 2-4). *Innovative Approaches in Teaching Programming: A Systematic Literature Review* [Paper presentation]. The 12th International Conference on Computer Supported Education, Heraklion, Crete, Greece. https://doi.org/10.5220/0009190502050214

Sanz, A. (2015, April 26). *Why Teaching and Learning How to Code in Schools*. EdTech Review. http://edtechreview.in/trends-insights/insights/1934-why-teaching-and-learning-how-to-code-in-schools

Savery, J. (2006). Overview of problem-based learning: definitions and distinctions. *The Interdisciplinary Journal of Problem-based Learning, 1*(1). https://doi.org/10.7771/1541-5015.1002

Saw, K. G. (2017). Cognitive load theory and the use of worked examples as an instructional strategy in physics for distance learners: A preliminary study. *Turkish Online Journal of Distance Education*, *18*(4), 142-159. https://doi.org/10.17718/tojde.340405

Schraw, G., Dunkle, M. E., & Bendixen, L. D. (1995). Cognitive processes in well-defined and ill-defined problem solving. *Applied Cognitive Psychology*, *9*(6), 523-538. https://doi.org/10.1002/acp.2350090605

Schworm, S., & Renkl, A. (2007). Learning argumentation skills through the use of prompts for self-explaining examples. *Journal of Educational Psychology*, *99*(2), 285-296. https://doi.org/10.1037/0022-0663.99.2.285

Seehorn, D., Carey, S., Fuschetto, B., Lee, I., Moix, D., O'Grady-Cunniff, D., ... & Verno, A. (2011). *CSTA K--12 Computer Science Standards: Revised 2011*. ACM.

Seels, B. B., & Richey, R. C. (2012). *Instructional technology: The definition and domains of the field*. IAP.

Selby, C., & Woollard, J. (2013). Computational thinking: the developing definition. University of Southampton. Http://eprints.soton.ac.uk/id/eprint/356481

Seneviratne, O. (2017). Making computer science attractive to high school girls with computational thinking approaches: A case study. In *Emerging research, practice, and policy on computational thinking* (pp. 21-32). Springer, Cham.

Sethares, K. A. (2020). Dissemination of the Integrative Review. In *A Step-by-Step Guide to Conducting an Integrative Review*,(pp. 85-106). Springer.

Shareghi Najar, A., Mitrovic, A., & Neshatian, K. (2015). Eye tracking and studying examples: how novices and advanced learners study SQL examples. *Journal of computing and information technology*, *23*(2), 171-190. https://doi.org/10.2498 /cit.1002627

Shen, C. Y., & Tsai, H. C. (2009). Design Principles of Worked Examples: A Review of the Empirical Studies. *Journal of Instructional Psychology*, *36*(3).

Shin, N., & McGee, S. (2018). Identifying questions to investigate: designers should enhance students' ill-structured problem solving skills. *Center for Educational Technologies and the Classroom of the Future. http://www. cotf. edu/vdc/entries/illps. html. Accessed*, *13*, 181-201.

Si, J., Kim, D., & Na, C. (2014). Adaptive instruction to learner expertise with bimodal process-oriented worked-out examples. *Journal of Educational Technology & Society*, *17*(1), 259-271.

Simon, H. A., & Newell, A. (1971). Human problem solving: The state of the theory in 1970. *American Psychologist*, *26*(2), 145. https://doi.org/10.1037/h0030806

Sinnott, J. D. (1989). A model for solution of ill-structured problems: Implications for everyday and abstract problem solving. In J. D. Sinnott (Ed.), *Everyday problem solving: Theory and applications* (pp. 72–99). Praeger Publishers.

Smith, M. U. (1991). A view from Biology. Toward a Unified Theory of Problem Solving, edited by Mike U. Smith. Lawrence Erlbaum Associates.

Smith, P. L., & Ragan, T. J. (2004). *Instructional design*. John Wiley & Sons.

Snow, R. E. (1994). Abilities in academic tasks. In R. J. Sternberg & R. K. Wagner (Eds.), *Mind in context: Interactionist perspectives on human intelligence* (pp. 3–37). Cambridge University Press.

Soares, C. B., Hoga, L. A. K., Peduzzi, M., Sangaleti, C., Yonekura, T., & Silva, D. R. A. D. (2014). Integrative review: concepts and methods used in nursing. *Revista da Escola de Enfermagem da USP*, *48*(2), 335-345. https://doi.org/10.1590/s0080-62342014000002000020

Soleimani, A., Green, K. E., Herro, D., & Walker, I. D. (2016, June 21-24). *A tangible, story-construction process employing spatial, computational-thinking* [Paper presentation]. The 15th International Conference on Interaction Design and Children, Manchester, United Kingdom. http://dx.doi.org/10.1145/2930674.293070

Song, D., Hong, H., & Oh, E. Y. (2021). Applying computational analysis of novice learners' computer programming patterns to reveal self-regulated learning, computational thinking, and learning performance. *Computers in Human Behavior*, *120*, 106746. https://doi.org/10.1016/j.chb.2021.106746

Sorva, J., Karavirta, V., & Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)*, *13*(4), 1-64. https://doi.org/10.1145/2490822

Souza, M. T. D., Silva, M. D. D., & Carvalho, R. D. (2010). Integrative review: what is it? How to do it?. *Einstein (São Paulo)*, *8*, 102-106. https://doi.org/10.1590/S1679-45082010RW1134

Spanjers, I. A. E., Gog, T., & Merriënboer Jeroen J. G. (2012). Segmentation of worked examples: effects on cognitive load and learning. *Applied Cognitive Psychology*, *26*(3), 352–358. https://doi.org/10.1002/acp.1832.

Spiro, R.J., Coulson, R.L., Feltovich, P.J., & Anderson, D.K. (1988). *Cognitive flexibility theory: Advanced knowledge acquisition in ill-structured domains* (Technical Report No. 441). Urbana-Champaign, IL: University of Illinois, Center for the Study of Reading.

Stachel, J., Marghitu, D., Brahim, T. B., Sims, R., Reynolds, L., & Czelusniak, V. (2013). Managing cognitive load in introductory programming courses: A cognitive aware scaffolding tool. *Journal of Integrated Design and Process Science*, *17*(1), 37-54. https://doi.org/10.3233/jid-2013-0004

Stager, G. S. (2016). Seymour Papert (1928–2016). *Nature*, *537*(7620), 308-308.

Sternberg, R. J. (Ed.). (1994). *Thinking and problem solving.* Academic Press.

Sutton, A., Clowes, M., Preston, L., & Booth, A. (2019). Meeting the review family: exploring review types and associated information retrieval requirements. *Health Information & Libraries Journal*, *36*(3), 202-222. https://doi.org/10.1111/hir.12276

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive science*, *12*(2), 257-285. https://doi.org/10.1207/s15516709cog1202_4

Sweller, J. (2002, July 18-19). *Visualisation and instructional design* [Paper presentation]. The International Workshop on Dynamic Visualizations and Learning ,Tübingen, Germany.

