VISUALISED WORKED EXAMPLES FOR LEARNING INTRODUCTORY PROGRAMMING AT TERTIARY LEVEL

MARIAM NAINAN A/P T. K. NAINAN

SULTAN IDRIS EDUCATION UNIVERSITY

2021

VISUALISED WORKED EXAMPLES FOR LEARNING INTRODUCTORY PROGRAMMING AT TERTIARY LEVEL

MARIAM NAINAN A/P T. K. NAINAN

05-4506832 pustaka.upsi.edu.my Perpustakaan Tuanku Bainun Kampus Sultan Abdul Jalil Shah PustakaTBainun ptbupsi

THESIS PRESENTED TO QUALIFY FOR A
DOCTOR OF PHILOSOPHY

FACULTY OF ART, COMPUTING, AND CREATIVE INDUSTRY
SULTAN IDRIS EDUCATION UNIVERSITY

2021

05-4506832 pustaka.upsi.edu.my Perpustakaan Tuanku Bainun Kampus Sultan Abdul Jalil Shah PustakaTBainun ptbupsi

UNIVERSITI PENDIDIKAN SULTAN IDRIS
SULTAN IDRIS EDUCATION UNIVERSITY

Please tick (√)
Project Paper ☐
Masters by Research ☐
Master by Mixed Mode ☐
PhD ☑

INSTITUTE OF GRADUATE STUDIES

DECLARATION OF ORIGINAL WORK

This declaration is made on the 16 day of Feb 20 21

i. Student's Declaration:

I, Mariam Nainan A/P T.K. Nainan P20131001223 FSKIK (PLEASE INDICATE STUDENT'S NAME, MATRIC NO. AND FACULTY) hereby declare that the work entitled Visualised Worked Examples for Learning Introductory Programming at Tertiary Level is my original work. I have not copied from any other students' work or from any other sources except where due reference or acknowledgement is made explicitly in the text, nor has any part been written for me by another person.

Signature of the student

ii. Supervisor's Declaration:

I Balamuralithara A/L Balakrishnan (SUPERVISOR'S NAME) hereby certifies that the work entitled Visualised Worked Examples for Learning Introductory Programming at Tertiary Level (TITLE) was prepared by the above named student, and was submitted to the Institute of Graduate Studies as a * ~~partial~~/full fulfillment for the conferment of PhD (Instructional Technology) (PLEASE INDICATE THE DEGREE), and the aforementioned work, to the best of my knowledge, is the said student's work.

16/02/2021
Date

Signature of the Supervisor

UNIVERSITI PENDIDIKAN SULTAN IDRIS
SULTAN IDRIS EDUCATION UNIVERSITY

INSTITUT PENGAJIAN SISWAZAH /
INSTITUTE OF GRADUATE STUDIES

BORANG PENGESAHAN PENYERAHAN TESIS/DISERTASI/LAPORAN KERTAS PROJEK
DECLARATION OF THESIS/DISSERTATION/PROJECT PAPER FORM

Tajuk / *Title*: Visualised Worked Examples for Learning Introductory Programming at Tertiary Level

No. Matrik /*Matric No.*: P20131001223

Saya / *I* : Mariam Nainan A/P T.K. Nainan

(Nama pelajar / *Student's Name*)

mengaku membenarkan Tesis/~~Disertasi/Laporan Kertas Projek~~ (Kedoktoran/~~Sarjana~~)* ini disimpan di Universiti Pendidikan Sultan Idris (Perpustakaan Tuanku Bainun) dengan syarat-syarat kegunaan seperti berikut:-
acknowledged that Universiti Pendidikan Sultan Idris (Tuanku Bainun Library) reserves the right as follows:-

1. Tesis/Disertasi/Laporan Kertas Projek ini adalah hak milik UPSI.
   The thesis is the property of Universiti Pendidikan Sultan Idris
2. Perpustakaan Tuanku Bainun dibenarkan membuat salinan untuk tujuan rujukan dan penyelidikan.
   Tuanku Bainun Library has the right to make copies for the purpose of reference and research.
3. Perpustakaan dibenarkan membuat salinan Tesis/Disertasi ini sebagai bahan pertukaran antara Institusi Pengajian Tinggi.
   The Library has the right to make copies of the thesis for academic exchange.
4. Sila tandakan (√) bagi pilihan kategori di bawah */ Please tick (√) from the categories below:-*

☐	**SULIT/*CONFIDENTIAL***	Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub dalam Akta Rahsia Rasmi 1972. / *Contains confidential information under the Official Secret Act 1972*
☐	**TERHAD/*RESTRICTED***	Mengandungi maklumat terhad yang telah ditentukan oleh organisasi/badan di mana penyelidikan ini dijalankan. / *Contains restricted information as specified by the organization where research was done.*
√	**TIDAK TERHAD / *OPEN ACCESS***	

(Tandatangan Pelajar/ Signature)

Tarikh: 22/02/2021

(Tandatangan Penyelia / *Signature of Supervisor*)
& (Nama & Cop Rasmi / *Name & Official Stamp*)

Balamuralithara A/L Balakrishnan

Catatan: Jika Tesis/Disertasi ini **SULIT @ TERHAD**, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali sebab dan tempoh laporan ini perlu dikelaskan sebagai **SULIT** dan **TERHAD**.

Notes: If the thesis is CONFIDENTAL or RESTRICTED, please attach with the letter from the related authority/organization mentioning the period of confidentiality and reasons for the said confidentiality or restriction.

ACKNOWLEDGEMENT

This study was made possible with the support of the following: supervisor Associate Professor Dr. Balamuralithara A/L Balakrishnan, who gave guidance, support, and encouragement throughout the process; co-supervisor Professor Dr. Ahmad Zamzuri Bin Mohamad Ali as well as other academic staff of Faculty of Art, Computing and Creative Industry, who gave valuable input; friends who were supportive in different ways; family members who gave their prayerful support; my parents who gave loving, prayerful, and sacrificial support; and my God, Heavenly Father, and Saviour, who sustained, guided, and strengthened me.

