8-2018

# Senior Computer Science Students' Task and Revised Task Interpretation While Engaged in Programming Endeavor

Andreas Febrian
*Utah State University*

# SENIOR COMPUTER SCIENCE STUDENTS' TASK AND REVISED TASK INTERPRETATION WHILE ENGAGED IN PROGRAMMING ENDEAVOR

by

Andreas Febrian

A proposal submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Engineering Education

Approved:

| | |
|---|---|
| _____ | _____ |
| Oenardi Lawanto, Ph.D.<br>Major Professor | Kurt Becker, Ph.D.<br>Committee Member |
| _____ | _____ |
| Ning Fang, Ph.D.<br>Committee Member | Wade Goodridge, Ph.D.<br>Committee Member |
| _____ | _____ |
| Haito Wang, Ph.D.<br>Committee Member | Mark R. McLellan, Ph.D.<br>Vice President for Research and<br>Dean of the School of Graduate Studies |

UTAH STATE UNIVERSITY
Logan, Utah

2018

# ABSTRACT

Senior Computer Science Students' Task and Revised

Task Interpretation while Engaged

In Programming Endeavor

by

Andreas Febrian, Doctor of Philosophy

Utah State University, 2018

Major Professor: Oenardi Lawanto, PhD
Department: Engineering Education

Self-regulated learning is a situated and iterative goal-directed learning process that has a positive influence on students' academic success, problem-solving, and design quality. The heart of self-regulation is task interpretation, which determines students' selection of goals, objectives, criteria for success, and required cognitive strategies. Thus, task interpretation affects the entire problem-solving endeavor. Developing a computer program is a problem-solving process that requires employing various cognitive skills and considers the interplays of varying levels and types of abstractions; its complexity is one of the primary dropout reasons in computer science. Fortunately, learning various self-regulation strategies may help students to persist in computer science. This study aims to assess students' explicit and implicit task interpretation, their revisions, and factors that influence their revisions during a computer programming endeavor.

This study used qualitative case study design with two units of analysis, which were designing an object-oriented system and an algorithm. Two female and two male

senior computer science students were voluntarily recruited as cases. Each participant was asked to answer five programming problems while thinking aloud. In addition, they completed an initial task interpretation survey and answered post-problem solving interview questions for each problem. The participants' problem-solving endeavor were video- and audio-recorded, transcribed, and qualitatively coded by two experts. The average Kappa score was 1.00 suggesting a perfect agreement among coders.

The analysis suggests that the participants were capable of tailoring their problem-solving approach to the problems' characteristics, including when interpreting the tasks. All participants were also competent in interpreting the explicit and implicit aspects of the task and would refine their interpretation during the problem-solving endeavor, especially when the task contains an extensive amount of detail. Further, their competency deteriorated when the participants were overconfident, overwhelmed, utilizing an inappropriate presentation technique, or drawing knowledge from irrelevant experienced. Having an incorrect explicit task interpretation may result in an inaccurate implicit task understanding or even an unsuccessful problem-solving endeavor. Last, the participants tended to assume positively about their problem-solving approach and neglected managing unfavorable outcomes.

(295 pages)

# PUBLIC ABSTRACT

Senior Computer Science Students' Task and Revised

Task Interpretation while Engaged

In Programming Endeavor

Andreas Febrian

Developing a computer program is not an easy task. Studies reported that a large number of computer science students decided to change their major due to the extreme challenge in learning programming. Fortunately, studies also reported that learning various self-regulation strategies may help students to continue studying computer science. This study is interested in assessing students' self-regulation, in specific their task understanding and its revision during programming endeavors. Task understanding is specifically selected because it affects the entire programming endeavor.

In this qualitative case study, two female and two male senior computer science students were voluntarily recruited as research participants. They were asked to think aloud while answering five programming problems. Before solving the problem, they had to explain their understanding of the task and after that answer some questions related to their problem-solving process. The participants' problem-solving process were video- and audio-recorded, transcribed, and analyzed.

This study found that the participants' were capable of tailoring their problem-solving approach to the task types, including when understanding the tasks. Given enough time, the participants can understand the problem correctly. When the task is

complicated, the participants will gradually update their understanding during the problem-solving endeavor. Some situations may have prevented the participants from understanding the task correctly, including overconfidence, being overwhelmed, utilizing an inappropriate presentation technique, or drawing knowledge from irrelevant experience. Last, the participants tended to be inexperienced in managing unfavorable outcomes.

(295 pages)

# ACKNOWLEDGMENTS

All praises belong to Allah, The Most Merciful and The Most Gracious.

This dissertation study was a long and complex process that could not be possible without the support from many people, directly or indirectly. I wish to acknowledge and appreciate them.

To Dr. Oenardi Lawanto for his mentoring in various aspect of this study and self-regulation theory. I have gained an invaluable experience that helps me to grow to be a better researcher and learner. To my committee members, Drs. Kurt Becker, Ning Fang, Wade Goodridge, and Haito Wang for their support and wisdom on this study and broaden my perspective on engineering education research. To Dr. Idalis Villanueva, Dr. Angela Minichiello, and Laura Gelles for their crucial insights in qualitative research method.

To Alicia Melvin, Kamyn Peterson-Rucker, Matthew Eric D'Angelo, and Rosamaria Diaz who have helped me researching computational thinking and pilot testing this dissertation instruments. To my qualitative coders who have spent time and effort in interpreting the participants' transcriptions. To the pilot study and research participants for their cooperation and precious comments in improving this study reliability and impact. To Herika Hayurani and Nova Eka Diana for their valuable inputs on the research instruments.

To Shane Guymon for his willingness to be one of my discussion partners and critical suggestions on my writing style and grammar. To Sarah Lopez and Theresa Green for their suggestions on my writing. To Susan Hammond, Shane Bullock, and Cora Price

for their technical assistance. To Benjamin Call for lending me his research books. To Melissa Scheaffer, Duke Madson, Melanie Faustino Hansen, and Emma for their suggestions and help in reviewing the participants' task interpretation report.

To my family, especially for my mother and wife for their thought, prayers, support, and patience during my doctoral study. Last, but not least, to my friends on campus and the Logan Islamic Center, especially for Ozkan Fidan and Ahsanul Kibria for their companionship and support during this long doctoral study.

Andreas Febrian

CONTENTS

# LIST OF TABLES

Table Page

# LIST OF FIGURES

Figure Page

# CHAPTER I

# INTRODUCTION

## Background of Study

It is one of the digital age's visions to support people's daily activities seamlessly through embedded computing and information technologies (Weisser, 1991). Motivated scientists, engineers, and designers are eagerly finding a way to shorten the gap between the real and digital worlds. It is a long, challenging road, but they have made progress by means of smart-devices, and by integrating advanced computational abilities into existing familiar devices. The idea is to allow these devices to perform their core functions and, on top of that, several computational- and sensor-based operations. This approach is an idea that attracts various companies, national and international, big and small, to develop and deliver their signature smart-devices to the market (Apple Inc., n.d.; Google Inc., n.d.-c; Huawei Technologies Co., n.d.; Mercedes-Benz USA, n.d.; Samsung, n.d.; Smarthome, n.d.).

Astounding as it is, the invention of smart-devices only serves as a gateway to reduce the gap between the real and digital worlds. Some researchers believe that these devices need to assume more active roles in people's daily lives, such as providing in-context assistance (Bughin, Chui, & Manyika, 2010; Froehlich, Chen, Smith, & Potter, 2006; Trinh, Chung, & Kim, 2012). On the other hand, computers are still extensively used everywhere for handling both simple and complex tasks (Bundy, 2007). Some of these applications have integrated artificial intelligence (Geffner, 2014) which allows several job automation (Bui, 2015). Consequently, technology-integrated solutions have become common and set the standard for the next generation of professionals (i.e., having

some basic computer science (CS) skills) (Hambrusch, Hoffmann, Korb, Haugan, & Hosking, 2009; Henderson, 2009).

The CS skills are essential in the future, including for researchers, scientists, and business professionals. Unfortunately, student retention is still a significant problem in computer science (Ambrosio, Almeida, Franco, Martins, & Georges, 2012; Beaubouef & Mason, 2005; Howles, 2007; Kori et al., 2015; Wing, 2006). Most students are dropping out due to the immense challenges faced when learning programming (Anderson & Skwarecki, 1989; Guzdial et al., 2015; Howles, 2007; Kori et al., 2015). Although CS is not entirely about programming, it is still a part of and the most critical CS core skill (Denning et al., 1989; The Joint Task Force on Computing Curricula, 2013). Programming is the most efficient way to learn CS concepts and principles (Gal-Ezer & Harel, 1998; Lye & Koh, 2014; Wing, 2006, 2008). Exposing students to various self-regulation skills could help ease the learning process (Leiviskä & Siponen, 2013) and, at the same time, improve their programming performance (Bergin, Reilly, & Traynor, 2005; Kumar et al., 2005).

CS skills are problem-solving strategies (Glass, 2006), and lack of employing these and self-regulation skills during a problem-solving attempt might lead to failure (Schoenfeld, 1983). Falkner et al., (2014) reported that CS students are unable to align their problem-solving goals with the assessment criteria, which suggests inaccurate task interpretation efforts. Fortunately, students' task understanding evolved throughout the learning endeavor (Rivera-Reyes, 2015). In other words, students monitor their task understanding and approach throughout the learning enterprise (Isomöttönen & Tirronen,

2013). Therefore, understanding students' task interpretation and its revision are crucial for helping students to cope with programming challenges better. After all, every self-regulation activity starts with a task interpretation (Butler & Cartier, 2005).

## Research Questions

The purpose of this study was to investigate CS students' task interpretation during programming. More specifically, this study aimed to assess students' explicit and implicit task interpretation and their revision during the problem-solving process. These three research questions were used to guide the study:

1. What was the students' initial task interpretation (i.e., the explicit and implicit aspects) of the given problems?
2. How did their original understanding change during the problem-solving endeavor?
3. What were the influencing factors for any revisions of their initial task understanding?

## Research Design Overview

The within-site embedded qualitative multiple case study research approach was employed, which meant that this study recruited participants (i.e., multiple cases) at the researcher's institution (i.e., within-site) (Creswell, 2012), where each case consisted of two analysis units (i.e., embedded) (Yin, 2009). The research activities included IRB application, participant recruitment, data collection, preliminary analysis, member checking 1, data analysis, member checking 2, and reporting. All were completed in two semesters. Four senior computer science students at USU were recruited for this study

using convenience and purposeful sampling method. Each participant represented high- and low-performance male and female students. During the three-hour data collection period, each participant solved five programming problems while thinking aloud, filled out open-ended surveys, and answered interview questions; all were audio- and video-recorded. The researcher used a problem-space map during the observation (Johnson, 2008) to minimize observation faulty and uncaptured participants' thought processes, which developed based on the pilot study data. The analysis included organizing, transcribing, coding, analyzing, and triangulating the findings and interpretations of the collected data. In the end, each participant received a $40 Amazon gift card and a personalized SRL report as tokens of appreciation. Chapter IV presents the research design and justification in detail.

## The Significance of the Study

Educational researchers have found a positive relationship between students' problem-solving approach and self-regulation activities (Schoenfeld, 1983). Consequently, enhancing students' self-regulation skills can improve the success rate and quality of their attempt in finding the most appropriate solution for a given problem. A similar expectation is also true in computer science education (CSE), especially in programming problem-solving which is also a form of problem-solving approach (Glass, 2006). Numerous researchers believe students' task interpretation determine their self-regulation activities (Butler & Winne, 1995; Lawanto, Goodridge, & Santoso, 2011). Therefore, understanding students' task interpretation during programming problem solving could benefit all stakeholders: the students, instructors, educational institutions,

and CSE field. For the students, the findings of this study could help them understand the complexity of their thinking process during the programming endeavor. By deepening their appreciation of their thinking process, students could strive to become better self-regulated learners. For the instructors, this study could aid them in developing discipline-specific interventions and instructional approaches that could enhance students' self-regulation skills. The educational institutions will gain indirect impact through the improvement of students' self-regulation with an increase of retention rate. Last, this study contributes to the limited CSE literature on self-regulation during a programming venture, especially in the literature on the revision of students' task interpretation. The proposed method and research findings could aid other researchers who would like to further this investigation.

## Assumptions of the Study

In conducting this study, the researcher used five assumptions. First, the participants could read and communicate in English as expected from a typical US CS senior student. Second, the participants could employ the knowledge gained from the mandatory CS courses (e.g., Introduction to CS, Algorithm, and Data Structure courses) to solve programming problems. Third, the participants gave their best effort in solving all software design problems during the data collection. In addition to this assumption, the researcher provided anonymity, confidentiality, challenging problems, and personalized SRL reports for all participants to motivate them to give their best attempt. Fourth, the video transcription process was conducted with minimum error. Fifth, the utilization of two qualitative coders with minimum 0.81 Kappa score improved the

coding reliabilities. Viera & Garrett (2005) claim that a 0.81 or higher Kappa score can be interpreted as an almost perfect agreement between coders.

## Limitations of the Study

In this study, two male and two female senior CS students from the USU CS Department were recruited. All participants were asked to answer five programming problems in three hours. The analysis was focused on two problems, which were related to designing an object-oriented system and an algorithm. In other words, this study did not assess students self-regulation for all types of problems and programming paradigms. Self-regulation is agent-dependent, which means students might approach the same problem differently. Additionally, all participants were from the CS department at Utah State University (USU). Therefore, it is inappropriate to assume that all CS students always employ the task interpretation strategies found in this study, for all type of problems and in all difficulty levels. Further, due to the small number of applicants, the lowest participants' GPA was still above 3.00 on a 4-point scale, and thus might not fully represent the low-performance CS students. Task interpretation is only one of the factors that influence students' performance. This study omitted the other factors, such as students' motivation and self-efficacy. In term of research method, the thinking aloud might help the participants to self-regulate themselves better (Chi, De Leeuw, Chiu, & Lavancher, 1994) and influence the research results. Unfortunately, there is no known approach to overcome it. Therefore, readers need to be careful in interpreting findings and drawing conclusions from this study.

## Definition of Key Terms

Self-Regulated Learning (SRL): A situated and iterative goal-directed learning process that involves complex and dynamics activities (Butler & Cartier, 2005; Butler, Schnellert, & MacNeil, 2015; Butler & Winne, 1995).

Task interpretation (TI): Students' understanding of the relationship between the task and the required cognitive processes to complete it. (Butler, 1998).

The explicit aspect of task interpretation: "Information that is overtly presented in task descriptions and discussions" (p.2) which includes the task goal(s), requirements, constraints, and standard to be followed (Hadwin, Oshige, Miller, & Wild, 2009).

The implicit aspect of task interpretation: "Information [that] students might be expected to extrapolate beyond the assignment description" (p.2) which includes relevant concepts, knowledge, and cognitive processes (Hadwin et al., 2009).

Monitoring and fix-up: Students' activities of self-monitor progress (monitoring) and adjust goals, plans, or strategies based on self-perceptions of progress or feedback (adjusting approaches to learning) (Butler & Cartier, 2005).

Computer Science (CS): "The systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application" (Denning et al., 1989, p.12).

Computer Science Education (CSE): Any educational activities that enable learners to apply computing principles to any problems (Senske, 2011).

A problem or a task: A question or an issue that need to be examined and solved (Jonassen, 2010) which varies in terms of structuredness (i.e., from well- to ill-

structured), complexity (i.e., static to dynamic), and situatedness (i.e., social aspect of the problem) (Jonassen, 2000).

A design problem: A complex and ill-structured problem which has ambiguous goal specifications, multiple solutions, and the need to incorporate knowledge from various disciplines and domains (Jonassen, 2000) to meet particular needs and constraints (Engineering Accreditation Commission, 2003).

A software design problem: Any design problems in the computer science context where the problem, thought process, and the solution can be represented and carried out effectively by an information-processing agent (Grover & Pea, 2013) through utilization of various fundamental computing concepts (Wing, 2006). It is inherent in the computing discipline that the solution to a software design problem should be correct, accurate, and efficient (Denning et al., 1989).

Problem-solving: “A goal-oriented sequence of cognitive operations” (Anderson, 1980, p.257) to adapt to internal or external demands (Heppner & Krauskopf, 1987).

A programming paradigm: Any approaches that allow programmers to organize computer programming codes so they could focus on solving the problems instead of tinkering with the hardware details (Lee, 2014).

The imperative programming paradigm: An approach to organize computer programming codes where the program is decomposed into several manageable pieces in the forms of sub-programs or sub-routines (Lee, 2014).

Object-oriented programming paradigm: An enhancement of imperative programming paradigm which allows not only the sub-routine organization but also the

structuring of a computer program by defining classes of objects that have specific properties and functions (Lee, 2014).

## Dissertation Outline

This dissertation is organized into six chapters. Chapter I introduces the background, motivations, purpose, research design, assumptions, and limitations of the study. In Chapter II, relevant literature is elucidated to establish a solid basis for the study. The constructs and contexts included in the chapters are self-regulated learning with particular focus on task interpretation, CS, CSE, and software design problem-solving. Chapter III is dedicated to discuss the prior pilot study during the 2016 Research Experience for Undergraduate (REU) program. In this chapter, the lessons learned from the pilot study are reported including the plan to incorporate them into the dissertation study. Chapter IV presents the research methodology and design. In this chapter, the data collection and analysis methods are explicated with its justification. In Chapter V, the participants and findings are discussed to answer the research questions. Chapter VI presents the conclusion of the study, its implication, and recommendation for future studies.

# CHAPTER II

# LITERATURE REVIEW

## Introduction

The purpose of this literature review is to establish a firm foundation for this dissertation research by elucidating the relevant concepts, contexts, and studies based on available literature. In more specific, the objectives of this chapter are to:

1. Describe computer science as a discipline, computer science education, and programming and object-oriented design.
2. Describe the self-regulated learning (SRL) framework with an emphasis on task interpretation, monitoring, and their assessment methods.
3. Describe students' SRL during programming and object-oriented design.

This chapter consists of six sections, which are the introduction, biases and corrective methods, computer science education, task interpretation and monitoring strategies in self-regulated learning, self-regulation during programming and object-oriented design, and summary. The introduction section explicates the purpose and objectives of this literature review. The biases and corrective methods section describes potential biases and methods to minimize them. The computer science (CS) and its education section describes the research context, which includes the discipline of computer science, computer science education, and programming and object-oriented design. Since it is essential to understand the contexts surrounding a self-regulation activity (Butler et al., 2015; Cartier & Butler, 2004), understanding CS as a discipline is a significant step towards understanding students' self-regulation during the software

design endeavor (i.e., the programming and object-oriented design). The self-regulated learning section elucidates the self-regulation framework, task interpretation and monitoring, and assessment methods. The section after that discusses the CS students' self-regulation during programming and object-oriented design. Last, a summary of this chapter is provided.

## Biases and Corrective Methods

Biases occur when people use heuristics approaches to solve a complex problem (Cleaves, 1987), including when synthesizing literature for research purpose (Hamp-Lyons & Mathias, 1994; Petticrew & Roberts, 2006). Petticrew & Roberts (2006) stated that literature reviews tend to outline "highly unrepresentative samples of studies in an unsystematic and uncritical fashion" (p.5), which usually caused by the author's leniency to favors information and studies that coherent with the author's beliefs and experiences (Cleaves, 1987). There are six type of possible biases in this literature review, which are anchoring, availability, representativeness, internal coherence, selection, and information biases. Table 2-1 presents the definition of these biases based on Cleaves (1987).

Following Cleaves (1987)'s suggestions, three behavioral methods were employed to lower these biases, which were focusing, decomposition, and logic challenge. Focusing means "structuring both the task and the interviewing environment so that specific biases are identified and corrected as they become symptomatic" (Cleaves, 1987, p.164). The results of this approach are the purpose and objectives of this chapter. The decomposition means breaking down relevant concepts into sub-concepts and their relations to make it more manageable when identifying and synthesizing

relevant literature (Cleaves, 1987). The concept map presented in Figure 2-1 is the result of employing this corrective method. Last, the logic challenge means exhorting the researcher to provide a justifiable reason for including or excluding some concepts or literature (Cleaves, 1987). The researcher employed this method by discussing the study and its justification with peers (i.e., other graduate or engineering education students) and experts (i.e., engineering education professors or a librarian).

Table 2-1.

Possible Biases in this Literature Review

| Bias | Description |
|---|---|
| Anchoring | A tendency to start a discussion from the most natural starting point according to the author's perspective. |
| Availability | A tendency to treat available and accessible information as the truth, which also means if the author could not find the information, then it does not exist. |
| Representativeness | A tendency to assess an event or risk's probabilities based on its resemblance to the author's experiences, rather than using statistical means. |
| Internal coherence | A tendency to favor information that is consistent with the author's beliefs. |
| Selection | A tendency to limit the information based on what the researcher has experienced or expects to occur. |
| Information | A tendency to give more weight to concrete information which consistent with the researcher's beliefs. |

Six scholarly databases were used to find relevant academic publications, which are EBSCO, Science Direct, ACM, IEEE Xplore, ERIC, and Google Scholar. The goal of a literature search is to identify original publications, which is the documents "that was written by the individuals who actually conducted the research study or who formulated the theory or opinions that are described in the document" (Gall, Gall, & Borg, 2007,

p.98). There are three central topics, which are the computer science, computer science education, and self-regulated learning. In finding the relevant publications, the researcher used these keywords: self-regulated learning, metacognition, and self-regulation, task interpretation, task value, task demand, cognitive strategies, computer science education, computer science, programming design, and programming. Further, the researcher also used the combination of above keywords for narrowing the search results. Last, the researcher also explored publications that cited the selected literature using the "cited-in" feature in Google Scholar.

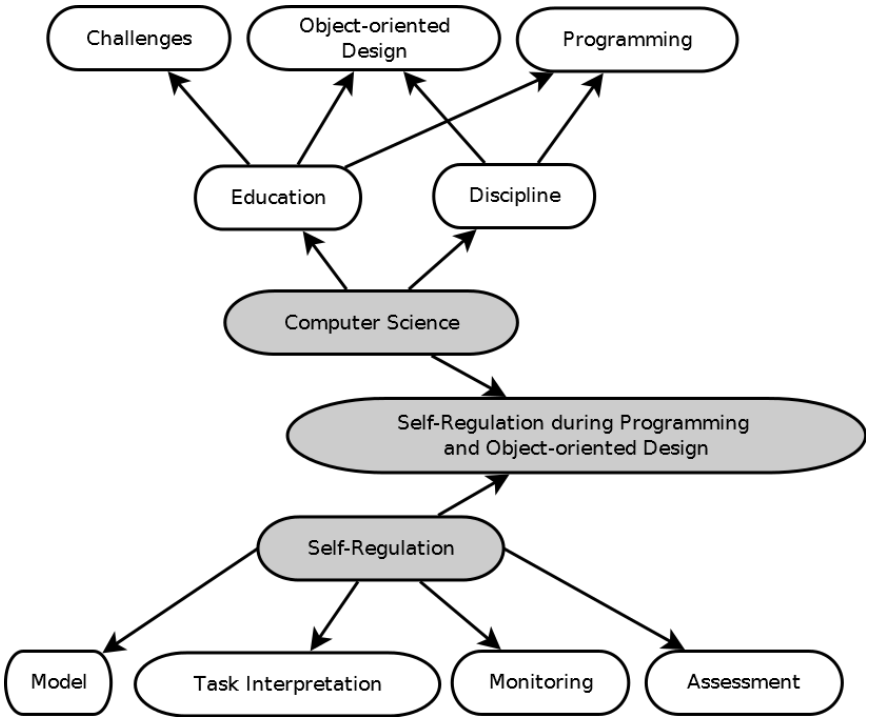



*Figure 2-1. Relevant concepts in this literature review.*

## Computer Science Education

This section discusses computer science as a discipline, computer science education (CSE), and the programming and object-oriented design. Being aware of CS as

a discipline is a major step towards understanding computer science education and the complexity of a programming endeavor (Gal-Ezer & Harel, 1998).

**Computer Science**

Computer science is a discipline which systematically studies "algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application" (Denning et al., 1989, p.12). Since it was born in the early 1940s, this discipline affects and gets affected by the rapid ever-changing technologies. This discipline encourages the development of innovative technologies, and in return, these technologies contribute to the new body of knowledge in CS. Nevertheless, its core concepts remain intact, which is the integration of mathematics, science, and engineering applied knowledge (Denning et al., 1989). Computer scientists use the theory of mathematics to develop notations and conceptual frameworks to represent virtual objects' behaviors and the relationships among them (Denning, 2003). They use science to explore system and architecture models and test whether the models could accurately predict the new behaviors (Denning, 2003). Computer scientists use engineering knowledge to develop "computer systems that support work in given organizations or application domains" (Denning, 2003, p.409).