Sweller, J. (2004). Instructional design consequences of an analogy between evolution by natural selection and human cognitive architecture. *Instructional science*, *32*(1), 9-31. https://doi.org/10.1023/B:TRUC.0000021808.72598.4d

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

Sweller, J. P., Ayres, P., & Kalyuga, S. (2011). *Explorations in the learning sciences, instructional systems and performance technologies, cognitive load theory*. Springer.

Sweller, J., & Cooper, G. A. (1985). The use of worked examples as a substitute for problem solving in learning algebra.*Cognition and instruction*, *2*(1), 59-89. https://doi.org/10.1207/s1532690xci0201_3

Sweller, J., & Sweller, S. (2006). Natural information processing systems. *Evolutionary Psychology*, *4*(1). https://doi.org/10.1177/147470490600400135

Sweller, J., Ayres, P., & Kalyuga, S. (2011). Measuring cognitive load. In *Cognitive load theory* (pp. 71-85). Springer.

Sweller, J., Ayres, P. L., Kalyuga, S., & Chandler, P. (2003). The expertise reversal effect. *Educational Phycologist, 38*(1), 23-31. https://doi.org/10.1207/S15326985EP3801_4

Sweller, J., Van Merrienboer, J. J., & Paas, F. G. (1998). Cognitive architecture and instructional design. *Educational psychology review*, *10*(3), 251-296. https://doi.org/10.1023/A:1022193728205

Sysło, M. M., & Kwiatkowska, A. B. (2013, February). Informatics for all high school students. In *International conference on informatics in schools: Situation, evolution, and perspectives* (pp. 43-56). Springer.

Taplin, M. (2006). *Mathematics through problem solving*. Institute of Sathya Sai Education.

Tedre, M. (2014). *The science of computing: shaping a discipline*. CRC Press.

Tedre, M., & Denning, P. J. (2016, November 24-27). *The long quest for computational thinking* [Paper presentation]. The 16th Koli Calling international conference on computing education research,

Tessmer, M., & Richey, R. C. (1997). The role of context in learning and instructional design. *Educational technology research and development*, *45*(2), 85-115. Koli, Finland. https://doi.org/10.1145/2999541.2999542

Tomasic, D. (2011). *Health sciences literature review made easy: The matrix method.* Jones & Bartlett Learning.

Toronto, C. E. (2020). Overview of the integrative review. In *A step-by-step guide to conducting an integrative review*, (pp 1-9). Springer

Toronto, C. E., & Remington, R. (Eds.). (2020). *A step-by-step guide to conducting an integrative review*. Springer.

Torraco, R. J. (2005). Writing integrative literature reviews: Guidelines and examples. *Human resource development review*, *4*(3), 356-367. https://doi.org/10.1177/1534484305278283

Torraco, R. J. (2016). Writing integrative reviews of the literature: Methods and purposes. *International Journal of Adult Vocational Education and Technology (IJAVET)*, *7*(3), 62-70. https://doi.org/10.4018/IJAVET.2016070106

Toukiloglou, P., & Xinogalos, S. (2022). Ingame worked examples support as an alternative to textual instructions in serious games about programming. *Journal of Educational Computing Research*, *60*(7). https://doi.org/10.1177/07356331211073655

Toy, S. (2007). *Online ill-structured problem-solving strategies and their influence on problem-solving performance*. Iowa State University.

Tu, J. J., & Johnson, J. R. (1990). Can computer programming improve problem-solving ability?. *ACM SIGCSE Bulletin*, *22*(2), 30-33. https://doi.org/10.1145/126445.126451

Uysal, M. P. (2014). Improving first computer programming experiences: The case of adapting a web-supported and well-structured problem-solving method to a traditional course. *Contemporary Educational Technology*, *5*(3), 198-217. https://doi.org/10.30935/cedtech/6125

Van Gog, T., Paas, F. (2012). Cognitive Load Measurement. In: Seel, N.M. (eds) *Encyclopedia of the Sciences of Learning*. Springe. https://doi.org/10.1007/978-1-4419-1428-6_412

Van Gog, T., & Rummel, N. (2010). Example-based learning: Integrating cognitive and social-cognitive research perspectives. *Educational psychology review*, *22*(2), 155-174. https://doi.org/10.1007/s10648-010-9134-7

Van Gog, T., Paas, F., & Van Merriënboer, J. J. (2004). Process-oriented worked examples: Improving transfer performance through enhanced understanding. *Instructional science*, *32*(1), 83-98. https://doi.org/10.1023/B:TRUC.0000021810.70784.b0

Van Gog, T., Paas, F., & Van Merriënboer, J. J. (2006). Effects of process-oriented worked examples on troubleshooting transfer performance. *Learning and Instruction*, *16*(2), 154-164. https://doi.org/10.1016/j.learninstruc.2006.02.003

Van Merriënboer, J. J. (1990). Strategies for programming instruction in high school: Program completion vs. program generation. *Journal of educational computing research*, *6*(3), 265-285. https://doi.org/10.2190/4NK5-17L7-TWQV-1EHL

Van Merriënboer, J. J. (1997). *Training complex cognitive skills: A four-component instructional design model for technical training*. Educational Technology.

Van Merrienboer, J. J., & Sweller, J. (2005). Cognitive load theory and complex learning: Recent developments and future directions. *Educational psychology review*, *17*(2), 147-177. https://doi.org/10.1007/s10648-005-3951-0

Vieira, C., Magana, A. J., Falk, M. L., & Garcia, R. E. (2017). Writing in-code comments to self-explain in computational science and engineering education. *ACM Transactions on Computing Education (TOCE)*, *17*(4), 1-21. https://doi.org/10.1145/3058751

Vieira, C., Magana, A. J., García, R. E., Jana, A., & Krafcik, M. (2018). Integrating computational science tools into a thermodynamics course. *Journal of Science Education and Technology*, *27*(4), 322-333. https://doi.org/10.1007/s10956-017-9726-9

Vieira, C., Magana, A. J., Roy, A., & Falk, M. (2021). Providing students with agency to self-scaffold in a computational science and engineering course. *Journal of Computing in Higher Education*, *33*(2), 328-366. https://doi.org/10.1007/s12528-020-09267-7

Vieira, C., Magana, A. J., Roy, A., & Falk, M. L. (2019). Student explanations in the context of computational science and engineering education. *Cognition and Instruction*, *37*(2), 201-231. https://doi.org/10.1080/07370008.2018.1539738

Vieira, C., Yan, J., & Magana, A. J. (2015). Exploring design characteristics of worked examples to support programming and algorithm design. *Journal of Computational Science Education*, *6*(1), 2-15. https://doi.org/10.22369/issn.2153-4136/6/1/1

Visser, W. (1996). Two functions of analogical reasoning in design: a cognitive-psychology approach. *Design studies*, *17*(4), 417-434. https://doi.org/10.1016/S0142-694X(96)00020-8

Voskoglou, M. G., & Buckley, S. (2012). Problem solving and computers in a learning environment. *Egyptian Computer Science Journal, 36*(4), 28–46.