ABSTRACT

The objectives of this study were to design and develop visualised worked examples for introductory programming at tertiary level, evaluate their effectiveness compared to subgoal labelled worked examples, explore students' engagements with visualised worked examples, and explore students' preferences and perceptions of the two types of worked examples. Quasi-experiment was conducted with 87, 79, and 78 students in three sessions in an introductory programming course in a foundation programme at a university in Selangor. Test data were collected and analysed using analysis of covariance and chi square tests. Students' engagements with visualised worked examples were observed and analysed qualitatively. Another intervention was conducted with 38 students in undergraduate programmes from the same university, who were presented both types of worked examples. Questionnaire data were collected and analysed quantitatively and qualitatively. The findings of this study showed no significant differences in effectiveness for knowledge and skill development but, for programming language and patterns knowledge development, pattern applications were significantly associated with type of worked examples ($\chi^2(2) = 16.48$, $p < .001$; $\chi^2(2) = 11.18$, $p = .004$; $\chi^2(1) = 5.07$, $p = .024$). Also, students were engaged with visualised worked examples. Additionally, 73.7% of the students preferred visualised worked examples and students perceived that visualised worked examples supported their understanding in various aspects. The conclusion was that visualised worked examples were able to significantly reduce the likelihood of wrong or omitted program statements in students' pattern applications. Also, students were engaged with visualised worked examples behaviourally, and by implication, cognitively. In addition, visualised worked examples were preferred by more students with positive perceptions. The implications were that this study extended research on worked example design, employing concepts of attention cueing and learner control, for programming education and provided empirical evidence of worked examples usage for programming education practice.

CONTOH-CONTOH KERJA DIVISUALISASI UNTUK PEMBELAJARAN PENGATURCARAAN PENGENALAN DI PERINGKAT PENGAJIAN TINGGI

ABSTRAK

Objektif-objektif kajian ini adalah untuk merancang dan mengembangkan contoh-contoh kerja divisualisasi untuk pengaturcaraan pengenalan di peringkat pengajian tinggi, menilai keberkesanannya berbanding dengan contoh-contoh kerja dilabel matlamat kecil, meneroka penglibatan pelajar dengan contoh-contoh kerja divisualisasi, dan meneroka pilihan dan persepsi pelajar terhadap dua jenis contoh kerja. Kuasi-eksperimen dilakukan dengan 87, 79, dan 78 orang pelajar dalam tiga sesi dalam kursus pengaturcaraan pengenalan dalam program asasi di sebuah universiti di Selangor. Data ujian dikumpulkan dan dianalisis mengguna analisis kovarians dan ujian chi square. Penglibatan para pelajar dengan contoh-contoh kerja divisualisasi diperhatikan dan dianalisis secara kualitatif. Intervensi tambahan dilakukan dengan 38 orang pelajar dalam program sarjana muda dari universiti yang sama, yang diberikan kedua-dua jenis contoh kerja. Data soal selidik dikumpulkan dan dianalisis secara kuantitatif dan kualitatif. Dapatan kajian ini menunjukkan tiada perbezaan yang signifikan dalam keberkesanan terhadap pengembangan pengetahuan dan kemahiran tetapi, terhadap pengembangan pengetahuan bahasa dan corak-corak pengaturcaraan, aplikasi corak berhubung secara signifikan dengan jenis contoh kerja ($\chi^2(2) = 16.48$, $p < .001$; $\chi^2(2) = 11.18$, $p = .004$; $\chi^2(1) = 5.07$, $p = .024$). Juga, para pelajar terlibat dengan contoh-contoh kerja divisualisasi. Tambahan pula, 73.7% daripada para pelajar suka contoh kerja divisualisasi dan pelajar-pelajar berpendapat bahawa contoh kerja divisualisasi menyokong pemahaman mereka dalam pelbagai aspek. Kesimpulannya ialah contoh kerja divisualisasi dapat mengurangkan dengan signifikan kemungkinan penyata-penyata aturcara yang salah diguna atau tertinggal dalam aplikasi corak para pelajar. Juga, pelajar-pelajar terlibat dengan contoh kerja divisualisasi secara tingkah laku, dan dengan implikasinya, secara kognitif. Tambahan pula, contoh kerja divisualisasi disukai oleh lebih ramai pelajar dengan persepsi-persepsi positif. Implikasi-implikasi adalah bahawa kajian ini memperluaskan penyelidikan mengenai reka bentuk contoh kerja, yang mengguna konsep petunjuk perhatian dan kawalan pelajar, untuk pendidikan pengaturcaraan dan memberi bukti empirikal bagi penggunaan contoh-contoh kerja untuk amalan pendidikan pengaturcaraan.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ANCOVA	Analysis of Covariance
CSS	Cascading Style Sheets
En	Expert n
HTML	Hyper Text Markup Language
RQn	Research Question n
Sn	Student n
SPn	Subproblem n
SPSS	Statistical Package for the Social Sciences
WEn	Worked Example n

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Programming education is mandatory for students in universities who are enrolled in computing-related undergraduate degree programmes, such as computer science or software engineering. Students in such programmes typically enrol in an introductory programming course in their first year of study, and subsequently, advanced programming courses. Moreover, nowadays, the use of information and communication technology is so widespread that even science and engineering undergraduate degree students have to learn programming as well. Such students are usually required to take at least an introductory programming course (Malhotra & Anand, 2019). Furthermore, with the advancement and importance of Industry 4.0, which has information and communication technology as part of its foundation,

programming education is becoming increasingly important (dos Santos et al., 2018). Hence, it is common for students in non-computing-related undergraduate degree programmes to enrol in introductory programming courses. In some universities, foundation programmes leading to an undergraduate degree, may also include an introductory programming course in their curriculum. Additionally, introductory programming courses may be taught at diploma level in colleges.

But, learning introductory programming is challenging for students (Kunkle & Allen, 2016; Lahtinen et al., 2005; Qian & Lehman, 2017). Although diverse methods and tools for learning and teaching introductory programming have been researched and utilised, issues and challenges still persists (Luxton-Reilly et al., 2018; Medeiros et al., 2018). Being the first programming course that students encounter, it is imperative that students in introductory programming courses are assisted in achieving success in their learning, especially in light of relatively high failure rates (Bennedsen & Caspersen, 2019; Medeiros et al., 2018; Watson & Li, 2014) and lack of programming competency after completing the course (McCartney et al., 2013; McCracken et al., 2001).

The current study sought to investigate issues related to development of introductory programming knowledge and skill of students at tertiary education institutions.

1.2 Background to the Study

Learning introductory programming involves learning a programming language and understanding related programming concepts, as well as, learning how to comprehend programs and create programs to solve problems. To help students better understand programming language statements, researchers have developed program visualisation tools, which explain program execution using a simplified model (Luxton-Reilly, 2018; Sorva et al., 2013). Program visualisation tools offer assistance in understanding programs at individual statement level. But, students must also be able to comprehend programs at more abstract level. Studies have shown that students lack abstraction ability for program comprehension (Allen et al., 2017; Busjahn & Schulte, 2013; Luxton-Reilly et al., 2018; Whalley et al., 2006). Busjahn and Schulte (2013) interpreted this lack of ability in relation to the Block model (Schulte, 2008) which explains the complexity of program comprehension.

In addition to learning how to comprehend programs, students are expected to be able to create programs as solutions to problems (Luxton-Reilly et al., 2018; Medeiros et al., 2018; Robins et al., 2003). Researchers have found that, even though students have knowledge of programming language statements, they do not necessarily know how to apply that knowledge to create program solutions (Heinonen et al., 2014; Jenkins, 2002; Lahtinen et al., 2005). Difficulty in designing programs to solve problems was among the issues identified in the context of a Malaysian university as well (Tan et al., 2009).