There are numerous existing and ongoing debates about computer science as a discipline (Clark, 2003). One of the discussion topics is regarding computing principles, which is also commonly known as computational thinking. Wing, (2008) defines computational thinking as "an approach to solving problems, designing systems and understanding human behavior that draws on concepts fundamental to computing"

(p.3717). According to Grover & Pea (2013), most academicians agreed on nine computing principles. Table 2-2 presents the definition of each computing principle. The nine computing principles are about ideas and conceptualization, not programming and artifacts (Wing, 2006). The discipline of CS is not only concerned with human-made information processes but also their cognitive enterprise (Denning, 2003).

*Table 2-2.*

The Nine Computing Principles

| Principle | Definition |
|---|---|
| abstraction and pattern generation | Identifying, populating, and organizing characteristics from an entity into a set of essential characteristics (TechTarget, n.d.; Wing, 2008). |
| systematic processing of information | A step-by-step agent-dependent instruction for processing a set of inputs into desired unambiguous output, which is also known as algorithm (Denning, 2003; Wing, 2008). |
| symbol systems and representations | Develop a model to store and express the characteristics and behaviors of an entity in an efficient way (Denning, 2003). |
| algorithmic notion of flow control | No precise definition found. |
| structured problem decomposition | Subdividing a computational problem into a simpler, more manageable sub-problems (Lee, 2014) |
| iterative, recursive, and parallel thinking | Identifying, populating, and organizing a set of behaviors that can repeatedly be performed or at the same time (Computer Hope, n.d.). |
| conditional logic | Identify a set of criteria to allow or disregard the execution of an instruction set (Computer Hope, n.d.). |
| efficiency and performance constraint | Identifying potential efficiency and performance issues, and developing a method to enhance them (Denning et al., 1989) |
| debugging and systematic error detection | Evaluate and improve the program's accuracy, consistency, performance, and efficiency under various conditions (Denning, 2004; Denning & Freeman, 2009). |

It is clear that the digital computer and computer programming play a significant role in this discipline (Denning, 2003). However, it is inappropriate to equate CS with

programming (Denning et al., 1989). The Joint Task Force on Computing Curricula (2013) in their 2013 CS curriculum guideline for undergraduate program identify 18 bodies of knowledge of computer science, where some of them do not solely focus on programming, for example, Discrete Structures, Human-Computer Interaction, Operating Systems, and Social Issues and Professional Practice.

As an academic discipline, computer science is a hard and applied discipline (Clark, 2003). It is a hard discipline because CS has a body of knowledge that all computer scientists subscribe to, which is the 18 bodies of knowledge. The CS is an applied discipline because it is "pragmatic and concerned with the creation of products and techniques" (Clark, 2003, p.75). The computer scientists always find a way to offer innovations for automating routine works and supporting the professionals in various domains (Denning, 2004; Denning et al., 1989). It is important to note that academy and industry do not necessarily have the same view about CS as a discipline (Clark, 2003). In this document, the researcher only focused on the academic perspective of CS.

**Computer Science Education**

In this computing-based era, CS skills are as fundamental as reading, writing, and arithmetic (Miller et al., 2013). It is important to note that computer science skills do not refer to the ability to use a computer and its applications (Gal-Ezer & Harel, 1998), such as a document processor, a spreadsheet developer, and an Internet browser; CS skills and computer literacy are not the same. Computer science skills refer to the ability to use the nine computing principles (Grover & Pea, 2013; Wing, 2006, 2008). Consequently, the ultimate goal of CSE is enabling learners to apply these principles to any problems

(Senske, 2011) by elucidating the relationship between computer applications and computer systems (i.e., hardware and operating systems) (Denning, 2003).

It is vital for CS educators to understand the nature of CS as a discipline and its relationship with other disciplines (Gal-Ezer & Harel, 1998), and CS-related instructional arts (Guzdial, 2008). They need to know extensive CS knowledge and skills, and have the ability to "convey this knowledge to others correctly and reliably, to teach the said skills, to provide perspective, and to infuse students with interest, curiosity, and enthusiasm" (Gal-Ezer & Harel, 1998, p.77). They must train CS professionals who are skilled, responsible, and exercise the ethics and standard practices set by the professional societies, such as the Association for Computing Machinery (ACM), and Institute of Electrical and Electronics Engineers (IEEE) (Denning, 2001, 2003).

Cross-disciplinary research is not foreign in CSE, especially for assessing students and instructors' perspective to enhance teaching and learning methods (Berglund, Daniels, & Pears, 2006; Diethelm, Hubwieser, & Klaus, 2012). In this study, the researcher only focused on the students' perspective and cognitive behavior related to programming. The role of programming is important in the CSE. Most people agree that knowing how to program is essential for studying CS concepts and principles (Gal-Ezer & Harel, 1998; Lye & Koh, 2014; Wing, 2006, 2008). Studies have found that students' first experience with computer programming in college influences their persistence in this discipline (Beaubouef & Mason, 2005; Kinnunen & Malmi, 2006; Kori et al., 2015). Numerous CS institutions reported a dropout rate of 30% to 50% (Beaubouef & Mason, 2005; Howles, 2007; Kori et al., 2015), including USU CS department (Office of

Analysis, Assessment, and Accreditation, Utah State University, 2016). Studies found that one of the major dropout reasons is the immense challenges in learning computer programming during students' first year (Anderson & Skwarecki, 1989; Guzdial et al., 2015; Howles, 2007; Kori et al., 2015). Leiviskä & Siponen (2013) believe that teaching self-regulation skills to students as early as possible might tackle this problem.

According to Gal-Ezer & Harel (1998), some programming concepts are hard to teach to and be absorbed by the students, such as control structure (i.e., conditionals logic, repetitions, and recursion) and the idea that a program is rigid "yet is supposed to deal with many different inputs of varying sizes" (p.83). Unfortunately, many first-year CS students enter the program due to their interest in using computer applications and playing games, which has little use in their study (e.g., programming) (Clark, 2003; Howles, 2007). The limited experiences with programming make students feel an excessive burden to understand and applied various CS concepts correctly, which then may drive them to cheat and plagiarize (Denning, 2004; Howles, 2007). Naturally, many CS educators tried to tackle this problem, either by enhancing instructional practices (e.g., through active learning) or developing computer-based instructional tools (Adams, 2007; Barak, Harward, Kocur, & Lerman, 2007; Briggs, 2005; Carnegie Mellon University, n.d.; Gonzalez, 2006; Krauss, 2008; MIT Media Lab, n.d.; Resnick et al., 2009; Ruthmann, Heines, Greher, Laidler, & Saulters, 2010; Whittington, 2004; L. Williams, Wiebe, Yang, Ferzli, & Miller, 2010).

Brennan & Resnick (2012) organizes the challenges in learning to program into three categories, which are concept, practice, and perspective. Understanding various CS

concepts become harder if the learners do not have an effective cognitive model of a computer (Ben-Ari, 1998). Without it, learners tend to construct their own rules, which are not part of the programming language (Lischner, 2001), for example assuming the variable initial assignment as a constant. Learners' misunderstanding usually worsens by their attempt to memorize, rather than to put more effort comprehending, the concepts (Whittington, 2004). Regarding the computing practice, Lischner (2001) reported that many first-year students struggle to study outside of the classroom during their transition from high school to college, which suggests many first-year students do not spend adequate time learning to program independently. On the other hand, intensive interaction with a computer discourages the students who prefer social or reflective learning style (Ben-Ari, 1998). Related to perspective, with the emergence of various computer-assisted educational tools, some students might think their competency in using these tools is reflecting their programming expertise, which is not the case (Wing, 2008).

**Programming and Object-oriented Design**

> "A person does not really understand something until he can teach it to a computer [i.e., write a program]" – Knuth

A computer program is "an abstract symbol manipulator which can be turned into a concrete one by supplying a computer to it" (Dijkstra, 1989, p.1401). Computer programming is a process of developing computer programs using any programming language and tools (Lee, 2014). Therefore, a programming activity concerns with the "interplay between mechanized and human symbol manipulation" (Dijkstra, 1989, p.1401). Programming involves translating a statement or way of thinking in the natural

language into a corresponding entity in another language (Renumol, Janakiram, & Jayaprakash, 2010). In other words, programming is a problem-solving activity. There are various programming languages at each level (i.e., machine, intermediate, and higher-order levels), each has its unique strengths and limitations (Denning, 2003). For the higher-order level, for example, Java™ and C/C++ programming languages are available to use. Out of the three levels, machine programming language is the most difficult to understand (Eden, 2007). Consequently, computer scientists should develop their skill to select the best programming language for solving a specific problem since it may affect the program's performance (The Joint Task Force on Computing Curricula, 2013). Further, a computer scientist must pay attention to the algorithm's correctness and efficiency (Gal-Ezer & Harel, 1998). Having an ability to write a computer program does not necessarily make someone a computer scientist or a programmer (Clark, 2003).

Programming languages help programmers to organize their code, so they can focus on solving the problem (Lee, 2014). This organizational framework is also known as the programming paradigm (Dictionary.com, n.d.; Lee, 2014). There are various programming paradigms, and some of them share common concepts and ways of thinking (Toal, n.d.). Two of the commonly used paradigms are imperative and object-oriented paradigms. In the imperative paradigm, programmers need to explicitly describe the required steps (i.e., algorithm) that the computer needs to follow to get the desired solution (Computer Hope, n.d.). This paradigm allows programmers to decompose a complex problem into smaller sub-problems and express the solution of each sub-problem in a subprogram or procedure (Lee, 2014). The object-oriented programming

(OOP) paradigm is an extension of the imperative paradigm, which allows programmers to organize their code into classes of objects and procedures (Lee, 2014). The programmers need to consider the relationship and accessibility among objects. The program structure, mechanics, data representation, and algorithm are equally important (Denning, 2003). The solution for the third and fifth problems in Appendix M is an example of an OOP and imperative programming respectively.

Despite the level of complexity and structure differences, most programming problems have multiple solutions, for example, the fourth problem in Appendix L and Appendix M. To solve such problem, the learners need to understand the contexts surrounding the problem, identify goals and constraints, produce artifacts, and restructure the problem. The programmer must consider the solution's simplicity, accuracy, efficiency, usability, software and hardware reliability, robustness, evolvability (i.e., easy to modify and scale), and security (Clark, 2003; Denning, 2003, 2004). In other words, programming is a design endeavor (Jonassen, 2000, 2010).

## Task Interpretation and Monitoring in Self-Regulated Learning

All effective learner deliberately utilizes judgmental and adaptive SRL strategies (Butler & Winne, 1995). Consequently, students who are capable of self-regulating themselves tend to achieve academic success (Butler & Cartier, 2005; Coutinho, 2007) and produce a quality design product (Lawanto, Butler, Cartier, Santoso, Goodridge, et al., 2013). Furthermore, SRL has a positive influence towards the problem-solving endeavor (Lawanto, 2010; Lawanto & Johnson, 2009; Pintrich, 2002). Inadequate self-regulation engagements may result in a fruitless problem-solving attempt (Schoenfeld,

1983). Therefore, understanding students' self-regulated learning is an important research endeavor. In this section, SRL framework, task interpretation strategies, monitoring strategies, and SRL assessment methods are discussed.

**Self-Regulated Learning**

Self-regulated learning is a situated and iterative goal-directed learning process that involves complex and dynamic activities (Butler & Cartier, 2005; Butler et al., 2015; Butler & Winne, 1995). It is important to understand the complex process of learning to appreciate SRL as a learning process framework. The Oxford University Press (2008) dictionary defines learning as an endeavor to gain skills or knowledge in a specific activity or subject. The proponents of behaviorism, cognitivism, and constructivism view learning differently (Ackermann, 1996; Bransford, Brown, & Cocking, 1999; Bruner, 1966; Ertmer & Newby, 2013; Mayer, 1996; Ormrod, 2007; Skinner, 1988). Learning is also affected by culture (Cobb, 1994), emotions (Artino & Stephens, 2007; Forgas, 2000; Lenox, Woratschek, & Davis, 2008; Peixoto, Mata, Monteiro, Sanches, & Pekrun, 2015; Pekrun & Perry, 2014; Sinatra, Broughton, & Lombardi, 2014), and motivations. (Pintrich, 2003; Ryan & Deci, 2000; Vansteenkiste, Lens, Elliot, Soenens, & Mouratidis, 2014; Wigfield & Eccles, 2000). The SRL tries to capture these influencing factors in a single framework (Butler & Cartier, 2005; Zimmerman, Heart, & Mellins, 1989).

This study defines learning as recursive cognitive processes of understanding stimulus (e.g., contents, situations, or problems) to select the most suitable responses, that is affected by one's motivation, belief, and past experiences. This study views learners as

"goal-directed agents who actively seek information" (Bransford et al., 1999, p. 10) and construct their own knowledge (i.e., facts, ideas, and beliefs) (Ben-Ari, 1998).

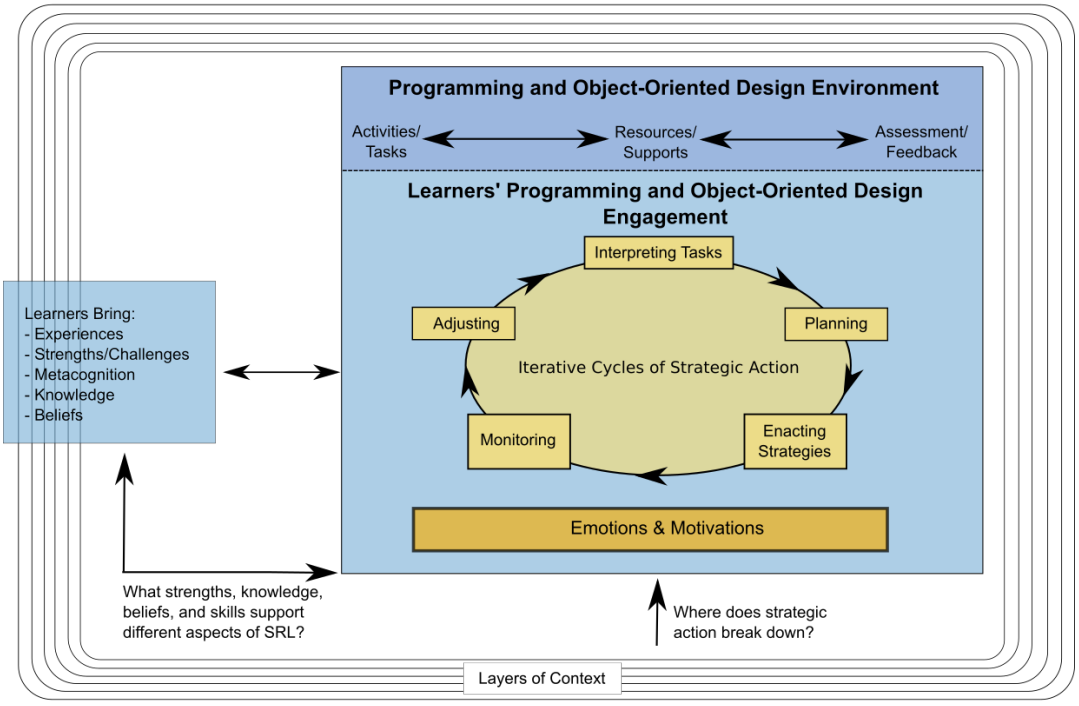



*Figure 2-2. Butler and Cartier's self-regulated learning model.*

There are at least five SRL models that have been introduced since 1996 by researchers, such as Zimmerman, Winne, Hadwin, Pintrich, Butler, and Cartier (Santoso, 2013). This study uses Butler & Cartier's model (BCM) for two reasons. First, BCM emphasizes the importance of contexts (i.e., facts and conditions) surrounding the self-regulation activities (Butler & Winne, 1995). The emphasis on contexts makes BCM applicable in any learning situation, such as medical and reading (Brydges & Butler, 2012; Butler & Cartier, 2004a, 2005; Butler, Cartier, Schnellert, Gagnon, & Giammarino, 2011; Cartier & Butler, 2004). Second, the BCM has been used to frame students' self-regulation while engaged in learning to program using an interactive learning tool

(Santoso, 2013), and in engineering design process (Febrian, Lawanto, & Cromwell, 2015; Lawanto, 2010; Lawanto, Butler, Cartier, Santoso, Goodridge, et al., 2013; Lawanto, Butler, Cartier, Santoso, & Goodridge, 2013).

*Table 2-3.*

Definition of All Strategic Actions in Butler and Cartier's SRL Model

| Strategic Action | Definition |
|---|---|
| Task interpretation (TI) | Students' understanding about relationships between task characteristics and associated processing demand (Butler, 1998). |
| Planning strategies (PS) | Selecting appropriate cognitive and metacognitive strategies for completing any tasks (Butler & Cartier, 2005). |
| Enacting strategies (ES) | Students' cognitive activities employed as they engage in their work executing the design tasks, as planned, monitored, and adjusted through metacognitive activity (Lawanto, Butler, Cartier, Santoso, Goodridge, et al., 2013). |
| Monitoring (M) | Students' activities of self-monitor progress, goals, plans, or strategies (adjusting approaches to learning) (Butler & Cartier, 2005). |
| Adjusting (A) | Students' activities of adjusting goals, plans, or strategies based on self-perceptions of progress or feedback (Butler & Cartier, 2005). This activity is always precedes by monitoring. |

As illustrated in Figure 2-2, the BCM describes SRL as the interaction between the programming and object-oriented design environment, the learners, and learner's engagement with the environment. In this study, the learning environment comprises of programming and object-oriented design tasks, available resources, available supports, assessment mechanisms, and external feedbacks (e.g., from the instructors or peers). The learners refer to their experiences, strengths, challenges, metacognition, knowledge, and beliefs. The learners' engagement with the environment involves their iterative cycle of strategic action (or a self-regulating process), emotions, and motivations. The self-regulating process encompasses task interpretation, planning, enacting strategies,

monitoring, and adjustment activities; see Table 2-3 for definition. These five strategic actions are dynamically interacting with each other in each learning episode. This study focused on students' task interpretation and monitoring strategies.

**Task Interpretation and Monitoring**

Task interpretation refers to students' understanding of the relationship between the task and required cognitive processes to complete it (Butler, 1998). It is the "critical first step in SRL" (Butler & Cartier, 2005, p.3) because it determines students' selection of goals, objectives, criteria for success, and required cognitive strategies. Butler & Cartier (2004b) argues that students' metacognitive knowledge about the task, including the typical task purpose, structure, and problem-solving approach, influences the quality of their task interpretation. According to Hadwin (2006), task interpretation includes socio-contextual, explicit, and implicit aspects; see Figure 2-3 for the model. The socio-contextual aspect refers to learners' awareness about the discipline-related knowledge, values, skills, and expertise (Hadwin et al., 2009). The socio-contextual awareness guides learners to select effective domain-specific strategies and be experts in their field (Butler & Winne, 1995; Hadwin et al., 2009). The explicit aspect of task interpretation refers to the "information that is overtly presented in task descriptions and discussions" (p.2) which includes the task goal(s), requirements, constraints, and instructions or standards to be followed (Hadwin et al., 2009). The implicit aspect of task interpretation refers to the "information students might be expected to extrapolate beyond the assignment description" (p.2) which includes relevant concepts, knowledge, and cognitive processes (Hadwin et al., 2009). Since understanding a task is the first step of a self-regulation

activity, learners' misinterpretation in one of the task interpretation aspect might inspire them to select and employ inappropriate strategies for completing the task (Butler, 1995).

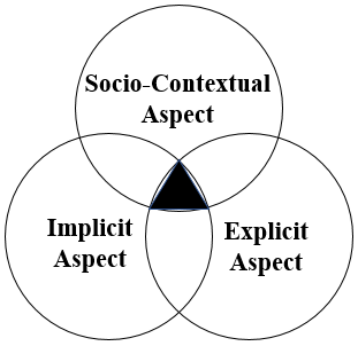



*Figure 2-3. Hadwin's task interpretation model.*

Rivera-Reyes (2015) reported that students have a better task understanding of laboratory activities after they had completed the task. This finding suggests that throughout their engagement, students monitor and update their understanding of the given task. Monitoring activity refers to students' self-assessment of their self-regulating process and progress towards achieving the goals (Butler & Cartier, 2005). Students who do not have relevant knowledge and skills on the task at hand will not be able to accurately and efficiently self-monitor their thought process (Isomöttönen & Tirronen, 2013). When students perceive an obstacle during their learning endeavor (e.g., missing information or lengthy process), they will self-evaluate their progress and reassess their success probability if they continue their effort, adjust their strategies, or both (Carver & Scheier, 1990). It is possible that learners use inappropriate parameters when self-evaluating their learning endeavor, which then drives them to select and employ the wrong strategies (Butler & Winne, 1995). Monitoring failure might also occur when the learners were overwhelmed with the task at hand (Butler & Winne, 1995).

**Assessing Students' Self-Regulation**

Research on students' self-regulation focuses on assessing students' awareness and regulatory responses in an academic environment (Pintrich, 2004; Zimmerman et al., 1989). According to Alexander et al., (2009), in any knowledge acquisition efforts, learners always consider four dimensions of learning. They are (1) the subject to learn; (2) the best place to learn about the subject; (3) the people who can help the learners mastering the subject; and (4) the most appropriate time to learn about the subject. Therefore, understanding the contexts surrounding a learning endeavor is essential.

Self-regulation is dynamic, multi-directional, and complex in nature (Butler et al., 2011). It might occur at anywhere and anytime (Alexander et al., 2009). Therefore, it is crucial to design a study that could capture students' knowledge development and cognitive strategies in each learning episode (Butler & Cartier, 2005; Winne & Perry, 2000) and utilize multiple assessment tools (Dinsmore, Alexander, & Loughlin, 2008). The common types of SRL assessment tools are a self-report survey, journal, observation, thinking aloud, and interview (Dinsmore et al., 2008). Butler & Cartier (2005) advise that although self-report instruments provide insights into students' learning engagement, they are not the best methods for assessing learners' actual behaviors. Related to the thinking aloud method, Jones & Idol (2013) noticed that learners might have a challenging time verbalizing their thought process due to their inability accessing relevant information, the lack of knowledge, and lack of awareness of their thinking complexity. It is also possible that learners have mastered the required skills to solve the problem which prevents them

from communicating their thought process verbally (Johnson, 2008). Additionally, self-explanation might help the learners to self-regulate themselves better (Chi et al., 1994).

## Self-Regulation during Programming and Object-Oriented Design

The majority of CS students are visual, sequential, sensing, and reflective learners (Alharbi, Henskens, & Hannaford, 2012). They like to utilize visual representations, acquire knowledge in a linear fashion, deal with facts and details, and monitor their learning progress periodically (Felder & Soloman, n.d.). Students who have high intrinsic motivations and task value (i.e., an appreciation towards the task relevancy) are more likely to use more SRL strategies and performed better in programming (Bergin et al., 2005). Additionally, Kumar et al. (2005) reported that students' SRL engagement positively influence their programming performance. Furthermore, students who employ discipline-specific SRL strategies are more successful in programming compared to their counterparts (Falkner et al., 2014).

Computer scientists engage in various strategies when developing, understanding, and debugging a program (Shaft, 1995). Havenga (2015) reported that students use the nouns and verbs in the task description as cues to understand the problem. Falkner et al., (2014) reported that students used various computing principles during a programming venture, and they believe that the structured problem decomposition is a critical CS skill but hard to master. Interestingly, some students are incapable of aligning their problem-solving goals with the assessment criteria (Falkner et al., 2014). This finding suggests that students were unable to employ various task interpretation strategies accurately during the programming endeavor.

In object-oriented programming, Havenga (2015) reported that students tend to have “fragmented knowledge and misconceptions of the object-oriented approach” (p.142) and insufficient implementation skills. Interestingly, they find that instead of focusing on acquiring the necessary knowledge first, students tend to continue engaging in programming activity and get frustrated. This report suggests that students were unable to utilize self-regulation skills during the object-oriented programming process fully.

Although CSE research is not uncommon (Berglund et al., 2006), the number of literature on CS students’ self-regulation while engaged in programming is limited.

## Summary

Although the demand for CS Professional is increasing (Hambrusch et al., 2009; Lacey & Wright, 2009), a large number of first-year students are dropping out due to the immense challenges in learning programming (Anderson & Skwarecki, 1989; Guzdial et al., 2015; Howles, 2007; Kori et al., 2015). Most of these challenges are related to CS concepts, practices, and perspectives (Brennan & Resnick, 2012). Exposing students with various self-regulation skills could help ease their learning process (Leiviskä & Siponen, 2013) and improve their programming performance (Bergin et al., 2005; Kumar et al., 2005). Falkner et al., (2014) reported that CS students are unable to align their problem-solving goals with the assessment criteria, which suggests inaccurate task interpretation efforts. Fortunately, students’ task understanding evolved throughout the learning endeavor (Rivera-Reyes, 2015). In other words, students monitor their task understanding and approach throughout the learning enterprise (Isomöttönen & Tirronen, 2013).

# CHAPTER III

# PILOT STUDY

## Introduction

"Do not take the risk. Pilot test first." - De Vaus (2013, p.48).