Voss, J. F., & Post, T. A. (1988). On the solving of ill-structured problems. *The nature of expertise* (pp. 261–285). Lawrence Erlbaum Associates Inc.

Walker, J. C. (2015). *Adaptability in the workplace: An exploratory study on adaptive performance in the workplace using a scenario-based tool* (Publication No. 13808099) [Doctoral dissertation, University of Pennsylvania]. ProQuest Dissertations Publishing.

Walker, L. O., & Avant, K. C. (2005). *Strategies for theory construction in nursing* (Vol. 4). Upper Saddle River, NJ: Pearson/Prentice Hall.

Wang, X. M., & Hwang, G. J. (2017). A problem posing-based practicing strategy for facilitating students' computer programming skills in the team-based learning mode. *Educational Technology Research and Development*, *65*(6), 1655-1671. https://doi.org/10.1007/s11423-017-9551-0

Wang, Y., & Chiew, V. (2010). On the cognitive process of human problem solving. *Cognitive systems research*, *11*(1), 81-92. https://doi.org/10.1016/j.cogsys.2008.08.003

Ward, M., & Sweller, J. (1990). Structuring effective worked examples. *Cognition and instruction*, *7*(1), 1-39. https://doi.org/10.1207/s1532690xci0701_1

Watson, R. (2018). Starting the "Discussion" Section of a Manuscript. *Nurse Author & Editor*, *28*(2), 1-5. https://doi.org/10.1111/j.1750-4910.2018.tb00015.x

Webb, H., & Rosson, M. B. (2013, March 6-9). *Using scaffolded examples to teach computational thinking concepts* [Paper presentation]. The 44th ACM technical symposium on Computer science education, Denver, Colorado, United States. https://doi.org/10.1145/2445196.2445227

Weber, G., & Brusilovsky, P. (2001). ELM-ART: An adaptive versatile system for Web-based instruction. *International Journal of Artificial Intelligence in Education (IJAIED)*, *12*, 351-384.

Weese, J. L. (2017). *Bringing computational thinking to K-12 and higher education*. Kansas State University.

Weese, J. L., Feldhausen, R., & Bean, N. H. (2016, June 26). *The impact of STEM experiences on student self-efficacy in computational thinking* [Paper presntation]. 2016 ASEE Annual Conference & Exposition, New Orleans, Louisiana. https://doi.org/10.18260/p.26179

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, *25*(1), 127-147. https://doi.org/10.1007/s10956-015-9581-5

Weintrop, D., Wise Rutstein, D., Bienkowski, M., & McGee, S. (2021). Assessing computational thinking: an overview of the field. *Computer Science Education, 31*(2), 113-116. https://doi.org/10.1080/08993408.2021.1918380

Weir, S. (1987). *Cultivating minds: A Logo casebook*. Harper & Row.

Weiss, R. E., Knowlton, D. S., & Morrison, G. R. (2002). Principles for using animation in computer-based instruction: Theoretical heuristics for effective design. *Computers in Human Behavior*, *18*(4), 465-477. https://doi.org/10.1016/S0747-5632(01)00049-8

Werner, L., Campe, S., & Denner, J. (2012, February 29- March 3). *Children learning computer science concepts via Alice game-programming* [Paper presentation]. The 43rd ACM technical symposium on Computer Science Education, Raleigh, North Carolina, United States. https://doi.org/10.1145/2157136.2157263

Whittemore, R. (2007). Rigour in integrative reviews. *Reviewing research evidence for nursing practice: Systematic reviews*, 149-156. https://doi.org/10.1002/9780470692127.ch11

Whittemore, R., & Knafl, K. (2005). The integrative review: updated methodology. *Journal of advanced nursing*, *52*(5), 546-553. https://doi.org/10.1111/j.1365-2648.2005.03621.x

Whittemore, R., Chao, A., Jang, M., Minges, K. E., & Park, C. (2014). Methods for knowledge synthesis: an overview. *Heart & Lung*, *43*(5), 453-461. https://doi.org/10.1016/j.hrtlng.2014.05.014

Wilkerson, M. H., D'Angelo, C. M., & Litts, B. K. (2020*).* Stories from the field: locating and cultivating computational thinking in spaces of learning*. Interactive Learning Environments, 28(3), 264-271.* https://doi.org/10.1080/10494820.2020.1711326

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717-3725. https://doi.org/10.1098/rsta.2008.0118

Wittwer, J., & Renkl, A. (2010). How effective are instructional explanations in example-based learning? A meta-analytic review. *Educational Psychology Review*, *22*(4), 393-409**.** https://doi.org/10.1007/s10648-010-9136-5

Wolz, U., Stone, M., Pearson, K., Pulimood, S. M., & Switzer, M. (2011). Computational thinking and expository writing in the middle school. *ACM Transactions on Computing Education (TOCE)*, *11*(2), 1-22. https://doi.org/10.1145/1993069.1993073

Wood, P. K. (1983). Inquiring systems and problem structure: Implications for cognitive development. *Human development*, *26*(5), 249-265. https://doi.org/10.1159/000137808

Wouters, P. (2017). Modeling and Worked Examples in Game-Based Learning. In *Instructional Techniques to Facilitate Learning and Motivation of Serious Games* (pp. 185-198). Springer.

Wouters, P., Paas, F., & van Merriënboer, J. J. (2008). How to optimize learning from animated models: A review of guidelines based on cognitive load. *Review of Educational Research*, *78*(3), 645-675. https://doi.org/10.3102/0034654308320320

Wouters, P., Tabbers, H. K., & Paas, F. (2007). Interactivity in video-based models. *Educational Psychology Review*, *19*(3), 327-342. https://doi.org/10.1007/s10648-007-9045-4

Yadav, A., Good, J., Voogt, J., & Fisser, P. (2017). Computational thinking as an emerging competence domain. In *Competence-based vocational and professional education*(pp. 1051-1067). Springer.

Yadav, A., Gretter, S., Good, J., & McLean, T. (2017). Computational thinking in teacher education. In *Emerging research, practice, and policy on computational thinking* (pp. 205-220). Springer.