With regards to helping students learn how to create programs in problem solving, researchers have developed intelligent tutoring systems that guide students while they solve problems (Crow et al., 2018; Lane & VanLehn, 2005). Such systems may assist students in solving the particular problem but may not provide support in generalising from the problem in order to develop knowledge of generalised solutions or patterns. Similarly, online systems have been developed to assist students in learning how to create programs. But, evaluation of the effectiveness of learning from such systems is limited. For example, Lee and Ko (2015) evaluated three different types of online learning resources: a tutorial system, an educational game system, and program creation platform. The researchers found no substantial gain in learning among students for all the three systems. Additionally, Kim and Ko (2017) analysed 30 online tutorial systems to identify features in relation to pedagogical effectiveness. They found that, while most systems emphasised how to apply particular programming language statements, the majority lacked instruction on when and why students should use them. In other words, the goals to be achieved when employing such statements were not emphasised.

Some researchers have stressed that students should have knowledge of programming goals and patterns (Castro & Fisler, 2016; De Raadt et al., 2009; Soloway, 1986). Goals and patterns are generalised information about problems and their solutions (Soloway, 1986). These researchers claimed that the ability to comprehend programs at an abstract level and to create program solutions may be facilitated if students organise their knowledge in the form of goals and patterns (Castro & Fisler, 2016; De Raadt et al., 2009), similar to the manner in which programming experts organise their knowledge (Soloway, 1986). So, they taught

goals and patterns explicitly (De Raadt et al., 2009; Proulx, 2000). The researchers found that students' acquisition and application of goals and patterns knowledge improved. However, adoption of such an instructional approach requires major changes in curriculum, and possibly, assessments as well.

Researchers have also developed tools based on the notion of goals and patterns (Guzdial et al., 1998; Hu et al., 2013). Students used such tools to decompose problems into goals and compose solutions using patterns. However, the tools do not give feedback to students on the correctness of their decomposition and composition processes. So, use of these tools may not lead to accurate structuring of goals and patterns knowledge.

Some researchers have proposed problem-based learning in light of constructivist perspective of learning (e.g., (Kay et al., 2000)). Similarly, others have proposed project-based learning, also based on constructivism (e.g., (Sorva & Seppälä, 2014)). However, in agreement with Clark et al. (2012) as well as Mayer (2004), the current study focussed on the constructivist view of learning rather than the constructivist view for teaching. Problem- and project-based learning are classified as constructivist views for teaching that are informed by constructivist view of learning (Mayer, 2004). Such a view for teaching is premised on the notion that learning should match the practitioners' working situation. Hence, emphasis is given to ill-structured or complex real-world problems and group-based or collaborative learning (Luxton-Reilly et al., 2018). Although such approaches may lead to increased motivation and social interaction, there is lack of consistent evidence in knowledge gains (Luxton-Reilly et al., 2018). Moreover, when teaching new concepts and skills,

researchers have argued that fully guided instruction, or providing information that fully explains the learning material, is more suitable than problem- and project-based learning (Kirschner et al., 2006). The reason for giving complete information is that students will create more accurate knowledge structures and do so more easily (Kirschner et al., 2006).

The foundational principle of constructivism is that students construct relevant knowledge structures in their own minds (Hoy, 2013; Mayer, 2004). They do not passively absorb information as knowledge. Irrespective of whether fully guided or other forms of instruction are employed, learning is only achieved when students engage in active cognitive processing to make sense of instructional material and construct knowledge (Mayer, 2004). However, the instructional approach adopted should be appropriate for the learning situation. Its suitability depends on factors such as the particular subject matter, students' knowledge level, and time constraints (Sorva & Seppälä, 2014). An instructional approach is not necessarily less or more effective compared to other approaches. It is more or less suitable, depending on varying factors (Sorva & Seppälä, 2014).

Problem- or project-based learning may not be appropriate for programming because of the tightly integrated nature of programming concepts (Robins, 2010). Learning a new concept usually requires understanding several other interconnected concepts. Helping students learn a few new concepts well, before moving on to others, may be more suitable for introductory programming courses. Students who struggle with concepts early in the course will have trouble coping with more complex concepts later in the course, which in turn, will contribute to failure in the

course (Luxton-Reilly et al., 2018). Conversely, students who are successful early in the course will find it more manageable when dealing subsequently with advanced concepts (Luxton-Reilly et al., 2018; Robins, 2010). Furthermore, problem-based learning may not be suitable because of the time constraints in a semester-based programme of study.

Introductory programming courses at universities and colleges are typically delivered through lectures and practical sessions in computer laboratories. Traditionally, during practical sessions, the approach used has been to give students problems to solve (Sweller & Cooper, 1985; van Merriënboer, 2013). This is similar to the practice in other domains like mathematics and science. But, during initial stages of knowledge and skill development, learning through problem solving may not be effective (Sweller et al., 2019; van Merriënboer, 2013; van Merriënboer & Sweller, 2005). When students are recently introduced to new programming language statements, they may not have sufficient knowledge to solve problems on their own (Medeiros et al., 2019; van Merriënboer, 2013; van Merriënboer & Sweller, 2005). Furthermore, they may not be able to construct knowledge from the specific problem-solving exercises, more so, when they hastily start with improper solution designs and then make futile attempts at correcting them (Ginat, 2007).

To develop students' introductory programming knowledge and skill within the time schedule of the course, a fully guided instructional approach is deemed to be more suitable. A fully guided approach that has been found to be more effective to help students during initial skill acquisition is example-based learning or learning by studying worked examples (Renkl, 2014; 2017). With example-based learning,

students do not start learning by solving problems. Instead, they study worked examples, which contain problem specifications and complete solutions. After worked example study, students proceed to solve problems on their own.

Learning from worked examples has been found to be more effective than learning through problem solving in domains such as mathematics and science (Sweller & Cooper, 1985; Sweller et al., 2019). However, in order to fully benefit from example-based learning, students must examine and understand how the solution solves the problem (Renkl, 2014). To do so, students must cognitively engage with, (i.e., actively process) the worked example (Renkl, 2014; 2017). Renkl highlighted the principle of self-explanation as crucial for effective example-based learning. Students construct knowledge through their self-explanations.

For the programming domain, the solution in a worked example would consist of a complete program with statements written in a programming language. Students need to understand how the individual statements work. They should also understand how groups of related statements work together to achieve higher-level purposes, that is, to understand the program at a more abstract level. This, in turn, would help them to learn about programming goals. Furthermore, they should generalise from the specific worked examples and recognise patterns in the solutions. In this manner, students would be able to construct knowledge structures in the form of goals and patterns. Ideally, worked examples should help students develop an understanding of the problem solving process as well.