The term pilot study means a "small scale version, or trial run, done in preparation for the major study" (Polit, Beck, & Hungler, 2001, p.467), which is aimed to "answer a methodological question(s) and to guide the development of the research plan" (Prescott & Soeken, 1989, p.60). Although the pilot study is highly encouraged in quantitative research (De Vaus, 2013), it is also beneficial for qualitative research (Kim, 2011). A pilot study can unravel potential problems in the research design, so it can increase the chance to make the primary study successful (van Teijlingen & Hundley, 1998).

The purpose of this pilot study is to train the researcher in as many elements of the research processes as possible. Specifically, the objectives of this pilot study are to develop and assess: (1) the success rate of the proposed recruitment approach; (2) issues of the proposed qualitative instrument; (3) the appropriateness of the data collection protocol (4) issues of the data analysis method; and (5) the suitability and applicability of the member checking approach.

The researcher utilized the 2016 Research Experience for Undergraduates (REU) program funded by the National Science Foundation (NSF) in the Department of Engineering Education at Utah State University (USU) to conduct the pilot study. Two REU students were assigned to work on this project under Dr. Lawanto's and the researcher's supervision. In this chapter, the researcher describes the 2016 REU program,

and then the approach and lessons learned regarding the participant recruitment method, qualitative instrument, data collection method, data analysis method, and member checking method. At the end, a summary of this chapter is provided.

## The 2016 Research Experience for Undergraduates

This REU site program is sponsored by the National Science Foundation to expose undergraduate students from all over the U.S. to engineering education research during the summer (Engineering Education Department Utah State University, 2016). Interested undergraduate students were expected to fill out an application form. In 2016, eight students were selected from 49 applicants to work in four different engineering education research projects, and two students were assigned to work on a specific project. Since most students did not have prior experience in engineering education research, the primary supervisors' role was providing mentorship to help them navigate through the research process successfully.

*Table 3-1.*

Summary of the Participants' Demographics

| Category | DanielO | Depend | George |
|---|---|---|---|
| Gender | Male | Male | Male |
| Age | 19 | 23 | 36 |
| Ethnic | Hispanic | Asian-Pacific Islander | Caucasian |
| Academic Level | Sophomore | Senior | Sophomore |
| GPA | 3.36 | 3.61 | 2.82 |
| Introduction to CS Grade | A- | A | A |
| Programming hours | 300 | 400 | 100 |

The goal of this REU project was to describe computer science (CS) students' self-regulation while engaged in programming. This ten-week research project was a qualitative case study that involves working in participant recruitment, data collection, data transcription, SRL coding, strategies coding, member checking, and reporting; Appendix B presents the research schedule. The research participants were three undergraduate CS students at USU, and their demographics were presented in Table 3-1. Before the data collection, each participant signed the REU IRB consent and selected alias to protect their identity. At the end of the project, each participant received a personalized self-regulation report and a $25 Amazon gift card.

## Participant Recruitment

The participant recruitment method in this study was convenient and purposeful because all participants were from USU CS department and not all of them could become research participants. To be recruited, the candidates had to have basic programming knowledge, which proven by completing the Introduction to Computer Science course with C- or better, and be willing to dedicate three hours for participating in various research activities (i.e., data collection and member checking).

There were only two courses offered by the CS department during summer 2016, the Introduction to CS and internship courses. Unfortunately, students who enrolled in both courses were not suitable research participants. The Introduction to CS course students were freshmen who did not know how to code correctly, and the internship course students were expected to come to the office during the working hours. Therefore, the best way to contact the potential participants was using the CS department's

broadcasting email system. The procedure to use this system was straightforward. The researcher only needed to send the recruitment information and asked the CS department officer to forward it to all CS students. The recruitment publication contained the project description, contact information, compensation, and participation requirements (see Appendix C). After three days, some students asked about course requirements. There were nine students applied, and the first four suitable applicants were selected. Then, all participants were asked to fill out a demographics survey (see Appendix D). Unfortunately, only three participants showed up during the data collection.

**Lesson Learned**

The email recruitment method was an effective approach to recruit USU CS students. Therefore, it must be utilized for the dissertation study. The recruitment publication must be improved by adding course and knowledge requirements.

## The Qualitative Instrument

The qualitative research instrument consists of five programming questions. The researcher selected and modified five programming problems from available online and offline resources, which are Coding Bat (http://codingbat.com/), Universitas YARSI, and the Head First Design Pattern book by Freeman, Bates, Sierra, & Robson (2004). Coding Bat is an online programming practice environment for Java™ and Python programming languages. This online application was designed and developed by Nick Parlante, a CS teaching faculty at Stanford, as an instructional tool for homework, self-study practice resources, lab exercises, and live lecture examples (Parlante, n.d.). Three problems from Coding Bat were selected and reformatted for the paper-and-pencil problem-solving

approach. Herika Hayurani provided sixteen programming problems. She is a faculty member in the Information Technology College at Universitas YARSI who specializes in delivering programming-related courses. One question, the last standing man, was selected because it allows computer scientists to provide multiple solutions using the imperative or object-oriented programming paradigms. One problem was developed based on the Head First Design Pattern book to enable computer scientists exhibiting their object-oriented design skills.

*Table 3-2.*

Major Changes made in the Qualitative Instrument

| No. | Problem Title | Major Changes Made |
|---|---|---|
| 1 | Locating the Errors | • Changed the title numbering format.<br>• Changed the title from "Awareness of Trivia" to "Locating the Errors."<br>• Changed the term "logic errors" to "errors."<br>• Added an introduction story. |
| 2 | Outputs Prediction | • Changed the title numbering format.<br>• Decreased the numbers of test case from seven to four.<br>• Added an introduction story. |
| 3 | Monopoly in the Middle-Ages | • Changed the title numbering format.<br>• Removed the last problem constraint because it can be inferred from the introduction story. |
| 4 | Algorithm Generation | • Changed the title numbering format.<br>• Added an introduction story. |
| 5 | The Last Standing Man | • Changed the title numbering format.<br>• Clarified the problem algorithm. |

The programming problems were then tested to two other REU students and three research participants; all were video and audio recorded. All testers agreed that the problems were challenging and intriguing. We observed that some testers experienced difficulty when solving the third (i.e., Monopoly in the Middle-Ages) and fifth (i.e., The

Last Standing Man) problems, and another tester commented on the unusual problem-numbering mechanism. The tester who was unable to answer the fifth question gave up after fifteen minutes and explained that he usually works on a challenging problem for few days to give himself a chance to see the problem from a different point of view. The pilot testing revealed that the qualitative instrument suffered from unbalanced problem length, clarity, grammar, and numbering issues. Revisions were conducted to address these issues, which are summarized in Table 3-2. The final qualitative instrument is available in Appendix L.

Some testers' difficulties in solving the third and fifth problems encouraged the assessment of problems' characteristics and difficulty levels. The problem characteristics refer to the problem structure, complexity, and required knowledge and cognitive skills (based on the Bloom's Taxonomy) to answer it. When assessing the problem characteristics, Jonassen (2000) and Gronlund, Gronlund, & Waugh (2013) were used as references. Appendix K presents all problems' characteristics. On the other hand, eleven people were asked to rate the problems' difficulty from 1 to 10, where 1 means a very easy problem and 10 means a very hard problem. The difficulty range was arbitrarily selected. These people were CS professionals, instructors, undergraduate teaching assistants, and undergraduate students. All problems' difficulties are in the range of 2.30 to 6.88 on a 10-point scale. Based on this assessment result (see Appendix K), these problems are suitable for CS senior students and can be solved within two and a half hours. Therefore, the difficulties that experienced by some of the testers were not due to

the problems characteristics and difficulty levels, but might be caused by participants' lack of self-regulation strategies.

**Lesson Learned**

During the pilot test, the qualitative instrument was developed and improved. Justifying the problem suitability is not easy, and requires in-depth analysis of the problems (Carruthers & Stege, 2013), such as assessing the problems' characteristics and difficulty levels. This pilot test showed that the qualitative instrument was suitable for the dissertation study.

## Data Collection

The student investigators collected data from three participants. The data collection process includes providing a brief description of the research project, signing the IRB consent, providing general instruction, demonstrating thinking aloud, helping participants to practice thinking aloud, addressing issues with participants' thinking aloud, and observing participants' problem-solving endeavor while thinking aloud. Each data collection process was expected to finish within two and half hours, and audio- and video-recorded. Appendix E presents the scripts used for describing the research project and demonstrating the thinking aloud method. The participants practiced thinking aloud using the first and second problems. Throughout the data collection, the student investigators used one of the prompts in Table 3-3 to remind the participants to think aloud. They developed these prompts based on literature and videos related to the verbal protocol (see Appendix B for their detailed research activities). We observed these

prompts were effective as non-leading reminders. All participants completed the data collection process in less than two and half hours.

*Table 3-3.*

Thinking Aloud Prompts

| **Prompts** |
|---|
| What are you thinking? |
| Tell me what you are thinking. |
| What is your strategy or plan? |
| Please remember that we need you to say what you are thinking. |
| Why are you doing that? |

During the data collection, the participants were provided with blank papers, a pen, a pencil, two chocolates, a water bottle, and a can of soda. The chocolates and drinks were provided in case they need to lower their anxiety with foods. We noticed that some participants like to use the pen, while others like to use the pencil. Some of them like to make marks on the problems, while others like to keep them intact. Some participants also like to use many papers while thinking.

The data collection is a crucial process in research. A simple technical problem could affect the accurateness and completeness of the research, and it might occur anytime to anyone, before, during, and after the data collection process. During the pilot study, two voice recorders were used as back up, and all collected data was uploaded immediately to the network storages (i.e., research NAS and Box). The voice recorders were useful because it enabled us to triangulate one of the participants' missing statement. The student investigators' negative attitudes, such as seeming uninterested or

sounding condescending towards the participants, could also negatively affect the participants' behaviors.

**Lesson Learned**

This pilot study verified the effectiveness of the developed prompts, and that all questions can be answered in two and half hours. It also demonstrated the importance of maintaining the research equipment regularly, providing options to the participants, having a secondary recording, backup research data to network storages, and being aware of our body languages. Therefore, thinking aloud reminder prompts will be used, and best practices will be exercised in the dissertation study. Additionally, the researcher recognized that other qualitative instruments need to be developed including the problem-space map for tracking, initial task understanding open-ended survey for assessing participants' initial task interpretation, and post-problem-solving interview for assessing the changes in participants' task understanding and their justification.

## Data Analysis

During the data collection, participants' notes, answers, and problem-solving endeavors were collected in the form of papers, video files, and audio files. The video files were transcribed, and then segmented and coded based on the BCM strategic action (see Table 2-3). After that, the student investigators interpreted the purpose each self-regulation activity. It was not an easy task because each student investigator has a different perspective. Additionally, sometimes the transcription could not capture the contexts surrounding the self-regulation activity, which required them to triangulate it

with the recorded videos and collected participants' notes and answers. For example, when solving Monopoly in the Middle-Ages problem, George said:

> "All right, so space… so then the board is going to be a thirty not space but a thirty value array, array of spaces, and space needs to include, so it is going to have a Boolean value for… whether it is owned or not."

The above excerpt could belong to either the task interpretation, planning, or enacting strategy. From the recorded video, it was clear that the George was adding information to Board and Space classes when he said that, which provided the missing context (i.e., adding information) and made enacting strategy as the most accurate code.

**Lesson Learned**

There are two valuable lessons learned. First, it is essential to understand the contexts surrounding a self-regulatory activity by triangulation. Second, a specific data analysis method for the dissertation study needs to be designed. Based on the first lesson learned, it is essential to consult with the recorded video when discussing coding differences in the dissertation study. Also, further transcriptions should incorporate some contexts by describing participants' activities, writing the first letter of related concepts in capital letter, and using a dash ("-") to indicate a quick focus change on participants' cognition. For example:

> All right, *[writing it down]* so Space-so then the Board is going to be a thirty-not Space, but a thirty value Array-Array of Spaces, and Space needs to include-so it is going to have a Boolean value for-whether it is owned or not.

## Member Checking

The purpose of member checking is to verify the credibility and accuracy of the researcher's interpretation from the participants' point of view (Creswell, 2012). In this pilot study, the participants were asked to review and give recommendations to improve the personalized SRL reports (see Appendix F). All participants agreed with their personalized report and suggested to add a brief description of the problems that they solved, a short comparison of their performance to others, and recommendations to improve their problem-solving skills based on research.

### Lesson Learned

Asking the participants to read and comment on the personalized SRL reports is a good approach for assessing their perspective on the research results and interpretation. All provided suggestions will be incorporated into the dissertation study's personalized SRL report.

## Summary

This pilot study was conducted as one of the 2016 REU research projects, in which goal was to describe computer science students' self-regulation while engaged in programming. Two undergraduate student investigators were assigned to this project, and they involved in the data collection, data transcription, SRL segmentation and coding, strategies coding, member checking, and reporting. Three USU computer science students were recruited as research participants. Each participant completed all research activities and received a personalized SRL report and a $25 Amazon gift card.

In relation to the dissertation study, the researcher learned that the email recruitment method was an effective approach to recruit CS students, and the recruitment information must include course and knowledge requirements. After three revisions during the REU project, the qualitative instrument is finalized. The problem-space maps and data analysis method needs to be developed. Last, the personalized report for member checking needs to be enhanced by adding a brief description of the problems, a short comparison of the participant's performance to others, and suggestions to improve the participant's problem-solving skills based on research.

# CHAPTER IV

# RESEARCH DESIGN

## Introduction

This chapter starts by reviewing the research questions which drove the dissertation study. After that, the researcher's positionality in this study is described and then followed by the discussion of the chosen methodology to answer these research questions. The chapter then continues by explicating the institutional review board application, research method, research participants, qualitative instrument, data collection procedure, and data analysis method.

## Research Questions

Educational research on students' self-regulation is necessary because studies found that self-regulated learning (SRL) positively influences students' academic achievement (Butler & Cartier, 2005; Coutinho, 2007) and design quality (Lawanto, Butler, Cartier, Santoso, Goodridge, et al., 2013). Additionally, teaching self-regulation skills as early as possible might increase students' persistence in the computer science (CS) department (Alexander et al., 2009). Student retention is one of the fundamental problems in computer science (Ambrosio et al., 2012; Beaubouef & Mason, 2005; Howles, 2007; Kori et al., 2015) and becomes more crucial since the demand for CS professionals is growing (Lacey & Wright, 2009). Most students drop out between the first and second year due to the immense challenges while learning computer programming (Anderson & Skwarecki, 1989; Beaubouef & Mason, 2005; Guzdial et al., 2015; Howles, 2007; Kori et al., 2015). Discovering such fact is discouraging because

knowing how to program is essential for studying computer science concepts and principles (Gal-Ezer & Harel, 1998). Therefore, understanding students' self-regulation is crucial for helping students to better cope with programming challenges. This research results will inform CS instructors and students' expectation on the nature of programming enterprises and help them to be more aware of their thinking process during the problem-solving endeavor. Three research questions were used to guide this investigation of undergraduate computer science students' explicit and implicit task interpretation, their revision, and monitoring strategies during programming. These questions were:

1. What was the students' initial task interpretation (i.e., the explicit and implicit aspects) of the given problems?
2. How did their original understanding change during the problem-solving endeavor?
3. What were the influencing factors for any revisions of their initial task understanding?

## The Researcher's Positionality

The researcher was a Doctoral student in engineering education with a Bachelor and a Master of Computer Science degrees. While pursuing the those degrees, the researcher participated in various activities, for examples as a teaching assistant for several different courses, an academic student-mentor, an instructor in many workshops, and a team member in various research projects. The researcher also had one and half years of experience as a faculty member in the College of Information Technology. One of the researcher's responsibility was to teach programming courses for first- and last-

year students. These prior knowledge and experiences have equipped the researcher with the necessary skills to conduct this study and shaped the researcher's beliefs that informed this study. This section aims to illuminate those beliefs and their effect on this dissertation research.

**Ontology**

Ontology refers to the nature of reality and its characteristics (Creswell, 2012). In this study, the researcher subscribes to the social constructivism (or interpretivism) and positivism and partially subscribes to behaviorism. In social constructivism, people develop personal meanings of their experience to understand the world they live in (Creswell, 2012). It is the researchers goal to gather and disclose the participants' views of the situation as much as possible, and then interpret the meaning of those views (Creswell, 2012). The researcher also subscribes to postpositivism, which means people's behaviors are logical cause-and-effect actions that can be determined based on existing theories (Creswell, 2012). Last, the researcher partially subscribes to behaviorism, which means that the researcher believes that fully functional humans inherently can become anything that they want (Ertmer & Newby, 2013; Ormrod, 2007)

**Epistemology**

Epistemology addresses the questions of what can be considered as knowledge and how it can be gathered and interpreted (Creswell, 2012). In this study, the participants were the source of knowledge, which include their demographics, experiences, observable actions, thought processes, justifications, perceptions, answers, and notes. Additionally, the researcher's observation memos about the participants were

also considered as a source of knowledge because it captured some aspect of the participants. The methods to gather and interpret the data are discussed in other sections.

### Axiology

Axiology refers to the values that the researcher bring into the study (Creswell, 2012). Some of those values are listed in this subsection, the others are mentioned in various places in this document. First, the researcher believes that fully functional humans inherently have the ability to become anything that they want (Ertmer & Newby, 2013; Ormrod, 2007). In other words, everyone has an equal potential to become a computer scientist. Second, accuracy, reliability, and effectiveness are essential aspects of an algorithm. Third, extensibility and reusability are crucial elements in any object-oriented design. Fourth, an action is influenced by the contexts surrounding that particular action. Fifth, sometimes people use various terminologies to refer to the same object or instance.

## Research Methodology

The purpose of this section is to explicate the justification for selecting the research questions and methodology (Burton, 2002). Between the research questions and approaches, there is a dialog that influences and refines each other, such that the research questions might limit the appropriate research methodologies and vice versa (Case & Light, 2011). There is limited partial knowledge in the literature about CS students' self-regulation and the quantitative instruments to measure it. Bergin, Reilly, & Traynor (2005) used MSLQ (or Motivated Strategies for Learning Questionnaire) for assessing the role of students' self-regulation in programming. However, this instrument is not

suitable for answering the research questions because it cannot assess the task understanding transformation and its justification. Therefore, the qualitative research method was employed. To be more specific, the researcher used the within-site embedded qualitative multiple case study research approach.

**Qualitative Case Study**

The qualitative case study research method is a qualitative approach for exploring a real-life, contemporary bounded system(s) or case(s) over time by collecting multiple detailed and in-depth data (Creswell, 2012). The bounded systems in this study were senior computer science students at USU and their programming endeavor. The case study approach was suitable because this research was an exploratory study. Further, Butler & Cartier (2018) recommends using case study research design to assessing and learning about students' self-regulated learning. Additionally, this method recommends to collect and analyze multiple detailed and in-depth data, which are consistent with Dinsmore, Alexander, & Loughlin (2008)'s suggestion for researching self-regulation.

**The Multiple Cases**

The cases are selected to best understand the issue of interest (Creswell, 2012). In this study, the issue was the CS senior students' task understanding and their revision. Therefore, knowledge must be drawn from them. This study focused on senior CS students because most students need more than two semesters to learn programming (Tew, McCracken, & Guzdial, 2005) and more time is required for mastering the skills to manage time and resources wisely during programming endeavor (Beaubouef & Mason,

2005). Through the course works, the senior students are expected to develop minimum programming and managerial skills for working in the industry.

Four senior students were selected as cases. Unlike grounded theory research, a case study usually involves five or less participants (Creswell, 2012). In selecting the prospective students, Creswell (2012) suggests getting as much diversity as possible. In this study, students were grouped by academic performance (i.e., GPA) and gender, and one student was selected from each group combination. The grouping by academic performance was based on findings that a competent self-regulated student tends to have an excellent academic achievement (Butler & Cartier, 2005; Coutinho, 2007). The grouping by gender was based on findings that during a learning and problem-solving endeavor, male and female students think, perceive, and self-regulate themselves differently (Irani, 2004; Lawanto, Cromwell, & Febrian, 2016; Madigan, Goodfellow, & Stone, 2007; Pivkina, Pontelli, Jensen, & Haebe, 2009).

**Participant Recruitment: Within-Site**

All cases were recruited from the USU CS department. By definition, this study is a within-site multiple case study research (Creswell, 2012). From another perspective, this study used the convenient sampling method because the USU CS students were readily and easily accessible population (Teddlie & Yu, 2007). However, this study was also using the purposeful sampling method because there were selection criteria used to ensure diverse participants (Creswell, 2012; Teddlie & Yu, 2007).

**Multiple Data Points**

Following Dinsmore, Alexander, & Loughlin (2008) and Creswell (2012)'s recommendations, multiple types of data were collected. In this study, the researcher utilized the thinking aloud method, problem-space maps, open-ended survey, and interview to generate the required data for answering the research questions. During the data collection, the participants answered five programming problems while thinking aloud and were audio- and video-recorded. Two types of data were collected from each problem: primary and secondary data. The primary data refers to all data points that can be used to answer the research questions, which include survey responses, problem-solving recorded audios and videos, and interview response. The secondary data refers to all data points that can be used to triangulate and refine the research findings and interpretations, which include the participants' answers to the programming problems, their notes, and the researcher's memos. The method to analyze the primary and secondary data is presented in the data analysis section.

**The Programming Problems.** All five programming problems (see Appendix L) either use the object-oriented or imperative programming paradigm, which are the paradigms of the 2016 top ten programming languages (Cass, 2016). Since most higher educational institutions have a tendency to use one of the popular programming languages as the centerpiece of their introduction to programming language (Denning, 2004), most CS students are familiar with these paradigms. All programming problems were developed and tested during the pilot study (see Chapter III for details). All questions' difficulty was rated by CS professionals, instructors, undergraduate teaching

assistants, and undergraduate students between 2.30 to 6.88 on 10-point scale which could be interpreted as easy to above medium difficulty and can be answered by most senior CS students at USU within three hours (see Chapter III for more information).

**Thinking Aloud Method.** Thinking aloud is a commonly accepted method to assess people's thinking process (Bainbridge & Sanderson, 2005). However, it is not a perfect method. First, thinking aloud could influence the results of this study because it might help the participants to self-regulate themselves better (Chi et al., 1994), and since there is no known approach to overcome it, this becomes the limitation of the study. Second, during the problem-solving endeavor, the participants might process multiple sets of information in a brief moment and forget to report them (Bainbridge & Sanderson, 2005). Third, the participants might not explicitly mention the relevant knowledge and thinking process that they used during problem-solving if not asked explicitly by the problem (Bainbridge & Sanderson, 2005). Fourth, the participants' tacit knowledge and skills might make them fail to report some of their cognitive activities during the programming endeavor accurately (Bainbridge & Sanderson, 2005; Johnson, 2008). Such condition is probable in this study because, throughout their educational experience, CS students might develop some tacit knowledge and skills related to programming. The tacit expertise enables people to execute certain activities automatically and is usually developed through extensive practices (Johnson, 2008). Nevertheless, this method is the only available method of investigation that looks to students' awareness on their thought processes as they engage in various cognitive activities to solve given problems.

**Problem-Space Map.** To handle the second, third, and fourth limitations of the thinking aloud method, Johnson (2008) proposed to utilize a problem-space map, which is a diagram that describes all relevant issues in a problem and their relationships. Problem-space refers to all relevant issues encountered during the process of solving a problem (AlleyDog.com, n.d.). The researcher used the problem-space map to track participants' task understanding prior, and the revision of their task understanding during the problem-solving endeavor.

**Open-Ended Survey.** The survey goal was to assess participants' initial explicit and implicit task interpretation. Consequently, the participants were asked to fill this survey after reading but before solving the problem.

**Interview.** To handle the second, third, and fourth limitations of the thinking aloud method, especially the issues related to design justification, the researcher conducted a semi-structured interview at the end of each problem-solving endeavor. Additionally, this interview served to assess the revision of participants' task interpretation and their justification for those changes.

**Embedded Data Analysis**

In this study, there were two units of analysis in each case, which were designing an object-oriented system (i.e., the third problem) and an algorithm (i.e., the fifth problem). The object-oriented system problem could only be answered using object-oriented programming paradigm. The algorithm problem could be answered using any programming paradigm. In terms of abstraction, the algorithm problem asked the participants to develop a function or a black box with a particular behavior. The object-

oriented problem asked the participants to develop multiple, integrated functions or black boxes. Thus, both problems required the participants to use different concepts and work on a different abstraction level. Additionally, the first, second, and fourth questions were easier problems compared to the third and fifth questions, and might not be able to showcase the participants' self-regulation skill. Since there were two units of analysis, this dissertation research used an embedded multiple case study design (Yin, 2009). The analysis process included organizing, transcribing, coding, and triangulating the findings and interpretations. All will be discussed in the data analysis section.

### Reporting Results

Following Yin (2009) and Creswell (2012)'s recommendation, this study report would include the description of contexts, cases, findings of each analysis unit, and general findings of participants' task interpretation and its revision.