Yu, C. H., & Ohlund, B. (2010). *Threats to validity of research design*. Creative-wisdome. http://www.creative-wisdom.com/teaching/WBI/threat.shtml

Zavgorodniaia, A. (2020, June 15-19). *Efficient Instructional Design of Programming Examples* [Paper presentation]. The 2020 ACM Conference on Innovation and Technology in Computer Science Education, Trondheim, Norway. https://doi.org/10.1145/3341525.3394006

Zelazo, P. D., Carter, A., Reznick, J. S., & Frye, D. (1997). Early development of executive function: A problem-solving framework. *Review of general psychology*, *1*(2), 198-226. https://doi.org/10.1037/1089-2680.1.2.198

Zhang, N., & Biswas, G. (2019). Defining and assessing students' computational thinking in a learning by modeling environment. In *Computational Thinking Education* (pp. 203-221). Springer.

Zhi, R., Price, T. W., Marwan, S., Milliken, A., Barnes, T., & Chi, M. (2019, February 27- March 2). *Exploring the impact of worked examples in a novice programming environment* [Paper presentation]. The 50th ACM Technical Symposium on Computer Science Education, Minneapolis, MN, United States. https://doi.org/10.1145/3287324.3287385

Zhong, B., Wang, Q., Chen, J., & Li, Y. (2016). An exploration of three-dimensional integrated assessment for computational thinking. *Journal of Educational Computing Research*, *53*(4), 562-590. https://doi.org/10.1177/0735633115608444

Zhou, W., He, Y., & Lu, X. (2016). Scratch deformation mechanism of copper based on acoustic emission. *Insight-Non-Destructive Testing and Condition Monitoring*, *58*(5), 256-263. https://doi.org/10.1088/1757-899X/418/1/012098

Appendices

Appendix A

Articles Used to Conduct Integrative Literature Review

Article Name	Author(s)	Year	Database / Journal
Learning programming via worked-examples: Relation of learning styles to cognitive load.	Abdul-Rahman, S. S., & Du Boulay, B.	2014	ERIC/ Computers in Human Behavior
The Effect of Employing Self-Explanation Strategy with Worked Examples on Acquiring Computer Programing Skills	Alhassan, R.	2017	EBSCO / Journal of Education and Practice
Interactive Example-Based Learning Environments: Using Interactive Elements to Encourage Effective Processing of Worked Examples	Atkinson, R. K., & Renkl, A.	2007	JSTOR / Educational Psychology Review
Learning from Examples: Instructional Principles from the Worked Examples Research	Atkinson, R. K., Derry, S. J., Renkl, A., & Wortham, D.	2000	JSTOR / Review of educational research
Cognitive Load Theory in the Context of Teaching and Learning Computer Programming: A Systematic Literature Review	Berssanette, J. H., & de Francisco, A. C.	2021	SCOPUS / IEEE Transactions on Education.
Developing Educational 3D Games with StarLogo: The Role of Backwards Fading in the Transfer of Programming Experience	Boldbaatar, N., & Şendurur, E.	2019	ERIC / ournal of Educational Computing Research
An Approach to Reducing Cognitive Load in the Teaching of Introductory Database Concepts	Bunch, J. M.	2009	ERIC / Journal of Information Systems Education
The Effect of Animation-Based Worked Examples Supported with Augmented Reality on the Academic Achievement, Attitude and Motivation of Students towards Learning Programming	Cevahir, H., Özdemir, M., & Baturay, M. H.	2022	ERIC/ Participatory Educational Research
A Programming Learning System for Beginners— A Completion Strategy Approach	Chang, K. E., Chiao, B. C., Chen, S. W., & Hsiao, R. S.	2000	SCOPUS/ IEEE Transactions on Education

A Comparison of Single- and Dual-Screen Environment in Programming Language: Cognitive Loads and Learning Effects	Chang, T. W., Hsu, J. M., & Yu, P. T.	2011	EBSCO / Journal of Educational Technology and Society
Davidovic, A., Warren, J., Trichina, E. (2003). Learning benefits of structural example-based adaptive tutoring systems	Davidovic, A., Warren, J., & Trichina, E.	2003	WOS / IEEE Transactions on Education
An Exploration of How a Technology-Facilitated Part-Complete Solution Method Supports the Learning of Computer Programming	Garner, S.	2007	SCOPUS/ Issues in Informing Science & Information Technology
Assessing algorithmic and computational thinking in K-12: Lessons from a middle school classroom	Grover, S.	2017	SCOPUS/ Emerging research, practice, and policy on computational thinking
Improving Engagement in Program Construction Examples for Learning Python Programming	Hosseini, R., Akhuseyinoglu, K., Brusilovsky, P., Malmi, L., Pollari-Malmi, K., Schunn, C., & Sirkiä, T.	2020	ERIC / International Journal of Artificial Intelligence in Education
Learning effectiveness and cognitive loads in instructional materials of programming language on single and dual screens	Hsu, J. M., Chang, T. W., & Yu, P. T.	2012	WOS / Turkish Online Journal of Educational Technology-TOJET
When Does Scaffolding Provide Too Much Assistance? A Code-Tracing Tutor Investigation	Jennings, J., & Muldner, K.	2021	WOS / International Journal of Artificial Intelligence in Education
Infusing Computational Thinking into STEM Teaching.	Jocius, R., O'Byrne, W. I., Albert, J., Joshi, D., Robinson, R., & Andrews, A.	2021	JSTOR / Educational Technology & Society
Computational What? Relating Computational Thinking to Teaching	Kale, U., Akcaoglu, M., Cullen, T., Goh, D., Devine, L., Calvert, N., & Grise, K.	2018	ERIC / TechTrends
A Systematic Literature Review of Automated Feedback Generation for Programming Exercises	Keuning, H., Jeuring, J., & Heeren, B	2018	WOS / ACM Transactions on Computing Education

Examining the Effects of Two Computer Programming Learning Strategies: Self-Explanation versus Reading Questions and Answers	Lee, N., & Hong, E.	2017	EBSCO / IAFOR Journal of Education
Cognitive Learning Strategies in an Introductory Computer Programming Course	Mahatanankoon, P., & Wolf, J.	2021	EBSCO / Information Systems Education Journal,
Scaffolding Problem Solving with Learners' Own Self Explanations of Subgoals	Margulieux, L. E., & Catrambone, R.	2021	EBSCO / Journal of Computing in Higher Education
Improving Problem Solving with Subgoal Labels in Expository Text and Worked Examples	Margulieux, L. E., & Catrambone, R.	2016	WOS / Learning and Instruction
Employing subgoals in computer programming education.	Margulieux, L. E., Catrambone, R., & Guzdial, M.	2016	EBSCO / Computer Science Education
Varying Effects of Subgoal Labeled Expository Text in Programming, Chemistry, and Statistics	Margulieux, L. E., Catrambone, R., & Schaeffer, L. M.	2018	ERIC / Instructional Science
Reducing Withdrawal and Failure Rates in Introductory Programming with Subgoal Labeled Worked Examples	Margulieux, L. E., Morrison, B. B., & Decker, A.	2020	EBSCO / International Journal of STEM Education
Optimizing Worked-Example Instruction in Electrical Engineering: The Role of Fading and Feedback during Problem-Solving Practice	Moreno, R., Reisslein, M., & Ozogul, G.	2009	PROQUEST / Journal of Engineering Education
The curious case of loops	Morrison, B. B., Margulieux, L. E., & Decker, A.	2020	WOS / Computer Science Education
Design and Evaluation of Worked Examples for Teaching and Learning Introductory Programming at Tertiary Level	Nainan, M., & Balakrishnan, B.	2019	ERIC / Malaysian Online Journal of Educational Technology
Example-Based Learning in Heuristic Domains: A Cognitive Load Theory Accoun	Renkl, A., Hilbert, T., & Schworm, S.	2009	ERIC / Educational Psychology Review
Design Principles of Worked Examples: A Review of the Empirical Studies	Shen, C. Y., & Tsai, H. C.	2009	ERIC / Journal of Instructional Psychology