Different worked example designs have been proposed to encourage self-explanation when studying worked examples. In the context of interactive computer-based learning environments, Atkinson and Renkl (2007) proposed three interactive mechanisms: missing steps to be filled by students with feedback, self-explanation prompts with feedback, and help on demand. Although these interactive mechanisms did facilitate learning, the researchers cautioned that they must be carefully designed to trigger processing of relevant aspects of the worked example content. Furthermore, when students' knowledge level is low, filling missing steps and providing self-explanations may cause high processing demands which may overwhelm students (Renkl, 2014). More recently, Renkl (2017) also employed self-explanation prompts.

Another worked example design to foster self-explanation is to embed explanations for steps, or groups of steps, in the solution. These explanations may be inserted in the form of text labels (Catrambone, 1998; Renkl, 2014). The labels are intended to explain that the group of steps achieve a certain subgoal (Catrambone, 1998; Renkl, 2014). Therefore, the labels have been named subgoal labels and the worked example design has been called subgoal labelled worked example design (Catrambone, 1998). For the programming domain, the solution in a subgoal labelled worked example would consist of program statements which are grouped and labelled.

With subgoal labels inserted in different parts of the solution, students are relieved of working out the purposes or subgoals of those parts. Thus, students may be able to understand the program at a more abstract level in terms of goals. This may help them to learn about programming goals. However, even though the explanations

are given, students still need to explain to themselves how the individual program statements work and how they achieve the subgoals. Furthermore, they need to explain to themselves how the subgoals contribute to the overall goal. Once they have knowledge of goals and associated program statements, students should be able to create programs to solve problems with similar goals. Thus, subgoal labelled worked example design seemed to be an appropriate worked example design to be used for teaching and learning introductory programming.

1.3 Problem Statement

Although the effectiveness of subgoal labelled worked examples for learning problem solving has been studied in other domains (Catrambone, 1994; 1996; 1998; Catrambone & Yuasa, 2006; Gerjets et al., 2004), research on subgoal labelled worked examples for introductory programming has been sparse. Recently, Morrison et al. (2015) studied its effect for a text-based programming language. Additionally, Margulieux and Catrambone (2016) investigated its use for a block-based programming language. Both studies showed promising results. A sample worked example with subgoal labels for the programming domain is shown in Table 1.1. It shows a problem specification and complete program solution with embedded labels.

Table 1.1

Worked Example with Subgoal Labels (as Comments) in Bold

Problem	Solution
Write a program to find and display the sum of 5 numbers entered by the user.	**//initialise sum to 0** sum = 0; **//repeat 5 times for 5 numbers** for (i = 0; i < 5; i++) { **//prompt and get number** cout << "Enter a number: "; cin >> number; **//add number to sum** sum = sum + number; } //**display sum of numbers** cout << "Sum is " << sum;

Grouping and labelling program statements in this manner is an application of the meaningful building block principle of example-based learning (Renkl, 2014). It demonstrates that the program is composed of groups of statements (or meaningful building blocks) that achieve certain purposes or subgoals, as described by the labels. It represents the solution as a composition of various building blocks.

The design intentions of subgoal labelled worked examples for teaching and learning introductory programming can be summarised as follows:

- Guide students' attention to the different parts of the program solution by labelling each part to encourage cognitive engagement.
- Assist students to understand that the program solution is made up of parts, each of which achieves a different subgoal.

- Encourage students to explain to themselves how the program statements work to achieve those subgoals and to understand the underlying programming concepts.

But, subgoal labelled worked example design does not illustrate how subgoals are derived from the problem. In other words, it does not illustrate problem analysis. Problem analysis is concerned with identifying the requirements of a problem and decomposing the problem into subproblems. It is important that problem analysis should also be represented in worked examples so that students learn that the problem solving process involves both problem analysis and solution generation.

Emphasising problem analysis is important because studies have shown that students have difficulty in problem analysis and do not adequately analyse problems during problem solving (Hanks & Brandt, 2009; Loksa & Ko, 2016; McCracken et al., 2001). For example, some researchers have commented:

> Many times the students try to solve a problem without completely [understanding] it. Sometimes this happens because the student has difficulties interpreting the problem statement and others simply because students are too anxious to start writing code and don't read and interpret correctly the problem description. (Gomes & Mendes, 2007, p. 2)

Additionally, Qian and Lehman had this to say: "novices often show difficulties in understanding the task and decomposing the problem" (p. 6). Other studies have shown that students have ineffective problem solving behaviour, resulting in random or haphazard activities because they do not follow a systematic problem solving

process of problem analysis and solution generation (Bennedsen & Caspersen, 2005; Gaspar & Langevin, 2007; Heinonen et al., 2014).

It is pointed out here that, in the current study, the terms subproblem and subgoal are used interchangeably. The term subproblem is used, in the context of problem analysis, to refer to the result of decomposition of a problem into smaller parts. The term goal is used, from the perspective of solution generation, to refer to what the solution achieves with respect to the problem. Therefore, the term subproblem and subgoal refer to the same thing but are viewed from different perspectives: subproblem from problem analysis perspective and subgoal from solution generation perspective. Furthermore, the term goal, in general, is used to refer to the overall goal as well as to a subgoal.

When labels are embedded in the solution in subgoal labelled worked examples, the outcomes of problem analysis (i.e., the subproblems) are conflated with the outcomes of solution generation (i.e., composition of parts which achieve different subgoals). The current study proposed that worked examples for teaching and learning introductory programming should emphasise problem analysis in addition to solution generation. Rather than inserting subgoal labels in the solution, subproblems should be presented separately to give more prominence to problem analysis. But, removing labels from the solution means that the physical adjacency between the label and associated group of statements in the program no longer exists. In other words, the connections between subproblems and associated solution parts are lost. These connections are important and must be shown because students need to be able to see which groups of statements are associated with the subproblems. Furthermore, the

connections between subproblems and elements in the problem specification from which they are derived must also be shown. Additionally, these connections should be shown dynamically as and when students need to see them in accordance to their learning needs.

Furthermore, worked examples for teaching and learning introductory programming should assist students to generalise from specific goals and associated program statements and construct knowledge of generalised goals and solution patterns. Knowledge of goals and patterns is helpful for students in program comprehension as well as program creation, as evidenced by researchers who taught goals and patterns explicitly (De Raadt et al., 2009; Soloway, 1986). Moreover, worked examples should demarcate the boundary of a control structure so that the extent of its control is visible. Additionally, worked examples should illustrate how program statements work.

More specifically, the current study proposed that worked examples for teaching and learning introductory programming should have the following design intentions, with the first three being similar to those for subgoal labelled worked examples:

- Guide students' attention to the different parts of the program solution to encourage cognitive engagement.
- Assist students to understand that the program solution is made up of parts, each of which addresses a different subproblem.