## Research Method

This study employed the within-site embedded qualitative multiple case study research approach. This means that this study recruited participants (i.e., multiple cases) from the researcher's institution (i.e., within-site) (Creswell, 2012), where each case consists of two analysis units (Yin, 2009). The research activities included IRB application, participant recruitment, data collection, preliminary analysis, member checking 1, data analysis, member checking 2, and reporting. All were completed in two semesters. Appendix N presents the research schedule in detail.

## Institutional Review Board Application

The goal of the Institutional Review Board (IRB) is to protect human participants' rights and welfare during the research process (Utah State University Office of Research and Graduate Studies, n.d.). Consequently, it is mandatory for the researcher to complete a human research protection training and acquire IRB's approval prior conducting this dissertation study. The researcher has completed and retook the Collaborative Institutional Training Initiative (CITI) on January 21, 2014, and December 2, 2016, respectively, and received a three-year curriculum completion report at the end of each training. Also, the researcher acquired IRB's approval on August 29, 2017, under the protocol number 8659. The IRB approval letter is available in Appendix O.

During the data collection, a signed letter of consent was collected from each participant to provide a legal binding document between both parties (i.e., the participants and the researcher). Additionally, this study only accepted adults (i.e., at least 18 years old according to UT law) as research participants to ensure the consent legality.

## Research Participants

This section describes the method for recruiting and selecting research participants. Four senior computer science students at USU were recruited for this research, which was an ideal number of participants in a case study (Creswell, 2012). As illustrated in Table 4-1, one participant was selected to represent high- and low-performance male and female students.

*Table 4-1.*

Number of Participants based on Gender and Academic Performance

| | **Gender** | | | |
|---|---|---|---|---|
| **Gender** | Male | | Female | |
| **GPA** | High | Low | High | Low |
| **Number of Participants** | 1 | 1 | 1 | 1 |

There were four criteria to become a research participant in this study. First, the candidate must be USU CS senior students. Second, the candidate must be an adult according to State of Utah's law (i.e., at least 18 years old) to ensure that his or her consent is legal (Institutional Review Board, 2011). Third, the candidate must have at least 2.30 GPA on a 4-point scale, which is a requirement for graduating from the USU CS undergraduate program (Utah State University, n.d.). By enacting this criterion, the researcher tried to ensure that all participants had the required skills to function as future CS professionals. Fourth, the candidate must have completed the Introduction to CS course (CS 1400) with C- or better, which is also a requirement for graduating from the USU CS undergraduate program (Utah State University, n.d.). Each selected candidate received a $40 Amazon gift card and a personalized SRL report at the end of the study.

**Participant Recruitment Method**

The goal of this process was disseminating recruitment publication to all USU CS senior students. Three methods were used to spread the recruitment publication. The first method was an email-dissemination approach. This method has been proven effective during the pilot study (see Chapter III for a detailed discussion). The researcher asked the person in charge of the CS department's broadcasting email system to forward the

recruitment publication to all CS senior students. The second method was by displaying recruitment announcement on the notice boards at Taggart Student Center, Old Main, and Engineering. These buildings were selected because the CS students use these buildings often for dining or classes. The third method was by communicating and recruiting the potential candidates face-to-face. All publication materials included "the name and address of the investigator and/or research facility; the condition under study and/or the purpose of the research; a summary of the criteria that will be used to determine eligibility for the study; a brief list of participation benefits, if any; the time or other commitment required of the participants; and the location of the research and the person or office to contact for further information" (Institutional Review Board, 2011, p.22).

All interested students filled an online application form, which available in Appendix G or at https://usu.co1.qualtrics.com/SE/?SID=SV_1M7vl0kUiumpcZD (this link is not searchable by the search engines). This form was adopted from the pilot study demographic survey (see Appendix D). In the first page, the application asked for the applicant's consent to participate in this study. Additionally, this application form automatically turned down applicants who do not meet the required criteria. See Appendix H for the automatic online application screening flowchart. The criteria for becoming a participant were willingness to participate in this study, being an adult, being a senior CS student at USU, having a minimum GPA of 2.30 on a 4-point scale, and earning a C- or better for the introduction to computer science course (i.e., CS1400). The last two requirements were derived from the USU CS bachelor degree requirement (Utah State University, n.d.).

### Participant Selection Method

A list of applicants was available through the online application form. Due to the automatic exclusion mechanism in the application form, the candidates were adults, senior USU CS students who had GPA between 2.30 to 4.00 and received C- or better for the CS1400 course. The selection method was straightforward. First, all applicants were grouped based on their gender, male or female. Then, the candidates in each cluster were ascendingly sorted based on their GPA. The first and the last applicants in each group were selected as research participants (i.e., the students with highest and lowest GPA). The researcher informed the selected applicants by email, set up the date and time for data collection, and asked them to fill the demographics survey (see Appendix I). The researcher reused most questions in pilot study demographics survey (see Appendix D) to develop the demographics form for this study. If one of the participants decided to discontinue their involvement in this study, the next applicant would be selected from the sorted list.

## Qualitative Instruments

This section discusses all qualitative instruments, which are the programming questions, problem space maps, open-ended survey, and interview.

### Programming Problems

There were five programming problems. All were either related to the imperative or object-oriented programming paradigm. In the first question, Locating the Error, the participants must identify two programming mistakes in a code snippet. In the second question, Output Prediction, the students must predict an algorithm outputs for given

input variations. The research used these two problems to familiarize the participants with the thinking aloud method and the data collection routine. In the third question, the Monopoly in the Middle-Ages, the participants must design a base for a game system using the object-oriented programming paradigm. In the fourth question, Algorithm Generation, the students must implement an algorithm with predetermined behaviors. In the fifth question, the Last Standing Man, the participants must also implement an algorithm with specific behaviors. However, the last question was more complex compared to the previous question. The fourth question contained three issues and three variables and was marked 3.00 out of 10.00 difficulty level. The fifth question contained at least five issues and 4 to 40 variables and was marked 6.56 out of 10.00 difficulty level. Please refer to Qualitative Instrument section in Chapter III and Appendix K for the detailed discussion on the problem difficulty. The third and last questions were the central problems in this study, which means the data analysis would be focused on illuminating the participants' task interpretation and its revision while engaged in these two problems. The fourth question served as a break question, which was to give the participant a time to calm down before answering the last question.

**Problem-Space Maps**

The researcher used problem-space maps to track participants' task understanding prior, and its revision during, the problem-solving endeavor. The problem-space map illustrates all relevant issues or tasks of a problem and their relationships in the form of a diagram. However, the researcher utilized a text-based problem-space map instead of diagram due to the sophisticated nature of design problem-solving process. For example,

the problem-space map of the third question contains 51 tasks and 16 possible creative improvements. Representing 67 possible cognitive activities in a form of a diagram was possible but the chart would be enormous and hard to use compared to in a form of plain texts. Appendix I presents the problem-space maps of all problems. Although these problem-space maps were developed and refined based on the pilot study data, these maps were still incomplete due to large solution variations.

In developing and refining the maps, all pilot study participants' transcribed responses were used. The first step was to code the transcriptions based on the issues (i.e., identifying variables and functions, and determining variable accessibility). This step required the researcher to engage in an open-coding activity. The second step was to group and integrate the identified issues to the maps. The issues grouping was driven by the nine computing principles (see Table 2-2) and BCM's strategic action (see Table 2-3). The last step was to verify the problem-space maps by validating the maps with the transcriptions, in such a way that the maps were capable of capturing all pilot study participants' thought process.

The researcher developed problem-space maps of all problems for two reasons. First, as a means to gain a deeper understanding of, and enhance the problem-space maps. Second, as a means to improve the researcher's sensitivity to, and mental preparation for tracking participants' thought process throughout the data collection session.

**Initial Task Interpretation Survey**

The purpose of this instrument is to assess participants' initial explicit and implicit task interpretation. The explicit task interpretation refers to the "information that

is overtly presented in task descriptions and discussions" (p.2) which includes the task goal(s), requirements, constraints, and instructions or standards to be followed (Hadwin et al., 2009). The implicit aspect of task interpretation refers to the "information [that] students might be expected to extrapolate beyond the assignment description" (p.2) which includes relevant concepts, knowledge, and cognitive processes (Hadwin et al., 2009). Based on these definitions, and Rivera-Reyes (2015)'s and Lawanto, Minichiello, Uziak, & Febrian (2018)'s works, six open-ended questions were developed.

Prior the data collection, the open-ended survey was verified, in such whether the open-ended survey and interview questions could performed their purpose, which were assessing the participants' initial task interpretation and its revision respectively. Two experts were involved, which were a university computer science instructor and an information technologist. They were asked to answer the third or fifth programming problem by following the data collection protocol (see the Data Collection Procedure subsection for the detailed information about this). In short, after reading the problem, they were asked to answer these six questions, the programming question, and then the interview questions. In the end, suggestions for aligning their responses with the researcher's expectations were discussed and incorporated into the questions. Also, one of the open-ended questions was removed, which was "What are the standards that need to be followed to answer this problem?" as suggested by the experts since it was unclear what was the 'standards' in that question referring to. Table 4-2 presents the final open-ended questions for assessing participants' initial explicit and implicit task interpretation.

*Table 4-2.*

Open-Ended Questions for Explicit and Implicit Task Interpretation

| No | Aspect | Question |
|---|---|---|
| 1 | Explicit | What is the primary goal of this problem? |
| 2 | Explicit & Implicit | In relation to the program that you will design, what are the requirements and constraints that you need to consider? |
| 3 | Implicit | What are the programming concepts related to this problem? |
| 4 | Implicit | What are your previous experiences related to this problem? |
| 5 | Implicit | In relation to the program that you will design, what are the steps (e.g., tasks) that you need to take? |

**Post Problem-Solving Interview**

Rivera-Reyes (2015) reported that students have a better task understanding of laboratory activities after they had completed the task. In other words, students' task interpretation transformed during their laboratory engagement. Similarly, CS students' task interpretation might also transform during the programming endeavor. One of the interview session goals is to assess the transformation of participants' task interpretation and their justification for those changes. Additionally, since the participants might process various information in a brief moment during the problem-solving endeavor, they may forget to report those processes (Bainbridge & Sanderson, 2005). Therefore, this interview also serves to capture unreported thought processes, especially that are related to design justifications.

The interview format is semi-structured, which means a set of open-ended questions can be used during the interview with a chance to explore a particular issue further (Whiting, 2008). Table 4-3 presents the interview questions and precondition for asking them. All questions have been verified concurrently with and using the same

verification method for the Initial Task Interpretation Survey (see the previous subsection, Initial Task Interpretation Survey). The purpose of the first, second, and third questions is to assess participants' awareness and perspective about the transformation of their task understanding. The purpose of the fourth question is to confirm whether the participants have an implicit task understanding related to a certain activity or not. If the participant did not have an implicit task understanding, the fifth question would assess the participants' justification for having a new or transformed task interpretation.

*Table 4-3.*

Interview Questions

| No | Condition | Question |
|---|---|---|
| 1 | None. | Do you think your task understanding changes during the problem-solving process? |
| 2 | If participant answered "yes" for question #1. | What are those changes? |
| 3 | Repeat and modify this question based on participant answer for questions #2. | Why did you change [something]? |
| 4 | Repeat and modify this question based on the observation results. | I noticed you did [something]. Did you think about doing that from the beginning? |
| 5 | If the participant answered "no" for question #4. | Why did you do [something]? |

## Data Collection Procedure

The data collection process consists of a brief information session, practice sessions, and problem-solving sessions. It took about three to four hours to complete each data collection process, and all were video- and audio-recorded. This section explicates the environment, thinking aloud method, brief information session, practice session, problem-solving session, and collected data.

**Environment**

Self-regulated learning activities can only happen because students are interacting with the learning environment (Bandura, 1977; Dinsmore et al., 2008). Therefore, knowing the problem-solving environment is essential to understanding students' self-regulation.

In this study, the participants' data were collected in one of the conference rooms of a research-dedicated building. The room shape was similar to a box with two glass doors opposing each other and a picture-window on the side of each door. Inside, there was an oval table in the middle and surrounded by chairs, a big TV monitor mounted on the wall, and a cabinet on one of the corners. The room was well illuminated, and the lights were controlled automatically by a sensor. Unfortunately, due to lack of movement from the researcher and participants, and nonexistent override control, the lights were frequently turned off automatically during the data collection and slightly disturbed the participants' problem-solving endeavor. Since all other available known rooms had a similar power-sensor setting, the researcher opted using this room throughout the data collection because its capabilities to minimize distractions from the passersby. During each data collection session, the participant was seated on a chair that could help him or her ignoring passerby. The researcher only handled one participant in each session, and gave one question at a time. The participants were provided with a pen, a pencil, 12-color highlighter, twenty sheets of white paper, two chocolate bars, two water bottles, and a can of soda. On the table, a recording camera was placed in front of the participants, and a voice recorder was placed on each side of the participants.

**Thinking Aloud**

In this study, the participants must solve five programming problems while thinking aloud. Although, it is a commonly accepted method to assess people's thinking process (Bainbridge & Sanderson, 2005), sometimes the participants might forget to think aloud. In such situation, the researcher used one of the pilot study prompts to remind them (see Table 3-3 for prompts details).

**Brief Information Session**

The goal of the brief information session was to inform the participants about the study purpose, participants' research activities (i.e., participate in the data collection and member checking sessions), data recording, benefits for taking part in the study (i.e., a $40 Amazon gift card and a personalized SRL report), and the thinking aloud method. The researcher used the pilot study method and problem (see Appendix E) to inform the participants about the thinking aloud method. Additionally, the participants were asked to read and sign the IRB consent.

**Practice Session**

The goal of this session was to familiarize the participants with thinking aloud and the data collection routine by completing and reflecting on the first and second programming problems. As illustrated in Figure 4-1, the data collection routine was reading the problem description, answering the initial task understanding survey, solve the programming problem while thinking aloud, and participate in an interview after solving the problem. When answering the initial task interpretation survey, the participants' were prohibited rereading the problem description to avoid them in revising

their task understanding. After finished solving a problem, the researcher answered the participants' questions and addressed their deficiencies if any.

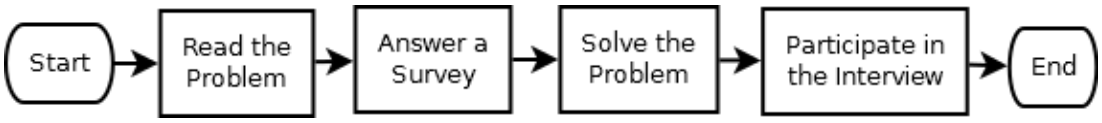



*Figure 4-1. The data collection routine.*

**Problem-Solving Session**

The goal of this session was to collect participants' thought processes while engaged in the third, fourth, and fifth programming problems. During this session, the participants followed the data collection routine in Figure 4-1 for each problem.

**Collected Data**

The researcher collected six types of data for each question from each participant, which are the participants' survey responses, video and audio recording of their problem-solving endeavor, answers and notes, and interview responses. Additionally, the researcher also generated memos about the participants' behaviors.

## Data Analysis Method

This section discusses the detailed data analysis process, which includes organizing, transcribing, coding, analyzing, and triangulating the findings and interpretations of collected data. Additionally, the researcher generated memos related to the analysis and interpretation. In qualitative research, developing memos is an integral part of the analysis process because it helps researchers to gather ideas and develop theories about the data (Creswell, 2012).

**Data Organization**

The collected data were classified and stored based on the case (i.e., participant) and then by the problem. In each problem, there were two types of data, which were the primary and secondary data. The primary data refers to all data points that can be used to answer the research questions, which include survey responses, problem-solving recorded audios and videos, and interview responses. The secondary data refers to all data points that can be used to triangulate and refine the findings and interpretations, which include answers, notes, and the researcher's memos.

**Preliminary Analysis and Member Checking 1**

The goal of the preliminary analysis and member checking is to identify and clarify participants' ambiguous and unclear activities and self-reports. Creswell (2012) argues member checking is important to improve credibility of findings and interpretation from the participants' point of view. The preliminary analysis consisted of three steps. First, developing descriptions of each participant' ambiguous and unclear activities and self-reports. Second, asking each participant for clarification via email (i.e., member checking). Third, incorporating participants' clarifications as transcription memos.

**Transcribing and Coding**

This process was only applicable to the problem-solving recorded audios and videos. The goal of transcribing is to reduce the data complexity (i.e., from multimedia to text) so it will be easier to be coded and analyzed (Lapadat & Lindsay, 1999). The audio and video files were transcribed verbatim to capture every spoken word, including the false starts and stutters (Tigerfish, n.d.). Additionally, following lesson learned from the

pilot study (see Chapter III), the researcher described contexts surrounding participants' activities, wrote the first letter of related concepts in capital, and used a dash ("-") to indicate a quick focus change on participants' cognition.

The transcriptions then were independently segmented and coded based on BCM strategic action (see Table 2-3) by three coders, which were the researcher, an information technologist, and a Ph.D. candidate in engineering education. The information technologist was responsible for the third problem (i.e., Monopoly in the Middle-Ages) because he was familiar with the object-oriented paradigm and had experience in developing an Android game. The Android is an open operating system for small devices, such as phone, that based on Java™ and object-oriented programming (Google Inc., n.d.-b, n.d.-a). The Ph.D. candidate was responsible for the fifth problem (i.e., The Last Standing Man) because he was familiar with the imperative programming paradigm. Additionally, since the Ph.D. candidate has Master and Bachelor in engineering, he has a strong mathematical skill, which was necessary for understanding the participants' approach to solving the fifth problem. A qualitative analysis software, the MaxQDA version 11 and 12 (see http://www.maxqda.com/), was used during the coding process, and a practice session with each coder was held prior the independent coding.

The qualitative coding is an interpretive activity, not a precise science (Saldana, 2008). It is a step to organize and understand the collected data (Basit, 2003). Naturally, all coders returned with different results in some parts of the text. These differences were resolved through discussions, and the inter-coder agreement in the form of Kappa score

was calculated. Kappa score is one of the common method to calculate inter-coder agreement (Viera & Garrett, 2005). By employing two coders in each problem and having a Kappa score between 0.81 to 0.99, the researcher could ensure the coding reliability (Viera & Garrett, 2005). Reflecting on the pilot study experience, it was important to not solely depend on the transcriptions during the qualitative coding because it could not capture all the relevant contexts of the participants' cognitive activities. Therefore, verification through videos was necessary during the inter-coder discussion. Using the final codes, the researcher identified the participants' self-regulation activities and determined the task interpretations associated with the identified self-regulation activities.

**Analysis**

The goal of the analysis was to answer the research questions. To be more specific, the analysis aimed to identify participants' initial understanding, their transformed task interpretation, and factors influencing the task revisions. The researcher only analyzed the collected data related to the central programming problems, which were the third and fifth programming questions. Both problems required the participants to use different concepts and work on a different abstraction level.

**Identifying Participants' Initial Interpretation.** In this analysis, the researcher used the participants' survey responses, participants' interview responses, and the researcher's memos. The survey responses contained the participants' explicit and implicit task interpretation. The interview responses contained the participants' perception of their task revision and unreported thought processes. The goal of this

analysis was to develop a list of participants' initial task interpretation. This analysis consisted of four steps. The first step was to prepare a list to record the participant's explicit and implicit task interpretation. The second step was to move the participant's survey response to the list. The third step was using the participant's interview response to identify initial task interpretation and put it on the list. The fourth was using the researcher's memos to determine entries related to the participant's initial task interpretation and put it on the list. Since there were four participants, this step was repeated four times. After that, the list of participants' initial task interpretation was completed.

**Identifying Participants' Task Interpretation Revisions.** In this analysis, the researcher used the participants' interview responses, final codes, and the researcher's memos. The goal of this analysis was to develop a list of the participants' task interpretation revision and their relationship with the initial task understanding. This analysis consisted of five steps. The first step was to prepare a list for recording the participant's transformed task interpretation and its relationship with the initial task understanding. The second step was using the participant's interview responses to identify transformed task interpretation and put it on the list. The third step was using the final codes to identify activities that could not be associated with the identified task interpretation, and put it on the list. The fourth step was using the researcher's memo to determine entries that identify the participant's task interpretation revision and put it on the list. The fifth step was to identify the relationship between identified transformed and

initial task interpretation. Since there were four participants, this step was repeated four times. After that, the list of participants' transformed task interpretation was completed.

**Identifying Influencing Factors in Participants' Task Revisions.** In this analysis, the researcher used participants' interview responses, final codes, and the researcher's memos. The goals were to enhance the list of the participants' task interpretation and revision by adding the activities that justify those revisions and develop themes for those activities. This analysis consisted of four steps. The first step was to open the list of the participant's transformed task interpretation and its relationship. The second step was using the participant's interview responses to identify the participants' justifications related to the transformed task understanding and put it on the list. The third step was using the final codes to identify monitoring activities related to the transformed task interpretation and put it on the list. The fourth step was using the researcher's memos to determine entries that identify task interpretation revision-related activities and put it on the list. Since there were four participants, this step was repeated four times. At this point, the list of task interpretation revision-related activities was completed. The next step was to segment and code those activities by employing open coding and then followed by developing categories and themes based on the codes.

**Member Checking 2**

The purpose of member checking is to validate the credibility of findings and interpretation from the participants' point of view (Creswell, 2012). For the second member checking, the researcher developed a personalized SRL report based on the analysis results. The report included a brief description of the problems, participant's task

interpretation, the participant's task revision, a comparison of the participant's performance to others, and suggestions to improve the participant's self-regulation skills. Each participant was asked to comment on the report and their identified self-regulation strategies. The researcher included those comments in the dissertation report. If one of the participants disagreed with the report and the researcher agreed with him or her, then the researcher adjusted the report. If not, then the researcher only reported it as comments from the participants.

# CHAPTER V

# THE PARTICIPANTS AND FINDINGS

## Introduction

This chapter starts by describing the research participants and recruitment challenges. After that, the qualitative coding result is described, followed by brief depictions of the participants' approaches to solving the third and fifth problems (i.e., the units of analysis). Last, the chapter discussion continues with answering the research questions.

## The Participants

The participants were essential elements of this study because they were the sources of knowledge that enabled the researcher to answer the research questions. Four participants were recruited, and they provided digital consent in the application form and also signed the letters of consent at the beginning of data collection session. Please refer to Chapter IV for details on participant recruitment and selection method. The higher- and lower-performing participants in each group (i.e., male and female) were Jake and Rusty, and Anne and LStew, respectively. Each participant had a GPA above 3.00 on a 4-point scale and received a $40 gift card and a personalized self-regulation report (see Appendix P for more details). This section focuses on describing recruitment challenges and the participants.

### Recruitment Challenge

Facing challenges when recruiting research participants is a common issue in any research involving human participation (Gul & Ali, 2010; Koo & Skinner, 2005; Leonard

et al., 2003), including this study. There were only eight males and one female students who applied as research participants in Fall 2017. Please note that the online application form only yielded participants who matched with the study criteria (please see Chapter IV and Appendix H for details). The researcher then selected one male applicant with the highest GPA, another male with the lowest GPA, and the only available female applicant. In Spring 2018, the researcher disseminated a recruitment announcement for a female participant, one female student responded and was selected as the final participant. The limited number of applicants prevented the researcher from selecting wider GPA range.

The Office of Analysis, Assessment, and Accreditation (2017), Utah State University (USU) reported that 624 people were registered as full or part-time undergraduate CS students in Fall 2017. Out of those, 201 students were seniors, which consisted of 177 (88%) males and 24 (12%) females. According to Cora Price, the second staff assistant of USU CS department, some senior students had jobs or only registered in online courses. Further, Price explained that some of them only registered as active students but did not take any courses due to various reasons, such as serving on a religious mission.

**Jake**

Jake was a 25-year old Caucasian male with 3.96 GPA on a 4-point scale and was familiar with imperative, object-oriented, and logic programming paradigms. He passed Introduction to Computer Science 1 (CS 1400) course with an A and completed Calculus I, Calculus II, Discrete Mathematics, Linear Algebra, Introduction to Computer Science 2, Methods in Computer Science, Algorithms and Data Structures, Introduction to Event

Driven Programming and GUI's, Introduction to Software Engineering, Advanced Algorithms, Operating Systems and Concurrency, and Developing Dynamic, Database-Driven, Web Applications courses with a C- or better. These courses indicated that Jake had more than the necessary knowledge to answer all programming problems in this study. Jake also mentioned that he had served as a teaching assistant for CS 1400. During the data collection, he correctly answered all practice (i.e., the first and second) and break (i.e., the fourth) questions.

Jake had an intense interest (i.e., ten out of ten) in computer programming and had spent around 5800 hours in developing those skills. Jake also mentioned that Biochemistry affected his programming abilities; he stated, “I feel that Biochemistry courses have given me a unique perspective on programming. There are many correlations between protein and sensory regulations and software input/output that have helped me grasp and apply new principles quickly.” In Biochemistry, one needs to understand a molecule’s structure, function, and behaviors (Biochemical Society, n.d.). In a sense, trying to understand a molecule is similar to comprehending a computer program, a class, or a function. Through Biochemistry, Jake developed a correct model of a typical programming design, which then helped him to understand various computing principles easily. Ben-Ari (1998) argues that trying to understand various CS concepts will become easier when one has correct and effective cognitive models associated with those concepts.