Teaching tip: Adaptive Instruction to Learner Expertise with Bimodal Process-oriented Worked-out Examples	Si, J., Kim, D., & Na, C.	2014	JSTOR / Journal of Educational Technology and Society
Ingame Worked Examples Support as an Alternative to Textual Instructions in Serious Games About Programming	Toukiloglou, P., & Xinogalos, S.	2022	SCOPUS / Journal of Educational Computing Research
Improving First Computer ProgrammingExperiences: The Case of Adapting a Web-Supported and Well-Structured Problem-Solving Method to a Traditional Course.	Uysal, M. P.	2014	EBSCO / Contemporary Educational Technology
Example-Based Learning: Integrating Cognitive and Social-Cognitive Research Perspectives	Van Gog, T., & Rummel, N.	2010	JSTOR / Educational psychology review
Process-Oriented Worked Examples: Improving Transfer Performance Through Enhanced Understanding	Van Gog, T., Paas, F., & Van Merriënboer, J. J.	2004	JSTOR / Instructional science
Writing In-Code Comments to Self-Explain in Computational Science and Engineering Education	Vieira, C., Magana, A. J., Falk, M. L., & Garcia, R. E.	2017	ERIC / ACM Transactions on Computing Education (TOCE)
Integrating Computational Science Tools into a Thermodynamics Course	Vieira, C., Magana, A. J., García, R. E., Jana, A., & Krafcik, M.	2018	ERIC / Journal of Science Education and Technology
Providing Students with Agency to Self-Scaffold in a Computational Science and Engineering Course	Vieira, C., Magana, A. J., Roy, A., & Falk, M.	2021	ERIC /Journal of Computing in Higher Education
Student Explanations in the Context of Computational Science and Engineering Education	Vieira, C., Magana, A. J., Roy, A., & Falk, M. L.	2019	ERIC / Cognition and Instruction
Exploring Design Characteristics of Worked Examples to Support Programming and Algorithm Design	Vieira, C., Yan, J., & Magana, A. J.	2015	SCOPUS/ Journal of Computational Science Education
A problem posing-based practicing strategy for facilitating students' computer programming skills in the team-based learning mode	Wang, X. M., & Hwang, G. J.	2017	JSTOR / Educational Technology Research and Development

Appendix B

Summary of Research Question No.1

Name of the article	Themes of the study (Q1 focused on WEs design)	notes about the theme design	Study methodology
A Comparison of Single- and Dual-Screen Environment in Programming Language: Cognitive Loads and Learning Effects	1-WEs design 2-IDT principles,	animated WEs	Quasi-experiment (quantitative)
Adaptive Instruction to Learner Expertise with Bimodal Process-oriented Worked-out Examples	1-WEs design, 2-IDT principles, 3-factors(environment), 4-successful example	process-oriented	Quasi-experiment (quantitative)
An Approach to Reducing Cognitive Load in the Teaching of Introductory Database Concepts	1-WEs design	sequencing WEs (Multiple scenarios of WEs)	Quasi-experiment (quantitative)
An Exploration of How a Technology-Facilitated Part-Complete Solution Method Supports the Learning of Computer Programming	1-WEs design, . 2-successful example	sequencing WEs (part-complete)	Quasi-experiment (quantitative)
Learning programming via worked-examples: Relation of learning styles to cognitive load.	1-WEs design, 2-factors (environment). 3-successful example	-sequencing WEs (part-complete)	Quasi-experiment (quantitative)
A Programming Learning System for Beginners—A Completion Strategy Approach	1-WEs design, 2-successful example	-sequencing WEs (part-complete)	Quasi-experiment (quantitative)

The Effect of Animation-Based Worked Examples Supported with Augmented Reality on the Academic Achievement, Attitude and Motivation of Students towards Learning Programming	1-WEs design, 2-successful example	-animated wes	Quasi-experiment (quantitative)
Improving Engagement in Program Construction Examples for Learning Python Programming	1-WEs design, 2-IDT principles .	-animated wes	Mixed methods, quasi-experiment (quantitative)
Cognitive Learning Strategies in an Introductory Computer Programming Course	1-WEs design	-sequencing scenarios (repeating the same problems)	Mixed methods, quasi-experiment (quantitative)
Cognitive Load Theory in the Context of Teaching and Learning Computer Programming: A Systematic Literature Review	1-WEs design, 2-IDT principles	-subgoal	Systematic review
Computational What? Relating Computational Thinking to Teaching	1-WEs design	-tutorial instruction as WEs	literature review
Design and Evaluation of Worked Examples for Teaching and Learning Introductory Programming at Tertiary Level	1-WEs design, 2-IDT principles, 3-factors (environment). 4-successful example	-sequencing WEs	Quasi-experiment (quantitative)
Developing Educational 3D Games with StarLogo: The Role of Backwards Fading in the Transfer of Programming Experience	1-WEs design, 2-IDT principles, 3-factors (environment), 4-successful example	-fading	Quasi-experiment (quantitative)
Employing subgoals in computer programming education.	1-WEs design, 2-IDT principles,	-subgoal	Quasi-experiment (quantitative)

	3-successful example		
Examining the Effects of Two Computer Programming Learning Strategies: Self-Explanation versus Reading Questions and Answers	1-WEs design, 2-IDT principles, 3-successful example	-tutorial instruction as WES	Quasi-experiment (quantitative)
Example-Based Learning in Heuristic Domains: A Cognitive Load Theory	1-WEs design, 2-successful example	-tutorial instruction as WES	literature review
Improving Problem Solving with Subgoal Labels in Expository Text and Worked Examples	1-WEs design, 2-IDT principles,	-subgoal	Quasi-experiment (quantitative)
Infusing Computational Thinking into STEM Teaching: From Professional Development to Classroom Practice	1-WEs design, (only)	-tutorial instruction as WEs	Quasi-experiment (quantitative)
Ingame Worked Examples Support as an Alternative to Textual Instructions in Serious Games About Programming	1-WEs design, (only)	-integrated with games	Quasi-experiment (quantitative)
Integrating Computational Science Tools into a Thermodynamics Course	1-WEs design, (only)	-animated wes	Quasi-experiment (quantitative)