- Encourage students to explain to themselves how the program statements work to address those subproblems and to understand the underlying programming concepts.
- Guide students' attention to different elements in the problem specification to encourage cognitive engagement.
- Assist students to understand that the subproblems are derived from different elements in the problem specification as a result of problem analysis.
- Encourage students to generalise from specific goals and solutions to general goals and associated solution patterns.
- Allow students to indicate when and for how long they wish the connections between subproblems and related parts of the worked example should be made visible, according to their learning needs.
- Enable students to see the extent of control of control structures used in the program.
- Enable students to understand how program statements.

1.4 Proposed Worked Example Design

The current study proposed a new worked example design to fulfill the design intentions listed above. In order to emphasise problem analysis, the new worked example design listed the subproblems identified for the given problem separate from the program solution in a new section of the worked example named analysis. The purpose was also to assist students to understand that problem analysis results in a list of subproblems derived from the problem.

But, it was still necessary for students to connect the subproblems to related parts of the worked example. In other words, the new worked example design must visualise, or make visible, the connections. Consequently, the proposed design was named visualised worked example design. The term visualised was taken to mean to make something visible to the eye (Deuter et al, 2015). It was not meant to implicate that visualised worked examples were designed to accommodate visual learning style (Huang, 2019).

The current study proposed to employ technological support to make visible the connections between subproblems and related parts of the worked example by incorporating two concepts of learning technology: attention guidance or cueing (De Koning et al., 2009; Mayer & Moreno, 2003) and learner control or interactivity (Domagk et al., 2010; Landers & Reddock, 2017). Attention cueing makes uses of cues or signals in learning material to guide students' attention to specific parts of the material (De Koning et al., 2009; Mayer & Moreno, 2003). Attention cueing was employed in visualised worked example design to guide students' attention to the different parts of the program solution as well as different elements in the problem specification, connected to a subproblem. So, attention cueing was used to help students visualise the connections between subproblems and related parts of the worked example. It was also used to assist students to understand that the program solution is made up of parts and to explain to themselves how those parts work to address the subproblems. Furthermore, it was to assist students to understand that the subproblems are derived from different elements in the problem specification as a result of problem analysis. Learner control allows a student to manipulate the learning environment in a way that he or she deems is suited for his or her own learning

(Domagk et al., 2010; Landers & Reddock, 2017). Learner control was used in visualised worked example design to allow students to indicate when and for how long they wished the connections between subproblems and related parts of the worked example should be made visible.

Hence, the rationale for employing attention cueing and learner control was to foster engagement with worked exmples and encourage students to self-explain. This, in turn, could lead to better learning. In the context of the current study, visualised worked examples were hypothesised to contribute to development of introductory programming knowledge and skill through engagement with worked examples. This aspect of the conceptual model of the current study is shown in Figure 1.1.

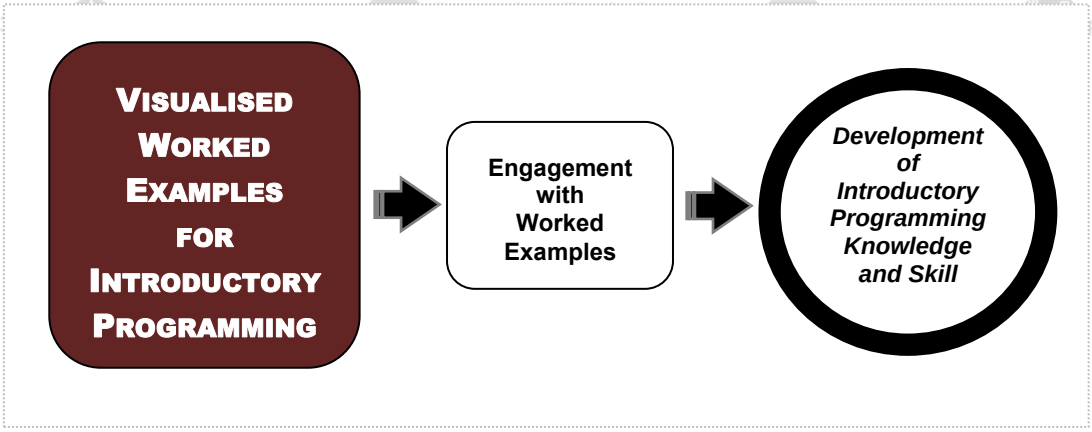

Figure 1.1. Conceptual Model of Current Study

The current study also proposed that visualised worked examples should be presented as a set, following the example set principle (Renkl, 2014). The purpose was to encourage students to generalise from the worked examples and construct knowledge of general goals and solution patterns. Moreover, inspired by the design of block-based programming languages (Maloney et al., 2010; Price & Barnes, 2015),

the current study proposed that the boundaries of control structures in the program solution should be outlined so that they are clearly visible to students. Additionally, to enable students to understand how the program works, the effect of the program in terms of its output for given input should also be illustrated in visualised worked examples.

1.5 Research Objectives

The current study sought to examine issues and challenges of development of introductory programming knowledge and skill and to propose a new worked example design. The current study also sought to compare the new worked example design to subgoal labelled worked example design. (It is noted that the term labelled worked example is used to refer to subgoal labelled worked example for the remainder of the thesis.) More specifically, the objectives of the current study were:

1) To identify issues and challenges of teaching and learning introductory programming at tertiary level and design and develop visualised worked examples, for development of introductory programming knowledge and skill.

2) To investigate the effectiveness of visualised worked examples compared to labelled worked examples for developing students' knowledge and skill, in terms of their performance in solving introductory programming problems.

3) To investigate the effect of visualised worked examples compared to labelled worked examples on the development of students' knowledge of programming language, goals, and patterns for solving introductory programming problems, in terms of pattern application.

4) To explore how students engage with visualised worked examples during their worked example study activity.

5) To explore which worked example design more students prefer for learning introductory programming: visualised worked example design or labelled worked example design.

6) To explore students' perceptions of visualised worked example design compared to labelled worked example design.

1.6 Research Questions and Research Design

The research questions in accordance to the six objectives of the current study were as follows:

RQ1 How should visualised worked examples, used for development of introductory programming knowledge and skill, be designed and developed, to address identified issues and challenges of teaching and learning introductory programming at tertiary level?

RQ2 Is there a difference in effectiveness of visualised worked examples compared to labelled worked examples, for developing students' knowledge and skill, in terms of their performance in solving introductory programming problems?

RQ3 How do visualised worked examples affect development of students' knowledge of programming language, goals, and patterns for solving introductory programming problems, compared to labelled worked examples, in terms of pattern application?

RQ4 How do students engage with visualised worked examples during their worked example study activity?