**Rusty**

Rusty was a 23-year old Caucasian male with 3.10 GPA on a 4-point scale and was familiar with imperative, object-oriented, logic, and visual programming paradigms. He passed Introduction to Computer Science 1 course with an A and completed Calculus I, Calculus II, Discrete Mathematics, Linear Algebra, Introduction to Computer Science 2, Algorithms and Data Structures, Introduction to Event Driven Programming and GUI's, Introduction to Software Engineering, Operating Systems and Concurrency, and Developing Dynamic, Database-Driven, Web Applications courses with a C- or better. These courses indicated that Rusty had more than the necessary knowledge to answer all programming problems in this study. Rusty also mentioned that he had served as a teaching assistant for CS 1400. During the data collection, he correctly answered all practice (i.e., the first and second) and break (i.e., the fourth) questions.

Rusty had an intense interest (i.e., ten out of ten) in computer programming and had spent 4160 hours in developing those skills. He did not share any personal or practical factors that might affect his programming abilities.

**Anne**

Anne was a 22-year old Caucasian female with 3.62 GPA on a 4-point scale and was familiar with the imperative and object-oriented programming paradigm. She passed Introduction to Computer Science 1 course with an A and completed Calculus I, Calculus II, Discrete Mathematics, Introduction to Computer Science 2, Algorithms and Data Structures, Advanced Algorithms, Introduction to Event Driven Programming and GUI's, Introduction to Software Engineering, Operating Systems and Concurrency, and

Developing Dynamic, Database-Driven, Web Applications courses with a C- or better. Further, she was registered in the Programming Languages course during the data collection. These courses indicated that she had more than the necessary knowledge to answer all programming problems in this study. Anne also mentioned that she had served as a tutor. During the data collection, Anne correctly answered all practice (i.e., the first and second) and break (i.e., the fourth) questions.

Anne had a medium interest (i.e., four out of ten) in computer programming and had spent around 2000 hours in developing those skills. She did not share any personal or practical factors that might affect her programming abilities.

When asked about the challenge of being a female computer science, Anne said, "Because there are not as many women, you do not have as many people to gauge it off … It is harder to know where you really stand with people." She elaborated that knowing that some of her classmates were able to easily understand challenging CS concepts lowered her sense of belonging; it was an "intimidating dynamic." Further, she mentioned that it was hard for an 18-year old female student to know that some of her classmates were exposed to programming, computational thinking, and CS prior pursuing their computer science degree; Anne said, "It is really hard not to quit before you recognize that." Anne's feeling was consistent with variously reported findings that the sense of belonging is essential for students, especially females (Falkner, Szabo, Michell, Szorenyi, & Thyer, 2015; Lewis, Anderson, & Yasuhara, 2016). Anne further said:

> "I know I am not as good as other people think I am, and as soon as they find out how bad I am at programming, then they will realize that I should not be here."

Such feeling is commonly known as the imposter syndrome. De Vries (1990) argues that people with imposter syndrome tend to “adopts a survival strategy based on inauthenticity in order to win approval of others” (p.678), which then preventing them to internalize their successes including in an academic environment (Clance & Imes, 1978; Cope-Watson & Betts, 2010).

During her final years and after competing in an internal programming contest, Anne was able to overcome her incompetent perception. She said, “For the last four years, I thought that I am not as smart as you guys [her peers] but that was all made up in my head.” She had served as the President of several clubs and as a college ambassador. She also involved in the Association for Computing Machinery for Women (ACM-W), the women chapter of ACM, as a mentor, where she helped other female students to have a positive and rewarding experience throughout their education.

**LStew**

LStew was a 22-year old Caucasian female with 3.36 GPA on a 4-point scale and was familiar with imperative and object-oriented programming paradigms. She passed Introduction to Computer Science 1 course with an A and completed Calculus I, Calculus II, Discrete Mathematics, Introduction to Computer Science 2, Algorithms and Data Structures, Introduction to Event Driven Programming and GUI's, Introduction to Software Engineering, Operating Systems and Concurrency, and Developing Dynamic, Database-Driven, Web Applications courses with a C- or better. Further, she registered in the Advanced Algorithms course during the data collection. These courses indicated that LStew had more than the necessary knowledge to answer all programming problems in

this study. During the data collection, she correctly answered all practice (i.e., the first and second) and break (i.e., the fourth) questions.

LStew had a strong interest (i.e., eight out of ten) in computer programming and had spent around 2100 hours in developing those skills. She also mentioned that her father, self-practice, self-efficacy, and self-comparison affected her programming abilities. LStew mentioned that her father was her mentor before and during her college career, and shared stories on how her father encouraged her pursuing her dream to become a computer scientist. LStew's father had served as one of the mentors for her robotics team in high school and became her private tutor for various courses. LStew's positive experience with mentoring is consistent with Ko & Davis' (2017) report that mentoring has a positive influence on students' perception of and interest in CS.

In addition to having a personal mentor, LStew also gained benefits by engaging in self-practice activities, including during her internship and as a teaching assistant for CS 1400. She said, "My internship at the Space Dynamic Laboratory made me a lot more proficient. I also think that being a teaching assistant for CS 1400 has helped me understand the basics of C++ a lot better and be more passionate about it." It was clear from her statement that practicing programming improved her self-efficacy. Miller et al., (2013) argue that the best way to improve students' computer science self-efficacy is through continuously applying the computer science principles. Additionally, Litchfield, Javernick-Will, & Maul (2016) argues that students' design experience in a highly contextual and complex environment improves their professional skills, or in this case, programming skills. Bandura (1986) defines self-efficacy as "people's judgments of their

capabilities to organize and execute courses of action required to attain designated types of performances" (p.391). Several studies reported there was a strong correlation between students' self-efficacy and the quality of their learning performance (Al-mehsin, 2017; Joo, Bong, & Choi, 2000; Paraskeva, 2007; Santoso, Lawanto, Becker, Fang, & Reeve, 2014; Santoso, 2013; Siddique, Hardré, & Altan, 2015). Similar to these reports, LStew mentioned how self-efficacy was affecting her programming abilities by saying, "I nearly failed a class because I did not believe I was capable of succeeding in it."

Lewis, Anderson, & Yasuhara (2016) reported that stereotypes are important for students including in computer science, and that CS students often assess their fitness to CS stereotypes which then affects their performance and feeling of belonging. LStew was not an exception; she said, "I have to ignore my colleagues and classmates programming 'successes' as that comparison game tends to reduce my self-esteem a lot and negatively impact my problem-solving and programming capabilities." LStew was not alone; while she was able to dismiss the negative effects of CS stereotyping, which was "singularly focused on CS, asocial, competitive, and male" (Lewis et al., 2016, p.30), she shared that some of her female friends were still struggling with it. Some studies argue that one of the reasons for women underrepresented in computing discipline (Fisher & Margolis, 2002; Galpin, 2002), including at USU (Office of Analysis, Assessment, and Accreditation Utah State University, 2017), is the stereotype of computer scientists (Graham & Latulipe, 2003; Irani, 2004; Outlay, Platt, & Conroy, 2017; Wang, Hejazi Moghadam, & Tiffany-Morales, 2017). Consistent with Irani (2004)'s report, LStew

mentioned that some female students felt they had to work harder to make people recognize their abilities.

## Qualitative Coding Results

The qualitative coding involved three coders, which were the researcher, an information technologist, and a Ph.D. candidate in engineering education (see Chapter IV for details). Please note that the coding process of the last participants' (i.e., Anne) transcriptions were conducted by the researcher and an information technologist. The interrater agreement (i.e., Kappa score) was calculated for each participant on each problem using MaxQDA, and the initial scores were in the range of -0.18 to 0.01, which indicates agreements by chances (Viera & Garrett, 2005). In calculating the Kappa score, MaxQDA also takes into account the segment size (MaxQDA, n.d.), in such that two coders need to have at least a 90% similar segment and use the same code to label that segment; the 90% segment similarity is MaxQDA default value and can be adjusted accordingly. Thus, having different codes was not the only reason for the poor agreement scores, but also due to the differences in segment size.

The most accurate strategic action code was not only influenced by the participants' action but also by their prior actions. For example, the 'enacting' code in Table 5-1 was appropriate because Rusty said those words after verbalizing his plan to check the algorithm's output for six inputs. Another example, when LStew was solving the third problem, she said:

> "And if I am a thief, maybe I can steal from a building, but I do not know how that would work with the rules of Monopoly. Anyway, I can think about that later."

Both coders agreed to label the first sentence as 'monitoring.' The second sentence was aligned with the definition of 'planning,' which is selecting appropriate cognitive and metacognitive strategies for completing any tasks (Butler & Cartier, 2005). However, because the second sentence occurred after LStew engaged in monitoring activity, the most appropriate code would be 'adjusting,' which refers to students' strategies adjustment based on self-perceptions of progress or feedback (Butler & Cartier, 2005). Thus, both coders agreed to code the second sentence as 'adjusting.'

*Table 5-1.*

Segment Example for Each Strategic Action Code

| **Strategic Action Code** | **Example** |
|---|---|
| Task Interpretation | "I am looking at this sentence, 'two, three, four players,' that is important,…. So, two to four." – Lstew when solving the third problem. |
| Planning | "I am going to grab one of these papers." – Jake when solving the fifth problem. |
| Enacting | "*[Writing it down]* 4 5 6. Right, 1 kills 2, gives the sword to 3, so 1 3 4 5 6. 3 now has the sword, he kills 4 and gives it to 5, so we have 1 3 5 6. 5 kill 6 and gives the sword back to 1, so we have 1 3 5, and then 1 kills 3 gives the sword to 5, and 5 kills 1, and 5 is the last man standing." – Rusty when solving the fifth problem and after saying, "I will do six people instead and see who survives." |
| Monitoring | "Just occurred to me, I should have been crossing things off for this paper as I had them written down." – Jake when solving the third problem. |
| Adjustment | "So before I continue, I am going to skim through again and make sure that I do understand, and that there are no any small details that I forgot." – Rusty when solving the third problem. |

Selecting the correct segments was important in this study because it helped the researcher to identify various thinking episodes, which could be determined by identifying the contexts related to each thought process (Butler & Cartier, 2005). As an example, at the end of his endeavor in solving the third problem, Jake said:

> "All right. So, that is everything-all have been taken care of. Now going along with the plan I had written down earlier, I would rewrite this *[solution]*, so it is more readable. *[That is]* just what I would do if I were showing this to an employer …"

The above passage was related to Jake's monitoring activity, but it should be coded as two segments. The first segment, which was the first and second sentences, was about Jake's monitoring activities of his progress in solving the problem. The second segment, which was the third and fourth sentences, was about Jake's monitoring activities about his progress toward conforming to his overall problem-solving approach.

In self-regulated learning (SRL) research, it is important to identify students' learning episodes and how they shifted through those episodes (Butler & Cartier, 2005; Winne & Perry, 2000). Therefore, during the meeting, the coders did not only discuss their code disagreements but also segment differences. On average, each coder in each problem made 65 code changes including the segments. After the discussion, all coders agreed on 1607 codes with the final Kappa score of 1.00 for each transcript, which indicates perfect agreements (Viera & Garrett, 2005). Table 5-1 presents examples of each code segment.

When coding the participants' transcriptions of the third problem, both coders agreed to consider most of the participants' rereading activities as task interpretation because they were appeared as understanding the problem for the first time. However, not all of their rereading activities were considered as task interpretation, for example, when Jake was verifying his interpretation on Buildings' characteristics and said, "Just double-check what the Buildings do; Buildings need to keep track of who owns them," both coders agreed to label it as monitoring activity.

### Participants' Self-Regulation in Solving the Third Problem

The third problem was Monopoly in the Middle-Ages. This ill-structured problem asked the participants to design a base for a digital version of a classic board game using the object-oriented programming paradigm. The problem provided detailed requirements and constraints including at least 18 issues, 24 functions, and 22 variables. Furthermore, it asked the participants to go beyond the listed requirements when appropriate and use their creativity to produce a thorough and extensive design. Under the Bloom's Taxonomy described in Gronlund et al. (2013), this problem is at level 6.2 which is creating, planning, or devising steps to accomplish a certain task. Gronlund et al. (2013) subcategorize level 6 Bloom's Taxonomy (i.e., creating) into three, which are generating/hypothesizing, planning/designing, and producing/constructing. It was necessary to know basic programming and object-oriented design to answer this question. Chapter IV presents a detailed discussion of this programming problem. In this section, the participants' approach to solving the third problem is described, including their initial

task interpretation (i.e., prior to solving the problem), problem-solving approach, and self-regulation activities.

**Jake's Self-Regulation in the Third Problem**

**Initial Task Interpretation.** Jake described the goal of this problem as "developing a class diagram and modeling all possible relationships between five or six different classes, such as players, building, square *[space]*, and items." Jake was aware that he needed knowledge and concepts of object-oriented design, including a class diagram. Since object-oriented programming is an extension of imperative programming (Lee, 2014), it can be implied that Jake is also referring to needing basic programming knowledge. It was clear that Jake's explicit understanding of this problem was correct.

Jake recognized that he needed to consider around ten requirements described in the problem when designing the solution and that he could not remember everything, except that there would be "player classes, items, buildings, player-action per turns, and all interacted in a specific way." In other words, Jake acknowledged there were many requirements that he needed to consider when solving this problem.

He believed that his Software Development (CS5700) course and work experience in refactoring a program would be valuable assets. Further, Jake elaborated that in the software development course, students were required to engage in similar planning activities (i.e., developing a class diagram) before writing any code. Refactoring is an advanced programming task, which is defined as "the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure" (Fowler & Beck, 1999, p.xvi). When refactoring, the

programmer must have both the overall and specific knowledge about the program and then develop an adjustment plan while keeping the program's external behavior intact.

As part of his implicit understanding of this problem, Jake described five steps to solve it. First, he needed to reread the problem. The previous explained implicit understanding (i.e., the paragraph above) influenced this first step, in which he was aware of multitudinous requirements and constraints in this problem but could not remember all of those. Second, he needed to create a rough draft of possible classes. Third, he needed to use entity-relationship diagram (ERD) notation to express the relationships among classes. He mentioned, "I have been working on the entity-relationship diagram a week and a half ago, so I want to model the classes' relationships like that," suggesting that Jake was more familiar with ERD compared to the class diagram because he engaged with ERD recently. The ERD is commonly used to describe a relational structure of a database system (TechTarget, n.d.), not a structure of an object-oriented system. Therefore, some classes' relationships could not be expressed correctly using the ERD, such as inheritance and realization. Fourth, he needed to iteratively adjust the classes' relationships until all the requirements were met. Fifth, he needed to evaluate his progress and identify chances to optimize, clarify, or simplify the design.

**Problem-Solving Approach.** As presented in Figure 5-1, Jake's approach to solving the problem was aligned with his initial problem-solving steps (i.e., the paragraph above). Since, he was starting "off with a vague idea of what the requirements were," Jake began by identifying the task goal and subgoal, and then went through each problem requirement sequentially twice. In his first iteration (i.e., went through the requirements),

Jake reread, interpreted, and solved each problem requirement. In other words, he was enacting the first, second, and third problem-solving steps. In his second iteration, Jake monitored his progress and clarified and simplified his design, which aligned with his fourth and fifth problem-solving steps.

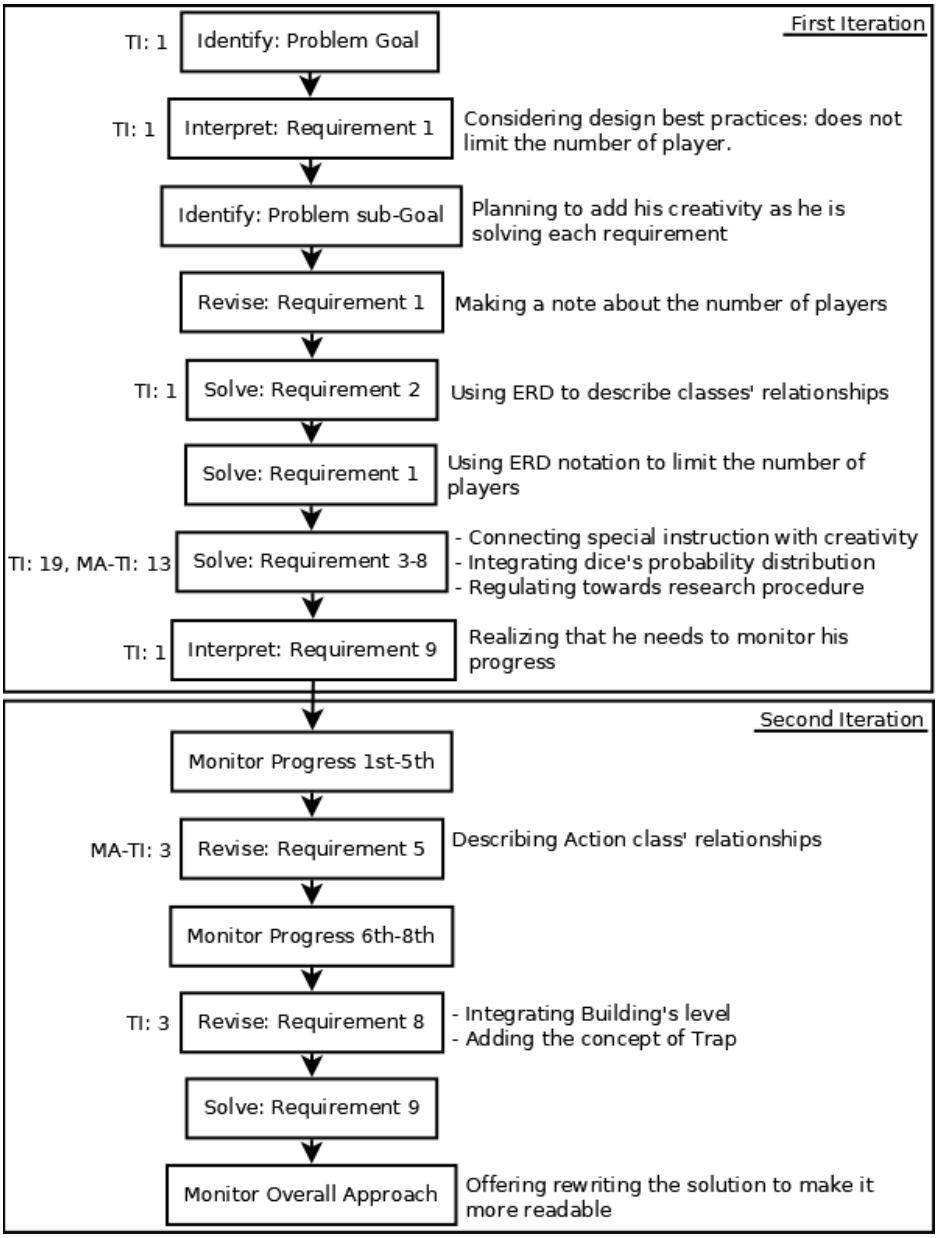



*Figure 5-1. Jake's approach for the third problem.*

Figure 5-1 presents Jake's problem-solving approach using a modified flowchart, in which the notations are consistent with the common flowchart symbols (Lucid Software Inc., n.d.), but it assumes the first and the last box as the first and last activity, respectively. The boxes represent Jake's observed problem-solving activities, the texts on the left represent the number of codes related to Jake's observed task interpretation (TI), and monitoring and adjusting on TI (MA-TI), and the texts on the right provide short elaborations on his problem-solving activity.

During his problem-solving endeavor, Jake was observed verbalizing 112 instances of strategic actions including 26 task interpretation (TI), seven planning strategies, 18 enacting strategies, 52 monitoring (M) activities, and nine adjustment (A) strategies; the number of code is presented to provide a better picture of the participant's self-regulation. Butler & Cartier (2005) argues each strategic action (i.e., planning, enacting, monitoring, and adjusting) starts with task interpretation, and any monitoring activities on task interpretation can result in a revised understanding of the problem. The researcher found all Jake's observed planning and enacting strategies were aligned either with his initial understanding of the problem or observed task interpretation and monitoring and adjusting activities on his understanding of the problem. For example, when incorporating building level requirement into his design, Jake said:

> "So... since this game is only 20 turns long, I am going to limit it *[the building's level]* at three, and each *[building]* has a level 1 property, level 2 property, and level 3 property" (i.e., a planning strategy).

Jake's decision to limit the building levels to three was informed by his understanding that there were only 20 turns in the game. Since this study focus on task interpretation and all Jake's other observed strategic actions were sequels of his task interpretation, focusing further analysis on his observed TI and MA-TI would be sufficient to answer the research questions. As presented in Figure 5-1, Jake's TI and MA-TI activities occurred throughout the problem-solving process, which suggests that he was continuously refining his understanding of the task as he worked through the problem.

When interpreting the requirements, Jake did not only consider given information but also integrated various issues, including original Monopoly's rules, prior gaming experience, the probability distribution of everyday events, the hypothetical company's structure, gameplay, and his awareness on his partial understanding of the game requirements. As a result, Jake's interpretations of the problem were sometimes beyond what was expected from the problem. For example, when interpreting the virtual dice's behavior, he considered the real-life dice's behavior and said:

> "Well, in the original game it *[the dice' values]* was *[between]* 2 to 12, but it had a probability curve that was greater towards the center. Do I need to mimic that too?" (i.e., monitoring his task interpretation).

Since the problem description did not have any specific instruction related to such behavior, Jake's decision to include it might be influenced by his prior experience, interest, or something else. He later confirmed (i.e., during the interview) that he had a passion for probability distribution functions. While this study considers the nature of Jake's contemplations as part of his self-regulation, other researchers may consider it as

examples of deep thinking (Fischer & Hommel, 2012; Renesse & DiGrazia, 2018; Wiersema & Licklider, 2009).

As presented in Figure 5-1, Jake occasionally monitored and adjusted his task interpretation throughout his problem-solving endeavor, in such that 26.23% of his monitoring and adjusting activities were related to task interpretation; the MA-TI percentage is given to provide a better picture on the participant's self-regulation. His MA-TI activities were related to remembering the requirements, associating his understanding of the problem with known concepts, confirming his interpretation by rereading the problem description, being aware of forgotten requirements, interpolating his interpretation, and adding creativity to the design, and all except the first two resulted with a revised task interpretation. For example, when he was wondering whether a Player could take multiple actions per turn, he said:

> "So, 1-to-1. In every turn *[a Player]* will have to move … *[based on]* possible actions. Can they take multiple actions per turn? *[Re-reading the problem description]* 'They can choose to do any of the following,' I imagine that means any one of the following *[actions]*. So, possible of 0 actions or 1, and at most one action per turn" (i.e., monitoring and adjusting his task interpretation).

In the first sentence, Jake was interpolating his understanding of the problem by considering multiple actions per turn. In the second, third, and fourth sentences, Jake was confirming his interpolation by rereading the problem description and then came up with the most relevant conclusion.

By comparing Jake's final design against the problem-space map, there were some missing design details including Items benefit for the Players, the access level (e.g., public or private) of the classes' properties and methods*, the trigger for special instruction*, the mechanics for determining Players' location on the board, the mechanics for initializing all game instances*, the mechanics for declaring the winner and stop the game*, and the classes' constructors*. The issues with an asterisk (*) were most likely caused by the limitation of ERD and its notations. As stated earlier, the ERD is not designed to describe an object-oriented system. This finding suggests that Jake's interpretation was incomplete and most of his incomplete interpretations were caused by selecting inappropriate modeling language for solving the problem. Jake's situation is consistent with Isomöttönen & Tirronen (2013)'s argument that relevant knowledge and skills are essential for having accurate and efficient self-monitoring activities.

**Rusty's Self-Regulation in the Third Problem**

**Initial Task Interpretation.** Rusty described the goal of this problem as "create*[ing]* a logic layer inside of our program that can function completely without interaction from the graphical user interface or user." Rusty's statement implied that he recognized the problem requirements as part of the game logic. The decoupling between the application logic and user interface is one of the best practices in software engineering (Boudreau, Tulach, & Unger, 2006; US7837556B2, 2001; US8924845B2, 2008; Rails Community, 2014; Unity Technologies, 2018). During the interview, Rusty shared that he learned about the logic-GUI-decoupling in one of his programming course assignment. He believed that resorting to logic and GUI coupling would introduce many

bugs and also complicate the program maintenance. Rusty was aware that he needed knowledge and concepts of inheritance for describing the Character, Items, and Building, and an understanding on "how to write a good class diagram so that they *[people in a hypothetical company]* are prepared to use my code." Since the object-oriented programming is an extension of the imperative programming (Lee, 2014), it can be implied that he is also referring to needing basic programming knowledge. It was clear that Rusty's explicit understanding of this problem was correct.