Learning effectiveness and cognitive loads in instructional materials of programming language on single and dual screens.	1-WEs design, 2-IDT principles, 3-factors (environment)	-animated wes	Quasi-experiment (quantitative)
Optimizing Worked-Example Instruction in Electrical Engineering: The Role of Fading and Feedback during Problem-Solving Practice	1-WEs design,	-fading	Quasi-experiment (quantitative)
Process-Oriented Worked Examples: Improving Transfer Performance Through Enhanced Understanding	1-WEs design, 2-IDT principles	-process-oriented	literature review
Reducing Withdrawal and Failure Rates in Introductory Programming with Subgoal Labeled Worked Examples	1-WEs design,	-subgoal	Quasi-experiment (quantitative)
Scaffolding Problem Solving with Learners' Own Self Explanations of Subgoals	1-WEs design, 2-IDT principles, 3-successful example	-subgoal	Quasi-experiment (quantitative)
The curious case of loops	1-WEs design,	-subgoal	Quasi-experiment (quantitative)
Varying Effects of Subgoal Labeled Expository Text in Programming, Chemistry, and Statistics	1-WEs design,	-subgoal	Quasi-experiment (quantitative)

When Does Scaffolding Provide Too Much Assistance? A Code-Tracing Tutor Investigation	1-WEs design,	-tutorial instruction as WES	Quasi-experiment (quantitative)
Davidovic, A., Warren, J., Trichina, E. (2003). Learning benefits of structural example-based adaptive tutoring systems	1-WEs design, 2-successful example	-multiple worked examples, -complete-partial	Quasi-experiment (quantitative)
Assessing algorithmic and computational thinking in K-12: Lessons from a middle school classroom	1-WEs design, ,	-tutorial instruction as WES	Quasi-experiment (quantitative)
Exploring Design Characteristics of Worked Examples to Support Programming and Algorithm Design	1-WEs design, 2-IDT principles, 3-factors (environment). 4-successful example	-fading	Mixed method

Appendix C

Summary of Research Question No.2

Name of the article	Themes of the study (Q2 focused on IDT principles)	notes about the theme design	Study methodology
A Comparison of Single- and Dual-Screen Environment in Programming Language: Cognitive Loads and Learning Effects	1-WEs design, 2-IDT principles,	- multimedia, - split attention	Quasi-experiment (quantitative)
Adaptive Instruction to Learner Expertise with Bimodal Process-oriented Worked-out Examples	1-WEs design, 2-IDT principles, 3-factors (environment). 4-successful example	- multimedia	Quasi-experiment (quantitative)
Cognitive Load Theory in the Context of Teaching and Learning Computer Programming: A Systematic Literature Review	1-WEs design, 2-IDT principles	- completion, - self-explanation , - split attention	Systematic review
Improving Engagement in Program Construction Examples for Learning Python Programming	1-WEs design, 2-IDT principles .	- self explanation	Mixed methods - Quasi-experiment (quantitative)
Design and Evaluation of Worked Examples for Teaching and Learning Introductory Programming at Tertiary Level	1-WEs design, 2-IDT principles, 3-factors (environment). 4-successful example	- self explanation	Quasi-experiment (quantitative)
Design Principles of Worked Examples: A Review of the Empirical Studies	1-IDT principles	8 principles in general	literature review

Developing Educational 3D Games with StarLogo: The Role of Backwards Fading in the Transfer of Programming Experience	1-WEs design, 2-IDT principles, 3-factors (environment). 4-successful example	- fading principle	Quasi-experiment (quantitative)
Employing subgoals in computer programming education.	1-WEs design, 2-IDT principles, 3-successful example	- self explanation	Quasi-experiment (quantitative)
Examining the Effects of Two Computer Programming Learning Strategies: Self-Explanation versus Reading Questions and Answers	1-WEs design, 2-IDT principles, 3-successful example	- multimedia	Quasi-experiment (quantitative)
Improving Problem Solving with Subgoal Labels in Expository Text and Worked Examples	1-WEs design, 2-IDT principles .	- self explanation	Quasi-experiment (quantitative)
Interactive Example-Based Learning Environments: Using Interactive Elements to Encourage Effective Processing of Worked Examples	1-IDT principles, 2-factors (environment)	- self explanation	literature review
Learning effectiveness and cognitive loads in instructional materials of programming language on single and dual screens.	1-WEs design, 2-IDT principles, 3-factors (environment)	- multimedia, - split-attention	Quasi-experiment (quantitative)
Learning from Examples: Instructional Principles from the Worked Examples Research	1-IDT principles,	Design features	literature review
Process-Oriented Worked Examples: Improving Transfer Performance Through Enhanced Understanding	1-WEs design, 2-IDT principles	- process - completion	literature review

Providing Students with Agency to Self-Scaffold in a Computational Science and Engineering Course	1-IDT principles, 2-successful example	- self explanation	Mixed method
Scaffolding Problem Solving with Learners' Own Self Explanations of Subgoals	1-WEs design, 2-IDT principles, 3-successful example	- self explanation	Quasi-experiment (quantitative)
Student Explanations in the Context of Computational Science and Engineering Education	1-IDT principles, 2-factors (environment)	- self explanation	Quasi-experiment (quantitative)
The Effect of Employing Self-Explanation Strategy with Worked Examples on Acquiring Computer Programing Skills	1-IDT principles	- self explanation	Quasi-experiment (quantitative)
Writing In-Code Comments to Self-Explain in Computational Science and Engineering Education	1-IDT principles	- self explanation	Quasi-experiment (quantitative)
Exploring Design Characteristics of Worked Examples to Support Programming and Algorithm Design	1-WEs design, 2-IDT principles, 3-factors (environment). 4-successful example	- self explanation, - split attention - intra inter	Mixed method