RQ5 Which worked example design do more students prefer for learning introductory programming: visualised worked example design or labelled worked example design?

RQ6 What are students' perceptions of visualised worked example design compared to labelled worked example design?

The research paradigm adopted for the current study was pragmatism. A combination of quantitative and qualitative approaches was used to best fit the purposes of the study. Accordingly, the research methods employed were chosen based on the research questions. The research method adopted to address RQ1 was a review of the relevant literature and development. For RQ2 and RQ3 to evaluate the

impact of visualised worked examples compared to labelled worked examples on learning, quasi-experiment research method was chosen and data was collected through problem solving assessments. During the quasi-experiment, observation method was also used to collect data on students' engagement with visualised worked examples to answer RQ4. For RQ5 and RQ6 concerning students' preferences for and perceptions of the worked example designs, another intervention study was conducted and data was collected using questionnaires. Development of the research questions and the selected research methods are elaborated in Chapter 3.

1.7 Theoretical Framework

The theoretical framework for the current study is illustrated in Figure 1.2.

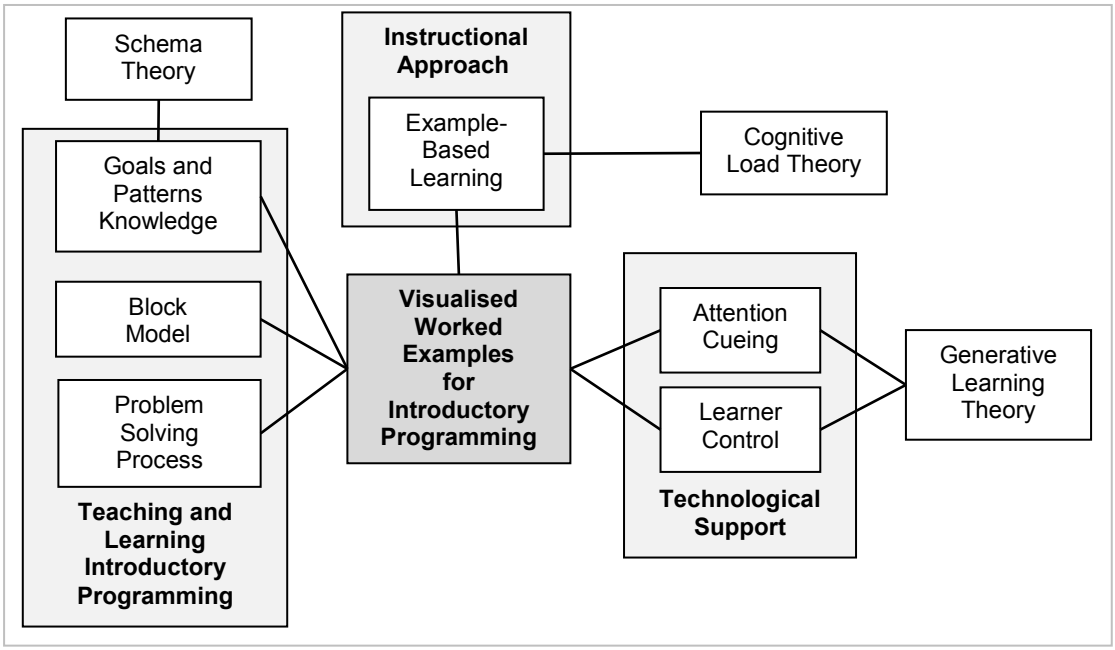

Figure 1.2. Theoretical Framework of Current Study

1.7.1 Example-Based Learning

Example-based learning, or learning from worked examples, is an instructional approach to help students learn how to solve problems by studying worked examples, prior to solving problems on their own (Atkinson et al., 2000; Renkl, 2014; Renkl, 2017; Sweller & Cooper, 1985; Ward & Sweller, 1990). Example-based learning is suitable for initial problem solving skill acquisition when students do not have sufficient domain knowledge (Paas et al., 2003), as explained by cognitive load theory.

1.7.2 Cognitive Load Theory

The theory that explains the effectiveness of learning from worked examples is cognitive load theory (Kalyuga & Singh, 2016; Paas et al., 2003; Sweller et al., 2019; van Merriënboer & Sweller, 2005). The theory explains that human cognitive architecture consists of working memory and long-term memory (Kalyuga & Singh, 2016; Sweller et al., 2019). Active cognitive processing for learning utilises working memory. But, working memory is limited in capacity. When students study worked examples, they do not need to solve the problems. So, cognitive processing demand is reduced. This leaves more working memory capacity for constructing knowledge for storage in long-term memory (Jonassen, 2010; Kalyuga & Singh, 2016; Sweller et al., 2019).

1.7.3 Schema Theory

Schema theory deals with how knowledge is organised in long-term memory. A schema is organised pieces of information representing knowledge related to a topic (Ambrose et al., 2010; Hoy, 2013). A person's knowledge organisation impacts its application. If a person's knowledge is organised in a manner that matches the information processing needs of the situation where it is required, then knowledge application is facilitated (Ambrose et al., 2010).

An expert's knowledge structures or schemas about a topic or domain is well organised with structural connections between the different pieces of information (Ambrose et al., 2010; Hoy, 2013). Experts generalise from specific problems and create schemas of general problem types and associated principles or procedures for their solution (Sweller et al., 2019). When encountering new problems of a problem type, the related schema is retrieved and associated principles or procedures are applied. In this manner, experts are able to solve problems efficiently (Moreno, 2006b).

1.7.4 Goals and Patterns Knowledge

In the programming domain, researchers have theorised that expert programmers organise their knowledge in the form of programming goals and patterns (Soloway, 1986; Soloway & Ehrlich, 1984). This knowledge enables them to comprehend and create programs more effectively and efficiently than novice programmers. Experts

are able recognise instantiations of known patterns in programs, and thereby, determine the goals of the programs (Robins et al., 2003; Soloway, 1986). Similarly, expert programmers are able to identify common goals in new problems and instantiate associated patterns for the solutions (Guzdial et al., 1998).

1.7.5 Block Model

Comprehending programs requires understanding programs at different levels and from different perspectives. The Block model proposed by Schulte (2008) defines four levels, namely, atoms, blocks, relations, and macro structure. Programs can also be understood in three dimensions: text surface, execution, or intention. These different levels and dimensions indicate that program comprehension is complex. The process of program comprehension is also cyclic as a person moves from one level to another for one or more statements and also changes from one dimension to another (Schulte et al., 2010). Understanding programs at the block level in terms of text surface and intention may be facilitated if one has knowledge of goals and patterns.