Rusty recognized that he needed to follow "clearly listed requirements and constraints" while also exercising his creativity when applicable. Although he had never designed a system of a similar size, he believed that his relevant programming assignments (i.e., related to inheritance and class diagram) would be valuable assets. During the interview, Rusty shared that the problem size made him worry, especially because due to multiple interactions in the game and said, "it is hard to assess: Is the design too open? Is this *[design decision]* to prone to bugs? Or have I… *[made]* it only communicate when it needs to?"

Related to steps for solving the problem, Rusty wrote:

> "First, I would draw up the class diagram to give myself a sort of roadmap for completing the assignment. Once I feel I have made it as robust as possible, I would start implementing super- and sub-classes case by case. It will be important to make sure that as I go forward, I am constantly referring to the requirements and constraints to make sure I am successfully completing the assignment."

Based on his description, Rusty's first step was identifying and creating classes based on the task description. His second step was restructuring the classes by utilizing the inheritance concept. Additionally, while designing, he would continuously monitor his progress and design compliance with the requirements.

**Problem-Solving Approach.** As presented in Figure 5-2, Rusty's approach to solving the problem was slightly different from his initial problem-solving approach (i.e., the paragraph above). Rusty began by verifying the problem goal, which was providing a class diagram. This step was not mentioned in his problem-solving approach. He then continued by rereading the problem description to "make sure that I do understand, and that there are no any small details that I forgot." Similarly, this process was not mentioned as one of his problem-solving steps. During the interview, Rusty explained that he frequently reread a problem description multiple times prior solving it because he was aware that "there were sentences and little lines that I did not catch the first time I read it." Therefore, it was possible that Rusty did not mention this step because he considered it as an inherent problem-solving approach. Interestingly, even though he was aware that he might miss some small critical details when he first read the problem description, Rusty only made mental notes during his rereading endeavor. Rusty then created the class diagram for each issue (e.g., the Board class, its properties and methods, and sub- and supporting classes and their relationships) based on his interpretation, and optimized the classes as he moved forward. During his design endeavor, he frequently monitored his progress and the design compliance with the requirements. In other words,

Rusty enacted his problem-solving steps after confirming the problem goal and rereading the problem description.

When solving this problem, Rusty was observed verbalizing 331 instances of strategic actions including 55 task interpretation, 21 planning strategies, 32 enacting strategies, 204 monitoring activities, and 19 adjustment strategies. Butler & Cartier (2005) argues each strategic action (i.e., planning, enacting, monitoring, and adjusting) starts with task interpretation, and any monitoring activities on task interpretation can result in a revised understanding of the problem. The researcher found all Rusty's observed planning and enacting strategies were aligned either with his initial understanding of the problem or observed task interpretation and monitoring and adjusting activities on his understanding of the problem. For example, when designing the Items for the Character class, Rusty extended his understanding of that issue and said:

> "… pretty sure that a Character will start with some predefined Items; I remember it saying that. *[Writing it down]* Array of Items and then as well as an amount of money that they start with" (i.e., task interpretation followed by enacting strategy).

Rusty's decision to include starting amount of money inside the Character class was informed by his understanding of the problem requirements on that issue. Since this study focus on task interpretation and all Jake's other observed strategic actions were sequels of his task interpretation, focusing further analysis on his observed TI and MA-TI would be sufficient to answer the research questions. As presented in Figure 5-2, Rusty's TI and MA-TI activities occurred throughout the problem-solving process, which suggests that

he was continuously refining his understanding of the task as he worked through the problem.

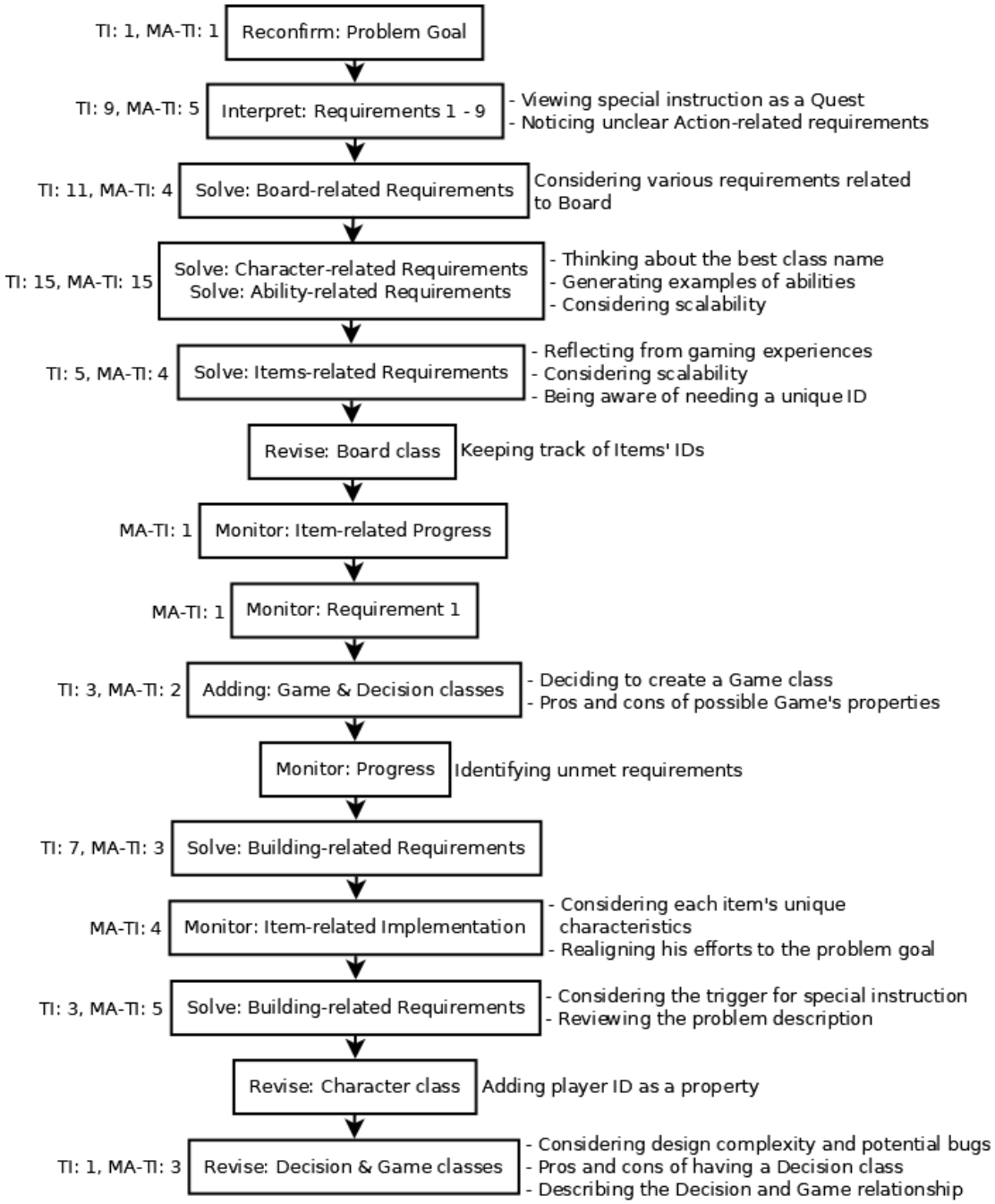



*Figure 5-2. Rusty's approach for the third problem.*

When interpreting the requirements, Rusty considered not only the provided information but also various issues, including the design clarity for future maintenance, prior gaming experience, gameplay, and his awareness of his partial problem understanding. Consequently, Rusty occasionally interpreted the task beyond what was required of the problem. For example, when he was describing the Item class' characteristics, he contemplated whether the Item had a price value or not and made a deduction by considering one of the item-related actions; he said, "…but if you can purchase them *[items]* from a shop, my assumption is that they do have a value" (i.e., task interpretation).

Rusty was observed engaging in monitoring and adjusting activities throughout his problem-solving endeavor, and as presented in Figure 5-2, 21.52% of those activities were related to task interpretation. In more specific, these MA-TI activities were about remembering the requirements, translating understanding to known concepts, clarifying problem scope, rereading the problem description, recognizing forgotten requirements, expanding understanding of the problem, and adding creativity to the design. All except the first two issues resulted in a revised task interpretation. As an example, when Rusty was generating possible implementations of Item's and Character's unique benefits and abilities respectively, he was overwhelmed by the vast possibilities. Rusty then said, "There is a lot of implementation *[details]* if you want to make it a robust game; we would not focus on that too much" and stopped generating further examples and refocused his problem-solving effort to complete the rest of the requirements.

By comparing Rusty's final design against the problem-space map, there were some missing design details including the mechanics for initializing starting Items and money, initializing Buildings on the Board, declaring a winner, and stopping the game. Further, there were some design issues that he thoughtfully considered and solved but not written including the details of special abilities, mechanic for virtual dice, Items benefit for the Characters, and limiting the number of players, board spaces, and turns. Renumol et al. (2010) reported that computer programming requires various cognitive skills and interplay of different level of abstractions which consequently increased brain processing load. Wing (2008) also postulates a similar argument in the context of computational thinking. Therefore, it was possible that Rusty's extensive problem-solving engagement incited his brain to clear some space in the working memory, and combined with lack of design notes, caused him to forget these design details. Anderson & Jeffries (1985)' study offers an explanation for the fact that Rusty still forgets these design details despite his continuous monitoring. They reported that students tend to oblivious to programming errors when there is information lost in their working memory, but the resulting programming is still justifiable. Therefore, this finding suggests that Rusty's interpretation was incomplete and most of his incomplete interpretations were caused by limited monitoring strategies, such as creating a design note.

**Anne's Self-Regulation in the Third Problem**

**Initial Task Interpretation.** Anne described the goal of this problem as "develop*[ing]* class diagram from given constraints." Anne was aware that she needed knowledge and concepts of object-oriented design, including a UML class diagram. Since

object-oriented programming is an extension of imperative programming (Lee, 2014), it can be implied that she is also referring to needing basic programming knowledge. It was clear that Anne's explicit understanding of this problem was correct.

Anne recognized that she needed to "follow given constraints, be creative in *[the]* development, and *[produce a]* clear design" so people in the hypothetical company could easily implement it. She elaborated that "You need to make sure that everything is … organized in a logical way" so people could easily understand how the classes work together. Anne believed that any programming assignments, especially object-oriented projects, would be valuable assets. Further, she said, "I think programming *[experience]* gives you a feel for how many classes is too many, does that *[behavior]* require its own class or could it just be a function."

As part of her implicit understanding of this problem, Anne described two steps to solve it. First, she needed to "go through each of the requirements and make a list of all the classes I think I need." She also said, "I think there were nine of them, but I do not remember them all," which explains the need to reread the problem description. Second, she needed to holistically think about the classes and requirements, such as "how do these relate to each other? Are any of them like subclasses?"

**Problem-Solving Approach.** As presented in Figure 5-3, Anne's approach to solving the problem was aligned to some extent with her initial problem-solving steps (i.e., the paragraph above). Anne began by monitoring the problem goal so she could direct her effort to achieve it. She then reread the problem description, while creating a list of needed classes and holistically thinking about the classes' properties, methods, and

relationships. In other words, she was enacting her problem-solving steps. After finishing reading the problem description, Anne stopped and thought about adding her creativity to the design; she said, “So if I was going to be creative... I honestly do not know. Maybe I will just start designing and then see if I think of something.” Anne admitted that creativity was not one of her strengths. Anne then continued by creating and enhancing a class diagram while continuously aligning her design to satisfy the requirements; this activity was not elicited in her problem-solving step.

Anne was observed verbalizing 170 instances of strategic actions during her problem-solving endeavor, including 25 task interpretation, two planning strategies, 11 enacting strategies, 124 monitoring activities, and eight adjustment strategies. Butler & Cartier (2005) argues each strategic action (i.e., planning, enacting, monitoring, and adjusting) starts with task interpretation, and any monitoring activities on task interpretation can result in a revised understanding of the problem. The researcher found all Anne’s observed planning and enacting strategies were aligned either with her initial understanding of the problem or observed task interpretation and monitoring and adjusting activities on her understanding of the problem. For example, when incorporating an abstraction of various structure types (e.g., shop) in the Space class, Anne said, “Okay, so Foos are made up of I-they are either Building, Shops or Instructions; *[writing it down]* so Spaces are made up of Foos” (i.e., monitoring followed by enacting strategy). Anne’s abstraction (i.e., the Foo class) was informed by her understanding of various structural types that could exist on a Space. Since this study focus on task interpretation and all Anne’s other observed strategic actions were sequels

of her task interpretation, focusing further analysis on her observed TI and MA-TI would be sufficient to answer the research questions. As presented in Figure 5-3, Anne's TI and MA-TI activities occurred throughout the problem-solving process, which suggests that she was continuously refining her understanding of the task as she worked through the problem.

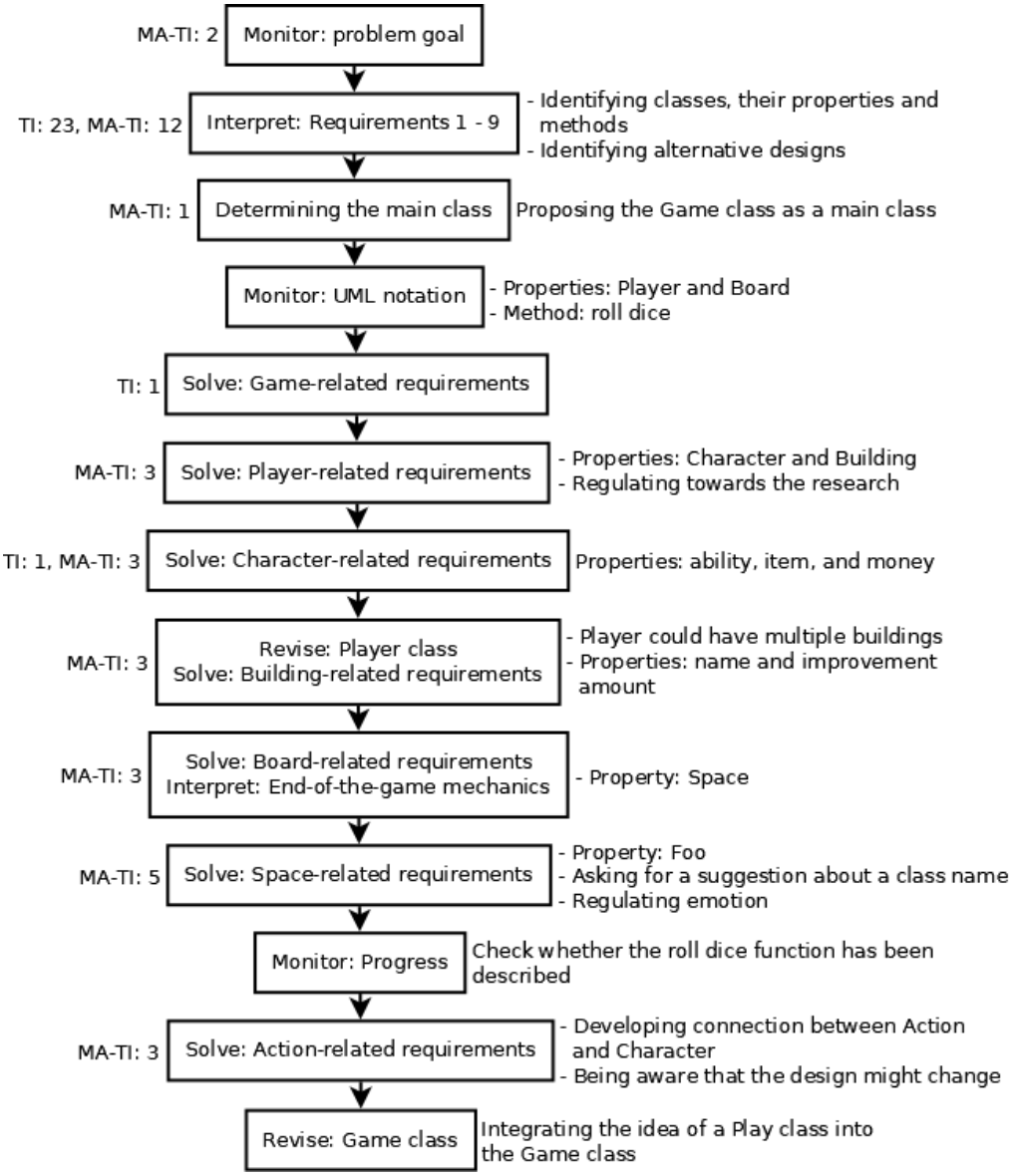



*Figure 5-3. Anne's approach for the third problem.*

When understanding the requirements, Anne considered the gameplay and the original Monopoly's rules, which enabled her to have sufficient interpretations for solving the problem. She was also observed making a direct connection between the requirements and associated approaches to accomplish them. For example, when reading one of the requirements, she said, "Then in *[reading the problem description]* their turn, each player must move, and they can choose to do any of the following; so we need an Action class" (i.e., task interpretation). In this example, Anne instantaneously identified that she needed an Action class. Ashcraft (1992) argues that instantaneous thinking is possible as a result of continuously exercising a particular problem-solving strategy which then strengthens the association between the nature of the problem and the corresponding approach to solving it. Thus, it is reasonable to assume that Anne's programming experience enables her to quickly drawing connections between the requirements and associated approaches.

As presented in Figure 5-3, Anne occasionally monitored and adjusted her task interpretation throughout the problem-solving endeavor, in such that 28.03% of her monitoring and adjusting activities were related to task interpretation. Her MA-TI activities were related to remembering the requirements, associating her understanding of the problem with known concepts, clarifying problem scope, confirming her interpretation by rereading the problem description, and interpolating her interpretation, and all except the first two resulted in a revised task interpretation.

Anne was also observed initiating a discussion with the researcher about her interpretations or approaches, which suggests that she often worked in a pair or a group

and that the research setting might negatively affect her problem-solving process. When being asked about that during the interview, she shared that she had a good friend and they often worked together in various courses. However, Anne's behavior (i.e., initiating a discussion with the researcher) does not suggest a lack of self-efficacy for solving the problem or over-reliance on teamwork. During the last interview, Anne shared that she participated in a team programming contest and was on the top 15$^{th}$ out of 200 teams, suggesting an exceptional self-efficacy on her programming skills. Further, Anne participated alone, which suggests she had outstanding self-reliance. Thus, Anne's behavior (i.e., initiating a discussion with the researcher) demonstrated her competency in using various coregulation skills. Coregulation is a transitional process in which the learners define and update their self-regulation skills for solving a problem through interaction with peers (Hadwin, Jarvela, & Miller, 2011; Rivera-Reyes, Lawanto, & Pate, 2016).

After initiating a discussion and learning that the researcher could not give any suggestions, Anne continued designing the Space class and said, "Well, okay, so Spaces have... um... my learning report is going to be: we do not know how you made it through this far actually" (i.e., monitoring activity). Considering the substance and its timing, the researcher recognized this statement as part of her emotion regulation.

By comparing Anne's final design against the problem-space map, there were some missing design details including the classes' and methods' access level, creativity enhancement, and mechanics to identify the Players' position on the board. A clarity

issue related to the robustness of one of the methods in handling the game logic also existed. This finding suggests that Anne's final interpretation was incomplete.

### LStew's Self-Regulation in the Third Problem

**Initial Task Interpretation.** LStew described the goal of this problem as "to design a system that implements the rules of monopoly in an object-oriented way and that is creative and easy to build upon and add to." She was aware that she needed knowledge and concepts of object-oriented design (e.g., classes, inheritance, dependencies, and decoupling), "UML class diagram, and ease-of-use *[in software design]*." Since the object-oriented programming is an extension of the imperative programming (Lee, 2014), it can be implied that LStew is also referring to needing basic programming knowledge. Seffah, Donyaee, Kline, & Padda (2006) argue there are ten critical factors in software usability (or ease-of-use) including efficiency, effectiveness, productivity, satisfaction, learnability, safety, trustfulness, accessibility, universality, and usefulness. It was clear that LStew's explicit understanding of this problem was correct.

LStew recognized that she needed to "follow the rules and constraints described in the problem" while also exercising her creativity when applicable. She was also aware that other people in the hypothetical company would use her code and that she needed to avoid common object-oriented programming pitfalls by reducing coupling and avoiding the diamond of death. In software design, coupling refers to "to the degree to which software components are dependent upon each other" (TechTarget, n.d.). Thus, tightly-coupled components increase the interdependencies, complexities, and maintenance costs. For example, a programmer needs to update component A of a software system.

However, since component A is tightly-coupled with B and C, the programmer need also to update these two components to ensure the system could work properly. In some programming languages, it is possible for a class to inherit properties and methods from more than one parent classes. The diamond of death is a situation where two or more parent classes have an identical public method signature (e.g., public void printMe()) and is not overridden by the child class (geeksforgeeks, n.d.). In such circumstances, it will be hard to determine from which parent the child class will inherit the method (e.g., printMe()). LStew believed that her experience in Object-Oriented Software Development (CS5700), Introduction to Computer Science 2 (CS1410), and Algorithm and Data Structures (CS2420) courses would be valuable assets.

LStew described eight steps to solve the problem. First, she needed to identify and create a list of requirement. Second, she needed to create a class diagram based on her understanding of the problem. Third, she needed to observe and create interfaces for any possible interplay among the classes. Fourth, she needed to review if there was any noticeable design pattern to be followed. Fifth, she needed to look for any poor design choices. Sixth, she needed to find opportunities for adding creativity to the design, and then noted that instead of performing this plan later, she might as well do it iteratively as she was solving the problem. Seventh, she needed to verify that all requirements were met by rereading the problem description.

**Problem-Solving Approach.** As presented in Figure 5-4, LStew's approach to solving the problem was aligned to her initial problem-solving approach (i.e., the paragraph above) to some extent, in such that instead of enacting the steps sequentially,

she combined them. She began by organizing the requirements and identifying the classes including their characteristics and relationships, which was aligned with her first problem-solving step. LStew continued by reviewing her notes and then drawing identified classes and their characteristics. While she was solving each issue (e.g., the Character class, its properties and methods, and sub- and supporting classes and their relationships), LStew continuously enhanced the design by describing the classes' interfaces, utilizing design known patterns, assessing the benefits of alternative design options, adding her creativity, and ensuring the design compliance with the requirements. In other words, LStew was enacting her second to seventh problem-solving steps.

When solving this problem, LStew was observed verbalizing 262 instances of strategic actions including 84 task interpretation, six planning strategies, 27 enacting strategies, 125 monitoring activities, and 20 adjustment strategies. Butler & Cartier (2005) argues each strategic action (i.e., planning, enacting, monitoring, and adjusting) starts with task interpretation, and any monitoring activities on task interpretation can result in a revised understanding of the problem. The researcher found all LStew's observed planning and enacting strategies were aligned either with her initial understanding of the problem or observed task interpretation and monitoring and adjusting activities on her understanding of the problem. For example, when designing a function for the Shop class, LStew said:

> "And a shop, when you sell an item to a shop, it needs to detract an item from the Player, so it *[shop]* should own that *[number of items]*. *[Writing it down]* So shop, item, there's a function" (i.e., task interpretation by enacting strategy).

LStew's decision to add a function for handling a possible action of selling an item was informed by her understanding of the entailed data flow. Since this study focus on task interpretation and all LStew's other observed strategic actions were sequels of her task interpretation, focusing further analysis on her observed TI and MA-TI would be sufficient to answer the research questions. As presented in Figure 5-4, LStew's TI and MA-TI activities occurred throughout the problem-solving process, which suggests that she was continuously refining her understanding of the task as she worked through the problem.

When interpreting the requirements, LStew considered not only the provided information but also various issues, including known design pattern, prior gaming experience, gameplay, and her awareness of her partial problem understanding. Consequently, LStew occasionally interpreted the task beyond what required of the problem. For example, when she was figuring out the nature of special abilities, she said, "Special abilities… I am trying to think of how that works out because I do not remember special abilities in the characters *[that]* I used to *[play in]* monopoly" (i.e., task interpretation). She then generated some possible implementation of special abilities, such as "If I am a King, maybe I get automatic discount … and if I am a thief, maybe I have the ability to steal *[from]* a building." During the interview, she clarified that although it was not necessary to find examples of special abilities, it helped her to understand their purpose and how to incorporate their behaviors in the class diagram (e.g., the methods' parameters).