Appendix D

Summary of Research Question No.3

Name of article	other themes	notes of theme design
Adaptive Instruction to Learner Expertise with Bimodal Process-oriented Worked-out Examples	1-WEs design, 2-IDT principles, 3-factors (environment). 4-successful example	time of the instruction
Design and Evaluation of Worked Examples for Teaching and Learning Introductory Programming at Tertiary Level	1-WEs design, 2-IDT principles, 3-factors (environment). 4-successful example	student engagement/active
Developing Educational 3D Games with StarLogo: The Role of Backwards Fading in the Transfer of Programming Experience	1-WEs design, 2-IDT principles, 3-factors (environment). 4-successful example	students background and language familiarity
Improving First Computer Programming Experiences: The Case of Adapting a Web-Supported and Well-Structured Problem-Solving Method to a Traditional Course.	1-factors (environment). 2-successful example	students background and language familiarity
Interactive Example-Based Learning Environments: Using Interactive Elements to Encourage Effective Processing of Worked Examples	1-IDT principles, 2-factors (environment)	student engagement/active
Learning effectiveness and cognitive loads in instructional materials of programming language on single and dual screens.	1-WEs design, 2-IDT principles, 3-factors (environment)	student engagement/active
Student Explanations in the Context of Computational Science and Engineering Education	1-IDT principles 2-factors (environment)	student engagement/active (communication)
Learning programming via worked-examples: Relation of learning styles to cognitive load.	1-WEs design, 2-factors (environment). 3-successful example	student engagement/ active (learning style)
Exploring Design Characteristics of Worked Examples to Support Programming and Algorithm Design	1-WEs design, 2-IDT principles, 3-factors (environment), 4-successful example	Features

Appendix E

Summary of Research Question No.4

Name of article	other themes	notes of theme design
A problem posing-based practicing strategy for facilitating students' computer programming skills in the team-based learning mode	1-successful example (it is only support using examples with suggesting guidance and collaborative to support learners)	- online environment via Moodle
Adaptive Instruction to Learner Expertise with Bimodal Process-oriented Worked-out Examples	1-WEs design, 2-IDT principles, 3-factors (environment). 4-successful example	- online learning environment
Design and Evaluation of Worked Examples for Teaching and Learning Introductory Programming at Tertiary Level	1-WEs design, 2-IDT principles, 3-factors (environment), 4-successful example	- Design WE by using a web page
Developing Educational 3D Games with StarLogo: The Role of Backwards Fading in the Transfer of Programming Experience	1-WEs design, 2-IDT principles, 3-factors (environment), 4-successful example	- Using 3D environment within block based programming
Employing subgoals in computer programming education.	1-WEs design, 2-IDT principles, 3-successful example	- app inventors
Examining the Effects of Two Computer Programming Learning Strategies: Self-Explanation versus Reading Questions and Answers	1-WEs design, 2-IDT principles, 3-successful example	- Tutorial (abstract)
Improving First Computer Programming Experiences: The Case of Adapting a Web-Supported and Well-Structured Problem-Solving Method to a Traditional Course.	1-factors (environment), 2-successful example	- Following Jonassen model with the WEs
Providing Students with Agency to Self-Scaffold in a Computational Science and Engineering Course	1-IDT principles, 2-successful example	- environment, - an experiment for studying MATLAB
Scaffolding Problem Solving with Learners' Own Self Explanations of Subgoals	1-WEs design, 2-IDT principles, 3-successful example	- Scaffolding technique with app inventor

An Exploration of How a Technology-Facilitated Part-Complete Solution Method Supports the Learning of Computer Programming	1-WEs design, 2-successful example	- Specific tool
Learning programming via worked-examples: Relation of learning styles to cognitive load.	1-WEs design, 2-factors (environment), 3-successful example	- interface of web page
The Effect of Animation-Based Worked Examples Supported with Augmented Reality on the Academic Achievement, Attitude and Motivation of Students towards Learning Programming	1-WEs design, 2-successful example	- Tutorial and using AR
Exploring Design Characteristics of Worked Examples to Support Programming and Algorithm Design	1-WEs design, 2-IDT principles, 3-factors (environment), 4-successful example	- Examples with visuals

Appendix F

Evaluation Phase

Name of the article	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Overall appraisal	Notes
Learning programming via worked-examples: Relation of learning styles to cognitive load.	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs in intro to programming course.
The Effect of Employing Self-Explanation Strategy with Worked Examples on Acquiring Computer Programing Skills	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the IDT principles for the algorithm design with WEs in intro to programming course.
Developing Educational 3D Games with StarLogo: The Role of Backwards Fading in the Transfer of Programming Experience	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding design WEs, the IDT principles, factors and has a successful example with StarLogoBlocks.
An Approach to Reducing Cognitive Load in the Teaching of Introductory Database Concepts	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes information regarding WEs design for data analysis and data collection.
The Effect of Animation-Based Worked Examples Supported with Augmented Reality on the Academic Achievement, Attitude and	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs in intro to programming course with and example.

Motivation of Students towards Learning Programming											
A Programming Learning System for Beginners—A Completion Strategy Approach	yes	no	no	no	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs in intro to programming course with an example.
A Comparison of Single- and Dual-Screen Environment in Programming Language: Cognitive Loads and Learning Effects	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding design WEs, the IDT principles for algorithm design with WEs in intro to programming.
Learning benefits of structural example-based adaptive tutoring systems	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs in JavaScript course.
An Exploration of How a Technology-Facilitated Part-Complete Solution Method Supports the Learning of Computer Programming	yes	no	no	no	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the evaluation skill with WEs in intro to programming course with an example.
Assessing algorithmic and computational thinking in K-12: Lessons from a middle school classroom	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the design of WEs in an intro to programming course to enhance all 4 computational thinking skills.
Improving Engagement in Program Construction Examples for Learning Python Programming	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs in Python course with providing an example.
Learning effectiveness and cognitive loads in	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design

instrumental materials of programming language on single and dual screens.											with WEs and factors that affect the designing an intro to programming course while providing an example.
When Does Scaffolding Provide Too Much Assistance? A Code-Tracing Tutor Investigation	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs in Python course.
Infusing Computational Thinking into STEM Teaching.	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs in Snap! programming course to promote pattern recognition, abstraction, algorithm design, decomposition.
Examining the Effects of Two Computer Programming Learning Strategies: Self-Explanation versus Reading Questions and Answers	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs ,factors , IDT principles that affect designing JavaScript courses while providing an example.
Cognitive Learning Strategies in an Introductory Computer Programming Course	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs in intro to programming course.
Scaffolding Problem Solving with Learners' Own Self Explanations of Subgoals	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs and IDT principles in the Maker app course with providing an example.
Improving Problem Solving with Subgoal Labels in Expository Text and Worked Examples	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs and IDT principles in intro to programming course.