1.7.6 Problem Solving Process

The problem solving process for the programming domain consists of problem analysis and solution generation (Deek et al., 1999; McCracken et al. 2001; Winslow, 1996). Problem solving is manageable if the problem is decomposed into subproblems through problem analysis (Deek et al., 1999; McCracken et al. 2001). Once

subproblems have been identified, partial solutions for achieving each subproblem can be determined (McCracken et al., 2001). These partial solutions can then be organised and integrated to form the complete solution (Deek et al., 1999; McCracken et al. 2001). The problem solving process for other domains, such as mathematics, similarly involves problem analysis and solution generation (Schoenfeld, 1980).

1.7.7 Generative Learning Theory

According to generative learning theory, learning is a generative activity which results in creation or modification of knowledge structures (Kalyuga & Singh, 2016). Learning occurs when students actively construct meaning from instructional material (Fiorella & Mayer, 2016). Students need to select relevant information to process, organise the information into meaningful relationships, and integrate the information with existing knowledge structures (Fiorella & Mayer, 2016). Generative learning theory is informed by Wittrock's theory of meaningful learning, Mayer's select-organise-integrate model of meaningful learning, and Chi's interactive-constructive-active-passive framework (Kalyuga & Singh, 2016).

1.7.8 Attention Cueing

Cueing or signalling is an instructional tactic to direct students' attention to important elements in instructional material (De Koning et al., 2009; Lorch, 1989; Lorch et al., 2011; Mautone & Mayer, 2001; Mayer & Moreno, 2003). Attention cues are deemed

to be effective in helping students select and organise information and integrate it with existing knowledge (Lorch, 1989; Lorch et al., 2011). Attention cues guide students to relevant information to process. This aspect of attention cueing is important when students do not have sufficient knowledge to differentiate between relevant and irrelevant information. Attention cues help students organise the information into relevant structures and relate them to information they already have (Lorch, 1989; Lorch et al., 2011). In this manner, attention cues facilitate generative learning (Mautone & Mayer, 2001).

1.7.9 Learner Control

Learner control relates to the degree of adjustment a person can make in a computer-based learning environment to suit his or her learning needs (Landers & Reddock, 2017; Carolan et al., 2014). Learner control concept has overlap with the concept of interactivity. Interactivity encourages a learner to engage with the learning environment via physical actions or behavioural activity (Domagk et al., 2010). Behavioural activity generates a response from the environment which may lead to cognitive activity (Domagk et al., 2010). This may ultimately engender generative learning (Carolan et al., 2014).

1.8 Operational Definitions

1.8.1 Introductory Programming Knowledge and Skill

In the current study, introductory programming knowledge was defined as knowledge of a programming language and knowledge of goals and patterns (Castro & Fisler, 2016; De Raadt et al., 2009; Soloway, 1986). Introductory programming skill was defined as skill in comprehending programs and problem solving skill (Robins et al., 2003). Problem solving skill consisted of skill in analysing problems and creating programs as solution to problems (Medeiros et al., 2018; Parsons et al., 2015; Selby, 2015).

1.8.1.1 Development of Knowledge and Skill for Problem Solving

Development of knowledge and skill for problem solving was assessed through problem solving assessments. Students were given introductory programming problems for which they had to write program solutions using a programming language (Luxton-Reilly et al. 2018). Development of introductory programming knowledge and skill was operationalised as performance in these assessments, which was measured as scores computed through quantitative analysis of the students' responses for the assessments. During worked example study, students were expected to comprehend the solutions presented in worked examples. Program comprehension skill was indirectly assessed through the problem solving assessment because students' performance in the programming assessment was an indication of their

ability to comprehend the solutions in the worked examples.

1.8.1.2 Development of Knowledge of Language, Goals, and Patterns

Development of knowledge of programming language, goals, and patterns was assessed through qualitative analysis of students' responses for the problem solving assessments. The analysis involved coding of the responses with respect to application of required pattern components in solutions to the assessments (Castro & Fisler, 2016; Kopec et al., 2007; Seppälä et al., 2015). Development of knowledge of programming language, goals, and patterns was operationalised as correctness of application of required pattern components. Correct application indicated knowledge of the goals, patterns, and the associated programming language statements. Incorrect application or missing pattern components indicated lack of knowledge of the goals, patterns or associated programming language statements.

1.8.2 Example-Based Learning

Example-based learning was defined as an instructional approach where students study worked examples first before they solve problems (Atkinson et al., 2000; Renkl, 2014; 2017). Prior to worked example study, students were presented with information on new programming topics during lectures. The worked examples illustrated the application of the new topics in solutions to programming problems (Atkinson et al., 2000). Example-based learning, or worked example study was used

during practical class sessions when students were at the initial stages of problem solving knowledge and skill acquisition.

1.8.3 Worked Example

A worked example contained a problem specification and a solution which was a complete program that solves the given problem (Renkl, 2014; 2017). In addition, a worked example contained sample runs of the program execution. A worked example was presented to students in a web-based learning environment, with one worked example per web page. Worked examples were presented as a set on multiple web pages (in keeping with the example-based learning example set principle (Renkl, 2014). The web pages were linked in sequence from first to last through hyperlinks.

1.8.4 Labelled Worked Example

A labelled worked example was defined as a worked example where the statements in the program were grouped according to the subgoals they achieved (Catrambone, 1996; 1998; Morrison et al., 2015). Textual labels were inserted at the head of each group of statements to explain the goal achieved. The labels also acted as cues to draw students' attention to the subgoal and associated group of statements (Catrambone, 1996; 1998). Labelled worked example design applied the meaningful building block principle of example-based learning (Renkl, 2014).

1.8.5 Attention Cueing

Attention cueing was defined as an instructional tactic to emphasise certain elements and to guide students' attention to them (De Koning et al., 2009; Lorch et al., 2011). Attention cueing was achieved through labelling or highlighting. Labelling was operationalised as textual labels inserted in the program solution. Highlighting was operationalised as change in background colour of the highlighted textual elements. The highlighting was also operationalised as outlining of the boundary of control structure statements.

1.8.6 Learner Control

Learner control was defined as the control a student has to adjust the learning environment in order to accommodate his or her learning experience (Landers & Reddock, 2017; Carolan et al., 2014). It was the potential given by the learning environment for students to initiate responses from the environment in reaction to students' actions (Domagk et al., 2010). Learner control was operationalised as the potential for students' mouse actions on the interface of the web-based learning environment to trigger responses in the form of highlighting and to navigate through the set of worked examples web pages.

1.8.7 Visualised Worked Example

Visualised worked example was defined as a worked example that had an additional analysis section where subproblems derived from the problem specification were listed. It also employed attention cueing, through highlighting, and learner control. Visualised worked example employed learner control in conjunction with attention cueing so that highlighting was initiated in response to students' mouse actions. The elements that were highlighted were the selected subproblem, the program statements, and the elements in the problem specification associated with that subproblem. All these elements were highlighted simultaneously. The highlighting of program statements associated with the subproblem was an application of the meaning building block principle of example-based learning (Renkl, 2014). If the highlighted program statements were enclosed within or included a control structure, then the control structure boundary was also highlighted.