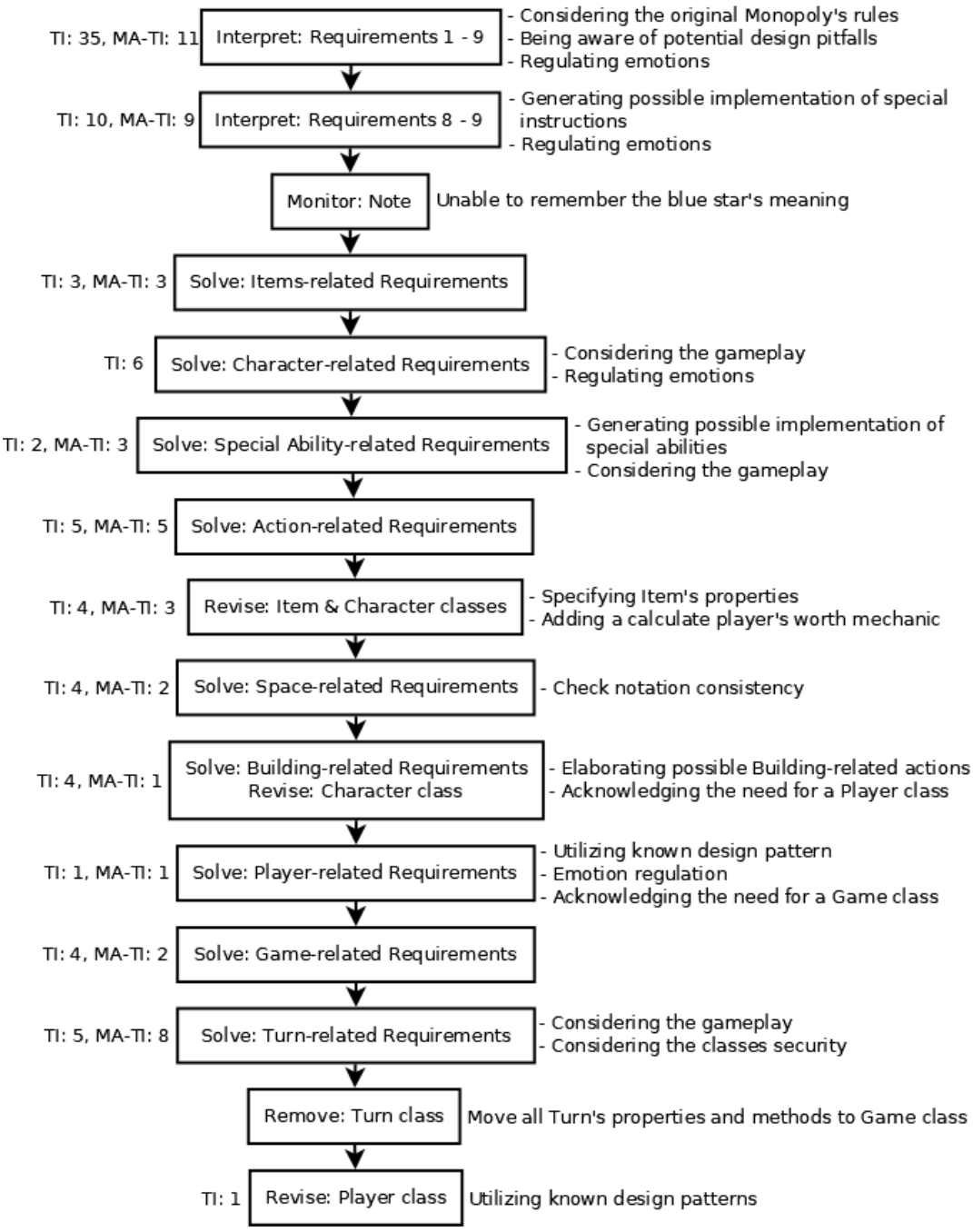



*Figure 5-4. LStew's approach for the third problem.*

LStew was observed engaging in monitoring and adjusting activities throughout her problem-solving endeavor, and as presented in Figure 5-4, 33.10% of those activities

were related to task interpretation. In more specific, these MA-TI activities were about remembering the requirements, associating her understanding of the problem with known concepts, clarifying problem scope, confirming her interpretation by rereading the problem description, interpolating her interpretation, and adding creativity to the design. All except the first two issues resulted in a revised task interpretation. As an example, after generating various possible implementations of special abilities, she assessed whether these possibilities corresponded the nature of board games; she said, “Okay I am going to take a step back and think about if I was playing this game as an actual board game, what would I do with the king?” (i.e., monitoring activity).

LStew was also observed self-regulating her emotion throughout the problem-solving endeavor. For example, after applying the singleton pattern to the Board and Game classes, she said, “That makes me feel a little better, knowing that I have got some patterns I can use …” (i.e., monitoring emotion). Singleton pattern is an object-oriented design technique to ensures that a class (i.e., blueprint) can only have one instance (i.e., product) at a time (Freeman et al., 2004; TechTarget, n.d.). It was important to note that utilizing various design pattern was part of LStew’s problem-solving approach which also improved the design clarity.

By comparing LStew’s final design against the problem-space map, there were some missing design details including the classes’ properties and methods’ access level. Further, there were some design issues that she thoughtfully considered and solved but not written including the mechanics to store building’s owner and identify the player’s

location on the board. Therefore, this finding suggests that LStew's interpretation was incomplete.

## Participants' Self-Regulation in Solving the Fifth Problem

The fifth problem was the Last Standing Man. This well-structured problem asked the participants to write pseudocode (i.e., non-specific programming language) that simulated each step in given procedure to determine the last standing man. The problem provided detailed requirements and constraints including at least five issues, one function, and 4 to 41 variables within a dynamic subsystem. Under the Bloom's Taxonomy described in Gronlund et al. (2013), this problem is at level 6.3 which is creating a product for a specific purpose. Gronlund et al. (2013) subcategorize level 6 Bloom's Taxonomy (i.e., creating) into three, which are generating/hypothesizing, planning/designing, and producing/constructing. It was necessary to a least know basic programming to answer this question. Chapter IV presents a detailed discussion of this programming problem. In this section, the participants' approach to solving the fifth problem is described, including their initial task interpretation (i.e., prior to solving the problem), problem-solving approach, and self-regulation activities.

### Jake's Self-Regulation in the Fifth Problem

**Initial Task Interpretation.** Jake described the goal of this problem as "find*[ing]* the position that will remain the longest in a circle of 3 to 40 people." He was aware that he needed to have the competency in the art of "making algorithms out of behaviors" and basic programming knowledge to answer this problem. Jake's understanding of the task goal was incomplete because the problems asked him to simulate given procedure and

print out the program's state every time a rebel die. Since task interpretation is the "critical first step in SRL" (Butler & Cartier, 2005, p.3). Butler (1995) argues incorrect task interpretation may lead learners to select and employ ineffective strategies to complete the task. Thus, Jake's incomplete task interpretation might influence him to choose wrong strategies.

Jake recognized that he did not need to consider the program's speed or memory used while designing the solution because it would be in pseudocode. However, his solution needed to be mathematically correct. He also mentioned that he had "the exact question in Discrete Mathematics" (MATH3000). The BCM describes that learners' self-regulation, including task interpretation, is bounded within multiple layers of context and one of those contexts was related to learners' experience (Butler & Cartier, 2004a, 2005; Butler et al., 2015; Cartier & Butler, 2004). Studies reported that students tend to start solving a problem intuitively and after that, they work interactively and analytically (Abdillah, Nusantara, Subanj, Susanto, & Abadyo, 2016; Ball, Ormerod, & Morley, 2004; Kahneman, 2003), including when interpreting a problem. Therefore, it was plausible that Jake's incomplete understanding was influenced by his experience in this Discrete Mathematics course.

As part of his implicit understanding of this problem, Jake described three steps to solve it. First, he needed to try a few examples with inputs of three to eight people to determine a pattern. Based on Jake's understanding of the problem, the pattern refers to parts of the algorithm or formula for solving the problem. Since the pseudocode's behavior was given in the problem description and it was necessary to simulate that

behavior as described, finding a pattern was an unnecessary problem-solving step which was influenced by Jake's incomplete task interpretation. Second, he needed to computationally model the pattern, such as using Array or modulus operation. Third, he needed to assess "other ideas that occurred" during the problem-solving endeavor.

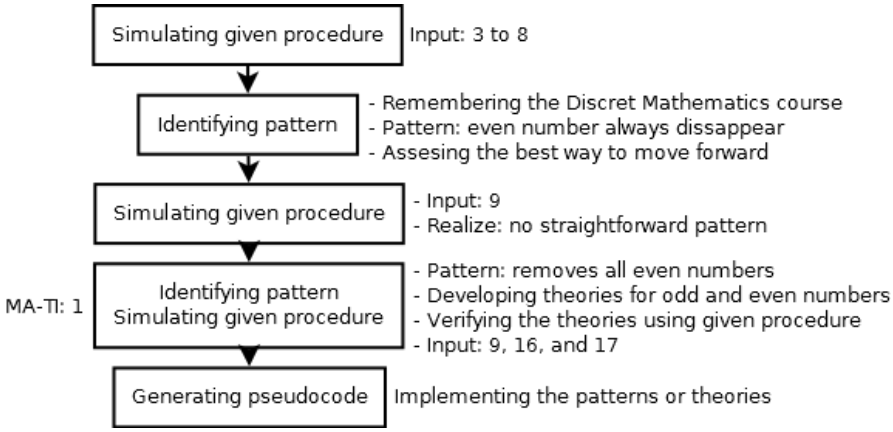



*Figure 5-5. Jake's approach for the fifth problem.*

**Problem-Solving Approach.** As presented in Figure 5-5, Jake's approach to solving the problem was aligned with his initial problem-solving steps (i.e., the paragraph above) to some extent. He began by simulating given procedure using all numbers between three to eight, inclusive as inputs. He then contemplated on the simulation results and tried to identify emerging patterns but came out with none. He then conducted another simulation with nine as input and realized that he would not get a straightforward pattern due to the nature of the problem. He then continued simulating and identifying patterns by considering odd and even numbers until he recognized useful patterns. In other words, Jake was iteratively enacting the first, second, and third problem-solving steps until he found the patterns. He then wrote the pseudocode and answered the problem; this activity was not elicited in his problem-solving step.

During his problem-solving endeavor, Jake was observed verbalizing 56 instances of self-regulation activities including four planning strategies, 42 enacting strategies, 69 monitoring activities, and seven adjustment strategies. Butler & Cartier (2005) argues each strategic action (i.e., planning, enacting, monitoring, and adjusting) starts with task interpretation, and any monitoring activities on task interpretation can result in a revised understanding of the problem. The researcher found all Jake's observed planning and enacting strategies were aligned either with his initial understanding of the problem or observed monitoring and adjusting activities on his understanding of the problem. For example, most of Jake's enacting strategies were related to identifying a working pattern and were informed by his initial task interpretation. Since this study focus on task interpretation and all Jake's other observed strategic actions were sequels of his task interpretation, focusing further analysis on his observed TI and MA-TI would be sufficient to answer the research questions.

As presented in Figure 5-5, Jake's MA-TI only occurred once during the problem-solving endeavor, and it was related to remembering the problem scope. He said, "So thankfully, I do not have to prove this *[pattern]* mathematically" (i.e., monitoring activity). This finding suggests that Jake did not change his task interpretation while solving the problem, and it was confirmed during the interview. Therefore, Jake's final understanding of the problem was still incomplete.

**Rusty's Self-Regulation in the Fifth Problem**

**Initial Task Interpretation.** Rusty described the goal of this problem as "to determine where in the circle Josephus should be *[which position]* in order to be the last

man standing." Rusty was aware that he needed basic programming knowledge, especially the comprehension of "Arrays with conditional operators and if statements." Based on this description, it was clear that Rusty's understanding of the task goal was incomplete because the problems asked him to simulate given procedure and print out the program's state every time a rebel die. Since task interpretation is the "critical first step in SRL" (Butler & Cartier, 2005, p.3). Butler (1995) argues incorrect task interpretation may lead learners to select and employ ineffective strategies to complete the task. Thus, Rusty's incomplete task interpretation might influence him to choose wrong strategies.

Rusty recognized that the requirement and constraint of this problem were "the algorithm must return the correct position, and the chosen number cannot die" respectively. He then explained this was his first time working on such a problem. However, Rusty clarified during the interview that he had solved similar problems in the Discrete Mathematics course, suggesting he could not make an immediate conscious connection between these two during the initial task interpretation.

As part of his implicit understanding of this problem, Rusty described two steps to solve it. First, he needed to "do a few examples by hand, given certain inputs … and look for common patterns that might show up." He also specified that he was interested in examining "odd groups and even groups, as well as large and small inputs." Based on Rusty's understanding of the problem, the pattern refers to parts of the algorithm or formula for solving the problem. Since the pseudocode's behavior was given in the problem description and it was necessary to simulate that behavior as described, finding a pattern was an unnecessary problem-solving step which was influenced by Rusty's

incomplete task interpretation. Second, assuming he found the pattern, he needed to "abstract it and put it into code."

**Problem-Solving Approach.** As presented in Figure 5-6, Rusty's approach to solving the problem was slightly different from his initial problem-solving approach (i.e., the paragraph above). Rusty began by simulating given procedure using odd and even numbers and then contemplated on the outputs, trying to identify emerging patterns. Although he found promising patterns, he decided to reread the problem description and realized that he misinterpreted the task. Following his revised task interpretation, Rusty converted the given procedure into pseudocode. In other words, Rusty was enacting his first problem-solving step until he realized his misunderstanding of the problem goal.

When solving this problem, Rusty was observed verbalizing 180 instances of self-regulation activities including 13 planning strategies, 29 enacting strategies, 131 monitoring activities, and seven adjustment strategies. Butler & Cartier (2005) argues each strategic action (i.e., planning, enacting, monitoring, and adjusting) starts with task interpretation, and any monitoring activities on task interpretation can result in a revised understanding of the problem. The researcher found all Rusty's observed planning and enacting strategies were aligned either with his initial understanding of the problem or observed monitoring and adjusting activities on his understanding of the problem. For example, most of Rusty's enacting strategies were related to identifying a working pattern and were informed by his initial task interpretation. Since this study focus on task interpretation and all Rusty's other observed strategic actions were sequels of his task

interpretation, focusing further analysis on his observed TI and MA-TI would be sufficient to answer the research questions.

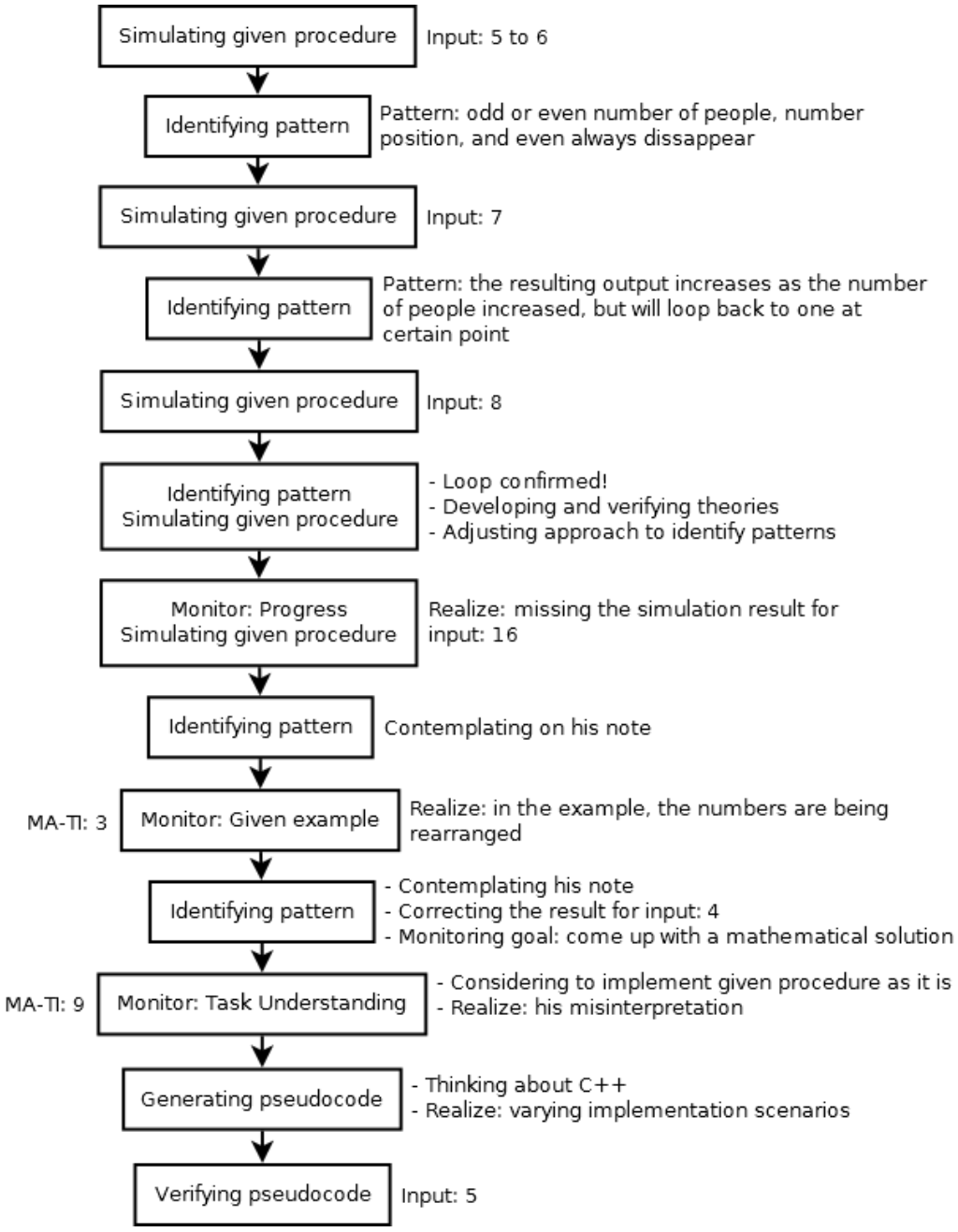



*Figure 5-6. Rusty's approach for the fifth problem.*

As presented in Figure 5-6, there were only twelve observed MA-TI instances and some of those were related to his revised task interpretation. During that critical time, Rusty said:

> "*[Reading the problem description]* You have to simulate each step... Oh my, gosh, I did not read that part thoroughly. *[Reading the problem description]* You have to simulate each step and then determine Josephus' position. Yes, so I was way overthinking it, *[put more emphasis in his voice]* way overthinking it" (i.e., monitoring activity).

During the interview, Rusty was asked to explain the trigger that encouraged him to reread the problem description. Rusty responded:

> "I kind of hit a cycle and I kept looping back to that *[mathematical model of the pattern],* and I was like, okay, something is wrong, I am either not getting something, or there is something obvious that I am skipping over. … but it was not until I felt I had exhausted all my resources, best guesses, and ideas…"

Rusty elaborated that he might be "a little bit overconfidence in thinking that I understood the problem," especially since he had "past experience with the problem that I thought was similar but turned out to be very different."

During the interview, Rusty was asked to elaborate on his next approach under the assumptions that he did not change his task interpretation, and could not found any pattern. Rusty responded, "There always a pattern. Sometimes it is not super obvious," and then elaborated, "Well, if they *[educators]* are asking this question, there has got to be a systematic way to approach it; there has got to be some underlying pattern." His

responses suggest that in an educational setting, all tasks have answers and can be solved using the typical problem-solving approaches related to that type of the tasks.

Although Rusty revised his task interpretation, his final solution was still incomplete, in such that his pseudocode was not designed to display each program state. Nevertheless, the researcher believes that Rusty had a correct task interpretation because, during the interview, Rusty said that the problem description provided "a possible visualization of what it was doing." Furthermore, he shared that developing a simulation program with a specific output format was "something that I have done before with other programming assignments."

**Anne's Self-Regulation in the Third Problem**

**Initial Task Interpretation.** Anne described the goal of this problem as "determine*[ing the]* last surviving space." She was aware that she needed to have a competency in the art of "creative problem solving" and although not mentioned explicitly, she understood that having basic programming knowledge was necessary to answer this problem. Anne's understanding of the task goal was incomplete because the problems asked her to simulate given procedure and print out the program's state every time a rebel die. Since task interpretation is the "critical first step in SRL" (Butler & Cartier, 2005, p.3). Butler (1995) argues incorrect task interpretation may lead learners to select and employ ineffective strategies to complete the task. Thus, Anne's incomplete task interpretation might influence her to choose wrong strategies.

Anne recognized that the "n *[number of people]* is given with function call" suggesting that she understood that her pseudocode should correctly handle any given

inputs as described in the problem description (i.e., three to forty). She also mentioned that she had worked on the "math proof of this problem *[but]* with a twist" in Discrete Mathematics course, suggesting that she was aware of the similarities and differences between these problems.

As part of her implicit understanding of this problem, Anne described three steps to solve it. First, she needed to try few examples "by hand until a pattern is detected." Based on Anne's understanding of the problem, the pattern refers to parts of the algorithm or formula for solving the problem. Since the pseudocode's behavior was given in the problem description and it was necessary to simulate that behavior as described, finding a pattern was an unnecessary problem-solving step which was influenced by Anne's incomplete task interpretation. Second, assuming she found the pattern, Anne needed to "program the solution." Third, she needed to inspect the program's logic including for "simplification or edge cases."

**Problem-Solving Approach.** As presented in Figure 5-7, Anne's approach to solving the problem was slightly aligned with her initial problem-solving steps (i.e., the paragraph above). She began by monitoring the problem goal and constraints; this activity was not elicited in her problem-solving step. Anne continued by simulating given procedure using all numbers between three to seven, inclusive as inputs and contemplated on the results. She noticed an unlikely pattern and then realized that she was not following given procedure correctly. Anne repeated the simulation and contemplated, trying to identify a pattern. While contemplating, Anne had an epiphany that she could exploit the problem constraints and created pseudocode without having to determine any

pattern. She said, “Since it *[the input]* is between 3 and 40, I would just program out each one *[input]* and have it in an Array and then return F *[associated output]*,” and then implemented this alternative solution. In other words, Anne only enacted her first problem-solving step and adjusted the rest.

During her problem-solving endeavor, Anne was observed verbalizing 92 instances of self-regulation activities including 20 enacting strategies, 65 monitoring activities, and seven adjustment strategies. Butler & Cartier (2005) argues each strategic action (i.e., planning, enacting, monitoring, and adjusting) starts with task interpretation, and any monitoring activities on task interpretation can result in a revised understanding of the problem. The researcher found all Anne’s observed planning and enacting strategies were aligned either with her initial understanding of the problem or observed monitoring and adjusting activities on her understanding of the problem. For example, earlier Anne’s enacting strategies were related to identifying a working pattern and were informed by her initial task interpretation. Since this study focus on task interpretation and all Anne’s other observed strategic actions were sequels of her task interpretation, focusing further analysis on her observed TI and MA-TI would be sufficient to answer the research questions.

As presented in Figure 5-7, Anne’s MA-TI only occurred at the beginning of the problem-solving endeavor, and they were related to remembering the problem scope and assessing whether she misunderstood given procedure’s behavior; both activities did not alter her task interpretation. Therefore, Anne’s final understanding of the problem was still incomplete.

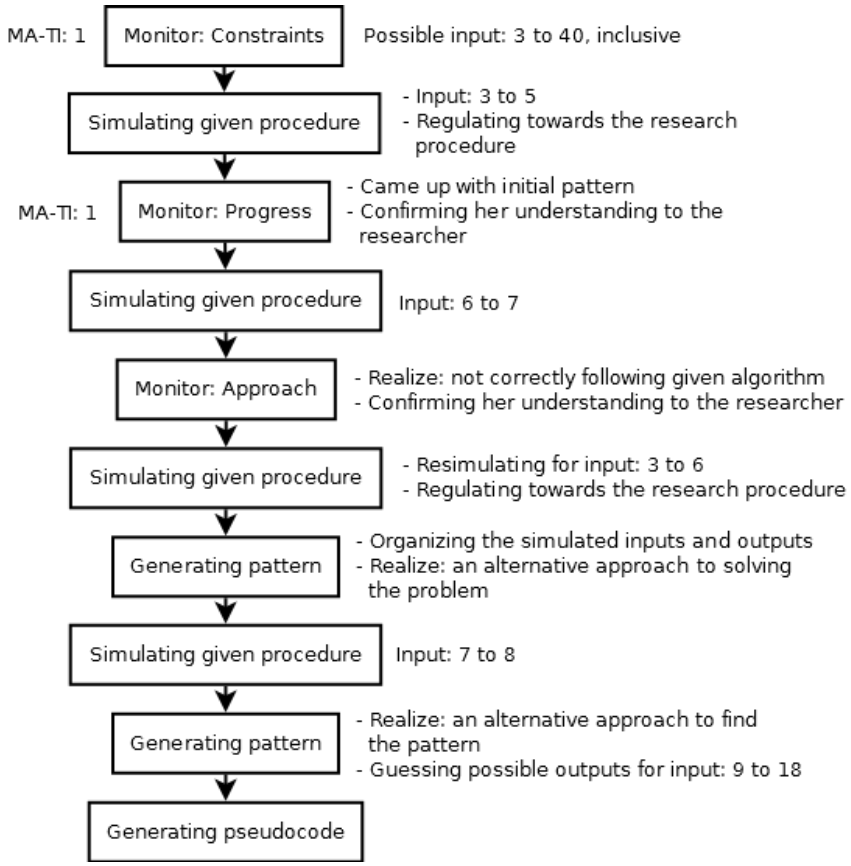



*Figure 5-7. Anne's approach for the fifth problem.*

Anne was also observed initiating a discussion with the researcher about her interpretations and approaches, which suggests that she often worked in a pair or a group and that the research setting might negatively affect her problem-solving process. When being asked about that during the interview, she shared that she had a good friend and they often worked together in various courses. However, Anne's behavior (i.e., initiating a discussion with the researcher) does not suggest a lack of self-efficacy for solving the problem or over-reliance on teamwork. During the last interview, Anne shared that she participated in a team programming contest and was on the top 15$^{th}$ out of 200 teams, suggesting an exceptional self-efficacy on her programming skills. Further, Anne was

participated alone, which suggests she had outstanding self-reliance. Thus, Anne's behavior (i.e., initiating a discussion with the researcher) demonstrated her competency in using various coregulation skills. Coregulation is a transitional process in which the learners define and update their self-regulation skills for solving a problem through interaction with peers (Hadwin, Jarvela, & Miller, 2011; Rivera-Reyes, Lawanto, & Pate, 2016).