Employing subgoals in computer programming education.	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs and IDT principles in intro to programming course with providing an example.
Varying Effects of Subgoal Labeled Expository Text in Programming, Chemistry, and Statistics	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs in intro to programming course.
Reducing Withdrawal and Failure Rates in Introductory Programming with Subgoal Labeled Worked Examples	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs in intro to programming course.
Optimizing Worked-Example Instruction in Electrical Engineering: The Role of Fading and Feedback during Problem-Solving Practice	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs in intro to programming course.
The curious case of loops	yes	yes	no	no	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs in intro to programming course.
Design and Evaluation of Worked Examples for Teaching and Learning Introductory Programming at Tertiary Level	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs, factors affecting the design, and IDT principles in intro to programming course with providing an example.
Teaching tip: Adaptive Instruction to Learner Expertise with Bimodal Process-oriented Worked-out	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs, factors affecting the design, and IDT principles in C programming

Examples											course with providing an example.
Ingame Worked Examples Support as an Alternative to Textual Instructions in Serious Games About Programming	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the algorithm design with WEs in the Blockly programming environment.
Improving First Computer ProgrammingExperiences: The Case of Adapting a Web-Supported and Well-Structured Problem-Solving Method to a Traditional Course.	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding factors that are affecting the algorithm design in the intro to programming course.
Writing In-Code Comments to Self-Explain in Computational Science and Engineering Education	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding IDT principles that are used with the algorithm design in the MATLAB course.
Integrating Computational Science Tools into a Thermodynamics Course	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding the design of algorithmic modeling in thermodynamics courses.
Student Explanations in the Context of Computational Science and Engineering Education	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study includes all the information needed regarding IDT principles and the factors that affect the design of algorithms with enhancing decomposition skill in the MATLAB course.
A problem posing-based practicing strategy for facilitating students' computer programming skills in the team-based learning mode	yes	yes	yes	yes	yes	yes	yes	yes	yes	include	The study provides an example in designing algorithms with WEs in C programming course.

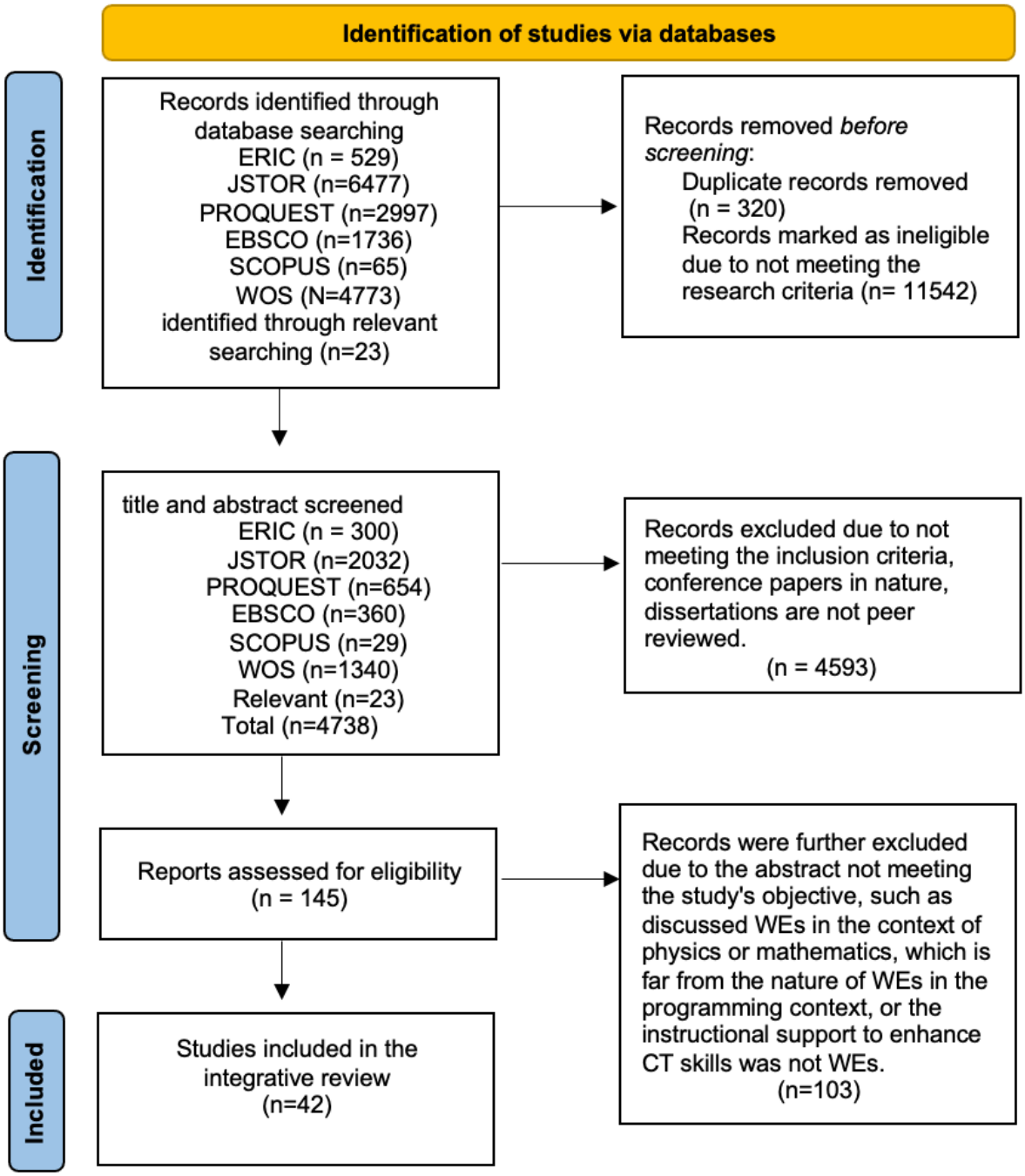
Name of the article	The purpose of the theory	assumption/ statement	structure	usefulness	gene raliz ation	Testability
Interactive Example-Based Learning Environments: Using Interactive Elements to Encourage Effective Processing of Worked Examples	yes	yes	yes	yes	yes	yes
Learning from Examples: Instructional Principles from the Worked Examples Research	yes	yes	yes	yes	yes	yes
Computational What? Relating Computational Thinking to Teaching	yes	yes	yes	yes	yes	It provides several examples about how to apply it in different fields
Example-Based Learning in Heuristic Domains: A Cognitive Load Theory	yes	yes	yes	yes	yes	yes
Design Principles of Worked Examples: A Review of the Empirical Studies	yes	yes	yes	yes	yes	yes
Example-Based Learning: Integrating Cognitive and Social-Cognitive Research Perspectives	yes	yes	yes	yes	yes	yes
Process-Oriented Worked Examples: Improving Transfer Performance Through Enhanced Understanding	yes	yes	yes	yes	yes	yes

Name of the article	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Overall appraisal
Cognitive Load Theory in the Context of Teaching and Learning Computer Programming: A Systematic Literature Review	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	Included
A Systematic Literature Review of Automated Feedback Generation for Programming Exercises	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	included

Name of the article	Q1	Q2	Q3	Q4	Q5	Notes
Providing Students with Agency to Self-Scaffold in a Computational Science and Engineering Course	yes	yes	yes	yes	yes	a design based research DBR with the focus on qualitative design
Exploring Design Characteristics of Worked Examples to Support Programming and Algorithm Design	yes	yes	yes	yes	yes	-

Appendix G

PRISMA flow diagram of the collected data



PRISMA flow diagram (adapted from Page et al. (2021)