1.8.8 Engagement with Worked Examples

Engagement with worked examples consisted of three components: behavioural, cognitive, and emotional (Fredricks et al., 2004). These were aligned to three components of the model of interactivity in computer-based learning environments (Domagk et al., 2010), namely, behavioural activity, cognitive/metacognitive activity, and emotion. Behavioural activities were operationalised as students' mouse actions on the interface of the web-based learning environment, which triggered responses. The learning environment's responses were presumed to initiate cognitive and

metacognitive activities which, in turn, led to further behavioural activities. Behavioural activity was observable. Since cognitive activity was not observable, behavioural activity acted as a proxy for cognitive activity. Behavioural activity represented behavioural engagement. Cognitive activity represented cognitive engagement. Emotional engagement was represented by students' perceptions of worked example design with regards to emotion.

1.9 Scope of the Study

The current study focused on learning introductory programming at university or college level. With respect to programming knowledge, an introductory programming course may emphasise one, or sometimes two, of several programming paradigms. The common choices are procedural or imperative, object-oriented, and functional programming (Nandigam & Bathula, 2013). The scope of the current study was limited to the procedural programming paradigm. The factors leading to this choice were:

- The procedural programming concepts of data types, variables, and control structures are listed as programming fundamentals and classified as essential computing foundational knowledge for degree programmes like software engineering according to the ACM/IEEE curricular guidelines (Joint Task Force on Computing Curricula, 2015).

- There has been an emphasis recently on computational thinking (Wing, 2006) which stresses algorithmic construction based on the procedural paradigm (Parsons et al., 2015).

- Learning object-oriented programming usually also involves learning procedural programming. Rist (1996) has argued that learning object-oriented programming adds the burden of learning object-oriented concepts on top of procedural concepts, as cited by Garner et al. (2005). But, learning procedural programming can be done independently of object-oriented programming.

- In interviews conducted with instructors of introductory programming courses in 28 of the 39 Australian universities offering such courses, more than 50% of the instructors chose to teach the procedural paradigm in their courses (Mason et al., 2012). The percentage for object-oriented paradigm was 25.0% and for functional paradigm was 2.3%. Some instructors chose to mix paradigms but typically started with procedural paradigm first and covered object-oriented concepts in the last few weeks. The researchers also commented on the downward trend among instructors teaching an objects-first approach based on comparison of data collected from previous studies. Even though the study considered only Australian institutions and it was conducted several years ago, it seemed reasonable to assume that procedural programming paradigm would still be relevant for introductory programming courses in other countries and at the current time as well.

In summary, learning programming in the context of the current study was limited to learning introductory programming at universities and colleges based on the procedural paradigm.

1.10 Limitations of the Study

The current study utilised a quasi-experiment for evaluating the effectiveness of visualised worked examples compared to labelled worked examples. To enable generalisability of the results to the target population, ideally random sampling and random assignment of students to experimental and control groups should have been carried out (Johnson & Christensen, 2014). The reason for not using random sampling was practical limitations. Random assignment of students to groups was also not carried out because intact classes were used. The whole class of students were assigned as either a control group or an experimental group. However, the quasi-experiment design was strengthened by conducting pretests for both groups in order to examine the equivalence of the groups.

The sample size for the quasi-experiment was also another limitation of the current study. This was due to limited accessibility to classes imposed by practical concerns of instructors of the introductory programming course where the quasi-experiment was conducted. Replication studies in other introductory programming courses in the future should help strengthen the interpretations of the findings of the current study.

Introductory programming courses cover a variety of topics that are considered fundamental to development of introductory programming knowledge and skill. For the quasi-experiment in the current study, the intervention sessions were limited to only three practical class sessions. This limitation was also due to practical concerns of instructors of the course. The topics in those class sessions were selection control structures, repetition control structures, and accumulators. Learning introductory programming entails learning a programming language. For the current study, the programming language used in the course was the C programming language. Future studies should examine further topics and other programming languages.

1.11 Significance of the Study

The current study contributes to society where information and communication technology plays an important role in the daily life of people. Undergraduate engineers and scientists, equipped with good programming knowledge and skill, are better able to utilise such technology for the benefit of the people. Improving learning of introductory programming is important for future engineers and scientists who need the knowledge and skill in the era of Industry 4.0. Universities and colleges that adopt worked example study in their introductory programming courses may enhance the outcomes of programming education of their graduates.

The findings of the current study contribute toward an understanding of how learning technology concepts of attention cueing and learner control may be employed with instructional principles of worked example design to create the new

visualised worked example design for learning introductory programming. It also contributes to research in computer science education in terms of the use of worked examples for programming education.

The findings of the current study provide evidential support to instructors in introductory programming courses who wish to adopt worked example study as an instructional strategy. The current study sought to address the problem of lack of problem analysis skill among students learning introductory programming through deliberations on how to emphasise problem analysis. The findings provide guidelines on how to design worked examples which emphasise problem analysis as well as encourage students to actively process worked examples for effective development of programming knowledge and skill.

The findings of the current study are of benefit to students who may utilise visualised worked examples during practical class sessions or for independent learning, either after a lecture or before a practical class session. This is particularly relevant for introductory programming courses where students face difficulties in solving problems on their own when their knowledge and understanding have not consolidated yet. The new visualised worked example design may foster self-explanation so that they can benefit from worked example study.

1.12 Summary and Organisation of the Thesis

Learning how to program is a must for students who are studying computing-related programmes at tertiary level. Learning programming should result in development of knowledge of programming language, goals and patterns, and the skill to solve programming problems. The current study proposed worked example study as an alternative approach which has been found to be more effective than learning through problem solving. However, to help students gain the potential benefits from worked example study, the worked examples need to be designed to foster engagement. Labelled worked example design may foster engagement but has shortcomings. The current study proposed a new worked example design, visualised worked example design. This chapter presented the research objectives and questions in relation to the design, development, evaluation, and exploration of visualised worked examples in comparison to labelled worked examples. It also presented the conceptual model and theoretical framework for the current study as well as its scope, limitations, and significance.

The remainder of the thesis is structured as follows. Chapter 2 elaborates on the theoretical framework that underpins the current study and presents details of the conceptual model for visualised worked example design. Chapter 3 discusses the overall research methodology and the selected research methods to address the research questions. This is followed by detailed accounts of the implementation of each of the selected research methods. Chapter 4 presents the design, development, and validation of visualised worked example design in relation to the conceptual model presented in Chapter 2. Chapter 5 presents the findings of the current study and

Chapter 6 discusses these findings. Chapter 7 concludes on the findings and presents the contributions of the current study. Future work on visualised worked example design and suggestions for further research concludes the discussion in Chapter 7.