Anne first discussion with the researcher was confirming whether her simulation results were correct. In response and due to surprise, the researcher confirmed the correctness of her results. After that, Anne noticed mistakes in her simulation results because she did not accurately follow the given procedure. She then reinterpreted the given procedure and tried to confirm the new interpretation. She said, "Are you allowed to tell me that *[her new interpretation]* is right or do I just have to stay here and bang my head against the wall?" The researcher responded that he could not answer that question, and Anne said, "Oh, no! Oh, wow!" It was clear that she was frustrated and surprised by the researcher's response. Although Anne continued trying to find a working pattern, she changed her approach to solving the problem at some point. During the interview, Anne explained that she was unsure whether she could find the pattern, especially since she made a mistake in following the given procedure.

During the interview, Anne was asked about the last instruction in the problem description, which was "You have to simulate each step and then determine Josephus' position. For example: …." She said, "That means this is supposed to be the printed out *[program output]*. If I call the function with five, then this should be the output of the

function." Anne elaborated that "I read through this instruction, but I did not remember it. I guess I got too caught up in solving it and forgot how the actual output looks like." Her statements suggest that given enough time and different settings, Anne would be able to interpret the problem correctly.

**LStew's Self-Regulation in the Fifth Problem**

**Initial Task Interpretation.** LStew described the goal of this problem as "to write pseudocode that figures out what position Josephus should be at in order to survive." She was aware that a competency in "making algorithms out of behaviors" and basic programming knowledge were necessary to answer this problem. LStew's understanding of the task goal was incomplete because the problems asked her to simulate given procedure and print out the program's state every time a rebel die. Since task interpretation is the "critical first step in SRL" (Butler & Cartier, 2005, p.3). Butler (1995) argues incorrect task interpretation may lead learners to select and employ ineffective strategies to complete the task. Thus, LStew's incomplete task interpretation might influence her to choose wrong strategies.

LStew recognized that "there will never be an input of zero or one because then the problem would not exist" and that program needed to "take an input, run through the formula, and return the output." She also mentioned that she had solved the exact question in the Discrete Mathematics final examination. Since learners' self-regulation is bounded within multiple layers of context, such as learners' experience (Butler & Cartier, 2004a, 2005; Butler et al., 2015; Cartier & Butler, 2004), and that students tend to start solving a problem intuitively (Abdillah et al., 2016; Ball et al., 2004; Kahneman, 2003),

including when interpreting a problem, it was plausible that LStew's incomplete understanding was influenced by her experience in this Discrete Mathematics course.

When describing the steps to solve the problem, LStew restated the program's behavior, which was to read given input, run given input through the formula, and print out the result, suggesting that it was important to remember the overall program flow when designing the solution. She then mentioned, "The formula is probably based on whether or not the number of people in the circle is even or odd" suggesting that finding an appropriate formula would be her problem-solving goal.

**Problem-Solving Approach.** As presented in Figure 5-8, LStew's began by monitoring the given procedure's behavior. She then simulated the given procedure, contemplated on the output, identified emerging patterns, and verified the accuracy of simulated outputs; she repeated this process until the end of her problem-solving endeavor. Unfortunately, LStew was unable to solve the problem. Additionally, LStew was observed expressing her frustration by frequently saying "I am so close!" throughout the problem-solving endeavor.

When solving this problem, LStew was observed verbalizing 338 instances of self-regulation activities including four task interpretation, 11 planning strategies, 36 enacting strategies, 277 monitoring activities, and 10 adjustment strategies. Butler & Cartier (2005) argues each strategic action (i.e., planning, enacting, monitoring, and adjusting) starts with task interpretation, and any monitoring activities on task interpretation can result in a revised understanding of the problem. The researcher found all LStew's observed planning and enacting strategies were aligned either with her initial

understanding of the problem or observed monitoring and adjusting activities on her understanding of the problem. For example, most of LStew's enacting strategies were related to identifying a working pattern and were informed by her initial task interpretation. Since this study focus on task interpretation and all LStew's other observed strategic actions were sequels of her task interpretation, focusing further analysis on her observed TI and MA-TI would be sufficient to answer the research questions.

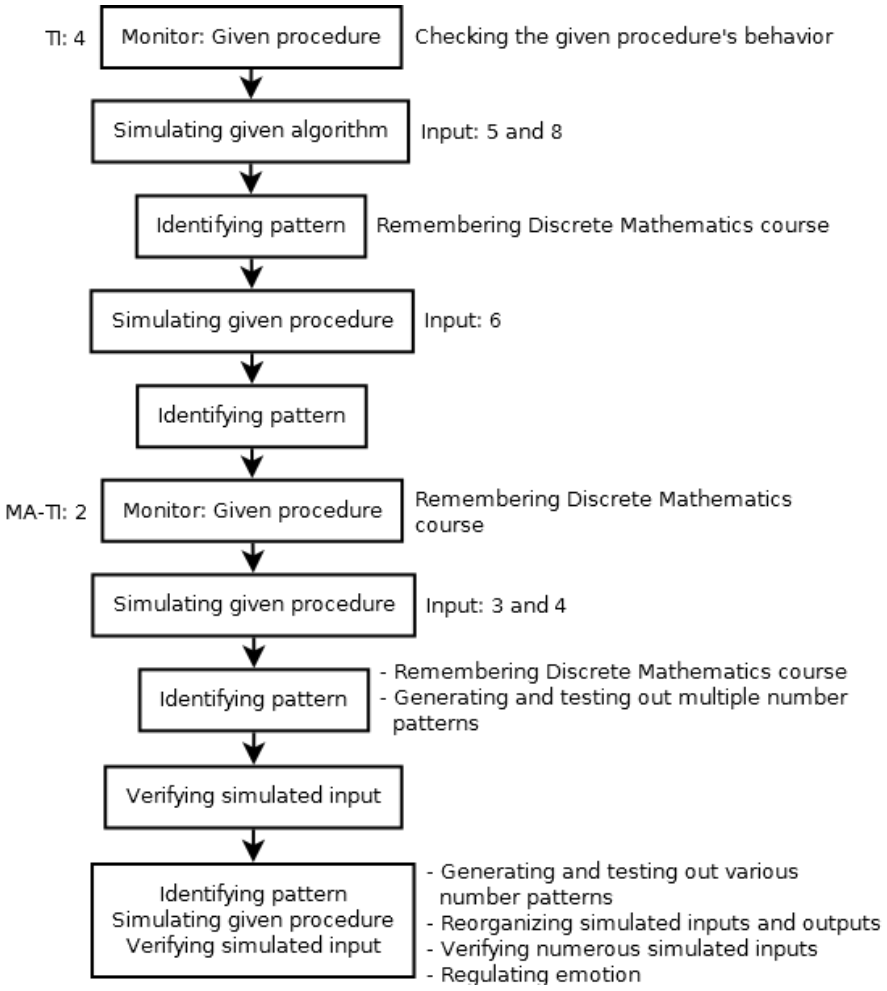



*Figure 5-8. LStew's approach for the fifth problem.*

As presented in Figure 5-8, there were only four and two instances of TI and MA-TI respectively, and all observed engagements did not alter her task interpretation. Therefore, LStew's final task interpretation was still incomplete. Further, LStew inability to solve this problem might be explained by her lack of monitoring activities on task interpretation. Schoenfeld (1983) argues that inadequate self-regulation activities may result in a fail problem-solving attempt.

## Addressing the Research Questions

In this section, the answer for each research question is presented by integrating all participants' initial task interpretation and problem-solving approach (i.e., the discussion in the Participants' Self-Regulation in Solving the Third Problem and Participants' Self-Regulation in Solving the Fifth Problem sections). Since there were two units of analysis, which were the third and fifth problem, the discussion for each question is grouped by these units.

### Research Question 1: What was the students' initial task interpretation of the given problems?

**The Third Problem.** Jake, Rusty, Anne, and LStew were able to correctly identify the explicit aspect of the third problem including determining the problem goal and provided requirements and constraints. Due to its size (i.e., had at least 18 issues, 24 functions, and 22 variables), the participants could not remember all the requirements and constraints. When interpreting the implicit aspect of the task, all participants were able to draw relevant experience from their programming courses. Jake also considered his refactoring experience during his internships as relevant. All participants correctly

understood that having object-oriented design and basic programming skills were essential to solving this problem. When describing their problem-solving steps, all participants expressed that they would iteratively solve the problem, either by going through the identified issues or listed requirements, while continuously optimizing (e.g., restructuring the classes or utilizing known design patterns), adding creativity, and aligning the design to comply with the requirements. This finding supports Felder & Soloman (n.d.)'s report that computer science students like to work linearly, handle facts and details, and monitor their progress periodically.

When interpreting the task, Jake, Rusty, and LStew also considered software design best practices related to easing the software maintenance, software usability, and design clarity. There were no notable differences between male and female or higher- and lower-performance participants' initial task interpretation. However, Jake's interest in probabilistic affected his task interpretation, especially related to the dice's behavior.

**The Fifth Problem.** Jake, Rusty, Anne, and LStew were unable to correctly identify the problem goal, in such they did not recognize that the problems asked them to simulate given procedure and provide a print out of the program's state every time a rebel die. Since all participants mentioned that they had worked on a similar problem in their Discrete Mathematics course, plausibly that experience profoundly influenced their task interpretation. This argument is consistent with SRL theory, which argues that students' experience influence their self-regulation (Butler & Cartier, 2004a, 2005; Butler et al., 2015; Cartier & Butler, 2004) and that students tend to start working intuitively (Abdillah et al., 2016; Ball et al., 2004; Kahneman, 2003). Further, that experience negatively

affected their interpretation of the requirements and constraints and their problem-solving steps. As an example, all participants thought that they needed to identify patterns to solving the problem, which was unnecessary. However, not all of the participants' task interpretations were wrong, for example, LStew correctly interpreted that "there will never be an input of zero or one because then the problem would not exist." This finding suggested having an incorrect task interpretation did not negatively influence other follow-up task understandings. Furthermore, in developing their problem-solving approach, the participants assumed they would be able to identify the patterns, which was worrying because it made them not generating any alternative approach in case something went wrong.

The finding suggested that the participants' incorrect task interpretations were caused by drawing knowledge and strategies from the Discrete Mathematics course. Their misinterpretations were systematic and made most of them oblivious to it. Such phenomenon is commonly known as confidence bias, which is "a systematic error of judgment made by individuals when they assess the correctness of their responses to questions relating to intellectual or perceptual problems" (Pallier et al., 2002, p.258).

There were no notable differences among male and female participants' initial task interpretation. However, a contrast was found between higher- and lower-performers. When interpreting the requirements and constraints of the problem, Rusty and LStew focused on the explicit aspect of the task, while Jake and Anne focused on the implicit aspect. For example, Jake interpreted that speed and memory utilization could be ignored because he only had to create pseudocode.

**Research Question 2: How did their original understanding change during the problem-solving endeavor?**

**The Third Problem.** All participants were continuously refining their understanding throughout the problem-solving process. Rusty's comment during the interview accurately describe this phenomenon:

> "The general understanding did not really change because I knew that I was going to be creating this class diagram, but as far as the *[understanding that affect my]* design decisions, it changed a lot" [Rusty - Third Problem Interview].

On average, each participant was observed verbalizing 41 task interpretation and 37 monitoring and adjusting activities related to their interpretation, which was 21.71% and 17.03% respectively of their total observed strategic actions. The TI and MA-TI percentages are given to provide a better picture of the participants' self-regulation. Although the participants continuously refined their problem understanding, their final task interpretations were still incomplete, suggesting that they were overwhelmed with the detailed of the task. This interpretation was consistent with Butler & Winne (1995)'s argument that being overwhelmed might lower students' self-regulation skills. Further, there were two other identified strategies that partially contributed to the participants' incomplete task understanding, which were selecting inappropriate modeling language and limited monitoring strategies.

There were no notable differences among male and female participants, as well as between higher- and lower- performers. However, Anne's revised task understanding was distinct compared to other participants. Most of her revised interpretation of the task were

unrelated to incorporating her creativity into the design. Plausibly, this trend was influenced by her low self-efficacy in creativity.

**The Fifth Problem.** All participants, except Rusty, did not change their task interpretation during the problem-solving endeavor. Each participant on average was observed verbalizing one task interpretation and four monitoring and adjusting activities related to their interpretation, which was 0.55% and 2.32% respectively of their total observed strategic actions, suggesting that they had limited task interpretation-related engagements. The TI and MA-TI percentages are given to provide a better picture of the participants' self-regulation. It was worth noting that the participants' TI and MA-TI engagements in this problem were substantially smaller compared to the third problem. Plausibly, the different problems' characteristics and the participants' familiarity with the fifth problem influenced their engagements.

The participants' final task interpretations were identical to their initial understanding of the problem. This finding supports Falkner et al. (2014)'s report that some students are unable to align their problem-solving goal with the assessment criteria. Rusty was an exception because he was able to gain an accurate understanding of the problem during the problem-solving endeavor. Rusty was observed verbalizing twelve monitoring activities related to his interpretation, which was 8.70% of his total observed strategic actions. Rusty's MA-TI engagements were higher compared to the other participants' average MA-TI activities. Aside from this, no other dissimilarities found among different genders and performance levels.

**Research Question 3: What were the influencing factors for any revisions of their initial task understanding?**

**The First Problem.** Two factors influenced the participants to revise their task understanding. First, they recognized the extensive requirements and could not remember all of those, in such it prompted the participants to reread the problem description as if they understood it for the first time. These phenomena were captured during the qualitative coding (see Qualitative Coding Results section for more detailed discussion). Second, all participants were aware that in designing a system, understanding how the requirements (or the associated classes) work together was critical. During a programming design activity, students need to employ various cognitive skills and consider the interplays of varying levels and types of abstractions (Renumol et al., 2010; Wing, 2008). Recognizing various levels and types of abstractions implies engaging in a structured problem decomposition, which according to students, is one of the critical computer science skill that is hard to master (Falkner et al., 2014).

**The Fifth Problem.** Since Rusty was the only participant who revised his task interpretation, the researcher only used his problem-solving approach to answer this research question. During the interview, Rusty's explained that his problem-solving endeavor was stagnant at a certain point and it alerted him that there was something wrong; he said, "I am either not getting something, or there is something obvious that I am skipping over." Rusty then reread the problem description and adjusted his task interpretation.

Rusty's behavior offers a new light in understanding Carver & Scheier (1990)'s study, in which they argue that when facing an obstacle (e.g., missing information or lengthy process), students will assess their progress and success probability, and adjust their strategies accordingly. Ge, Law, & Huang (2016) postulate that during a problem-solving process, learners work and self-regulated themselves within the problem-space and solutions-space and their self-regulation in these spaces are not the same. Using their theory, it is clear that Carver & Scheier (1990)'s argument is within the solution-space boundary. Rusty's behavior suggests that when facing an obstacle, students may also return to the problem-space, revise their task interpretation, and then adjust their strategies accordingly.

# CHAPTER VI

# DISCUSSION, CONCLUSION, IMPLICATION, AND RECOMMENDATION

## Introduction

In this chapter, the conclusion of the study is discussed, followed by its implication and recommendation for future studies.

## Discussion and Conclusion

The study findings suggest that the participants were cognizant of various programming problems and able to adjust their problem-solving approach accordingly, including when interpreting a task. Furthermore, the findings also reveal the nature of students' explicit and implicit task interpretation and their revision, which will be discussed separately.

The explicit aspect of task interpretation refers to the "information that is overtly presented in task descriptions and discussions" (Hadwin et al., 2009, p.2), including the participants' understanding of the problem goal and provided requirements and constraints. The findings suggest that the participants were competent in identifying the explicit aspect of the problem and integrating their existing knowledge to have a better understanding of the problem. However, the analysis also reveals that their competency deteriorated when they were familiar with the problem and overconfidence with that feeling (i.e., having a confidence bias).

Associating a new task to previously solved problems is a common problem-solving approach and an instance of good self-regulation (Butler & Cartier, 2004a, 2005; Butler et al., 2015; Cartier & Butler, 2004). However, as revealed during their problem-

solving endeavor, the participants' confidence bias prevented them from checking whether the association itself was correct and hindered them to gain an accurate interpretation and solve the problem correctly. This finding supports Rudolph, Niepel, Greiff, Goldhammer, & Kröner (2017)'s study, in which they reported that students' confidence in knowledge acquisition is closely related to their performance.

Out of four participants, Rusty was the only student who defeated his confidence bias when working on the fifth problem. After the problem-solving endeavor, he admitted that he might be "a little bit overconfidence in thinking that I understood the problem." Rusty's awareness on the stagnancy of his problem-solving endeavor, and that he often misses essential small details when interpreting a problem, inspired him to question whether his task understanding was accurate. In their language retrieval study, Miller & Gerci (2014) reported that students display an improved performance after failing to correctly answer one of the retrieval tasks, such that the failure reduces students' overconfidence and helps them to perform better. Thus, it was possible that Rusty's awareness on his tendency to be oblivious to some small essential details in a problem aided him to lower his overconfidence and monitor his task interpretation. Rusty's self-monitoring engagement and triumph in solving the fifth problem also supports Byun & Lee (2014)'s argument in their physics education research, to which they argue that students' learning and problem-solving strategies have a powerful influence to their success, even when compared to the number of problems that they have solved.

The implicit aspect of task interpretation refers to the "information students might be expected to extrapolate beyond the assignment description" (Hadwin et al., 2009, p.2),

including the participants' relevant experience, problem-solving steps, relevant knowledge and skills, and their extrapolated understanding of the problem requirements and constraint. Please note that some requirements and constraints were given explicitly in the description, which entailed they belong to the problem's explicit aspect.

The analysis suggests that the participants could draw relevant experience, consciously and unconsciously. Having relevant experience affects students' self-regulation (Butler & Cartier, 2004a, 2005; Butler et al., 2015; Cartier & Butler, 2004) because it enables them to utilize the associated effective strategies to complete the task. Falkner et al. (2014) argue that employing discipline-specific self-regulation strategies facilitates students to be successful in programming, suggesting the advantage of knowing and applying context-specific strategies. Thus, drawing strategies from irrelevant experience may result in producing an incorrect solution, or an ineffective or a failed problem-solving endeavor. For example, Jake was unable to address several design issues of the third problem due to his decision to utilize the entity-relationship diagram notations instead of the class diagram.

The analysis suggests that the participants were competent in identifying and extrapolating the problem requirements and constraints. The terms identify and extrapolate are used to emphasize that some of the requirements and constraints are presented in the problem description, and the others have to be extrapolated. Further, the term competent does not infer that the participants can determine all requirements and constraints during their initial task interpretation but rather, given enough time, they are able to do so. For example, when interpreting the third problem, the participants could

not mention all given requirements and constraints, but they could figure out most of those during the problem-solving process.

As mentioned in the previous paragraph, the participants were unable to figure out all requirements and constraints of the third problem. The analysis suggests that they were overwhelmed by the extensive amount of detail in the problem, which is consistent with Butler & Winne (1995)'s argument. As observed during the problem-solving process, sometimes being overwhelmed also hindered the participants to write their design ideas and decisions, and thus forgotten, which then made their final task interpretation incomplete.

The analysis suggests that the participants revised their understanding of the problem requirements and constraints during the problem-solving endeavor, only when the problem possessed many facets. During the third problem, for example, the participants reread the problem description as if they were interpreting it for the first time. When interpreting the requirements, the participants did not only consider given information but also integrated various relevant issues, such as software design best practices; such engagement is also known as deep thinking (Fischer & Hommel, 2012; Renesse & DiGrazia, 2018; Wiersema & Licklider, 2009).

The analysis suggests that the participants were proficient in identifying the most appropriate problem-solving steps according to their explicit and other implicit task interpretation. Further, the analysis reveals that the participants' problem-solving steps are informed by their metacognitive knowledge of the typical approach to solving a similar problem, which is consistent with Butler & Cartier (2004b)'s argument.

When describing their approach to solving the fifth problem, all participants were observed assuming that they could identify useful patterns, suggesting that they did not have a complete problem-solving steps, especially in relation to handling unfavorable outcome (i.e., could not find the patterns); it is important to note that it is unnecessary to find any patterns to solving this problem. One participant explained that, "if they *[educators]* are asking this question, there has got to be a systematic way to approach it; there has got to be some underlying pattern," which suggests the participants assumed that their typical problem-solving approaches were suitable to solve similar problems, at least in an educational setting. Consequently, their overconfidence and assumption on the problem-solving approach and the nature of educational tasks respectively, informed their self-regulation. For example, LStew, who failed to answer the fifth problem, was continuously trying to determine a pattern in such that she was reluctant to assess her progress and success probability and adjust her approach accordingly. After reading her report, LStew commented, "If you approach a problem by focusing on your strengths and flattering your ego you can sometimes miss obvious solutions because you were too busy focusing on how great your special skills are." LStew statement aligned with Jake's and Anne's train of thought and suggested high self-efficacy on their competency.

In conclusion, this study found that the participants were aware of various problems' characteristics and able to tailor their approach to solving the problems accordingly, including when interpreting a task. Given adequate time, all participants were competent in identifying the explicit and extrapolating the implicit aspects of the problem. Further, the participants were observed utilizing their existing knowledge to

have a better understanding of the problem. However, their task interpretation competence deteriorated when they were having a confidence bias, overwhelmed, or drawing knowledge from irrelevant experience. During the problem-solving endeavor, the participants tended only to revise their task interpretation when the problem possessed an extensive amount of detail. Last, when formulating their problem-solving approach, the participants tended to assume that they could solve it using existing problem-solving approaches in their arsenal, and thus did not prepare to handle unfavorable outcomes.

It is important to note that this study is not designed to get generalized findings but to capture as much diversity and depth as possible (Creswell, 2012) to elucidate the nature of computer science students' task interpretation. Related to diversity, be advised that this study does not assess students' task interpretation for all types of problems and programming paradigms. However, some of the findings may be transferable to various situations related to programming, software engineering, and general problem-solving.

## Research and Educational Implications

This study has research and educational implications for educational researchers, instructors, teaching assistants, and students in computer science. In this section, the discussion starts by eliciting the research implications, followed by the educational implications.

First, this study describes students' task interpretation and its revision during a programming endeavor and thus contributes to the limited computer science education literature on self-regulation. This study also supports and expands the findings of various self-regulation and problem-solving studies as demonstrated in the previous section.

Second, this study demonstrates that the integration of Butler & Cartier's self-regulated learning framework (Butler & Cartier, 2004a, 2005; Butler et al., 2015; Cartier & Butler, 2004) and Hadwin's task interpretation model (Hadwin et al., 2009) is possible and beneficial in better understanding students' self-regulation. Therefore, the integration of these models can be replicated in other studies.

Third, this study demonstrates the benefit of utilizing multiple assessment tools and considering students' learning episode to understand their self-regulation better, as recommended by Dinsmore et al. (2008) and Butler & Cartier (2005) respectively. Thus, the similar assessment and analysis methods can be replicated in other studies.

Fourth, this study responds to Teague (2009)'s calling that computer science educators "need to delve a little deeper than normal into the person behind the student, in order to determine the barriers … [that] affect their ability to learn to program" (p.178). Teague's calling suggests relying solely on reported learning and problem-solving phenomena are insufficient. Educators need to know more about the students (e.g., beliefs, characters, and experience) to design an effective intervention. For example, learning about Rusty's experience and beliefs shed light on how he was able to overcome his confidence bias.

Fifth, the description of participants' problem-solving endeavor may benefit computer science instructors, teaching assistants, and students by enabling them to reflect on their self-regulation and deepening their appreciation of students' thinking process complexity. Their reflection and appreciation might also enhance their metacognitive and problem-solving skills due to the increase of thinking process awareness.

Sixth, the study found that the male participants reported spending twice as much time to programming compared to the females. Since spending more time could infer gaining more programming experience and developing expertise (Dreyfus, Dreyfus, & Zadeh, 1987), this finding presents a potential gap between male and female students' expertise. Consequently, a follow-up study is needed to assess any contrasts between them. However, at the same time, it might be beneficial to encourage female students to spend more time programming. Studies reported that female computer science students want to use their programming skill to benefit the society (Balcita, Carver, & Soffa, 2002; Graham & Latulipe, 2003), but avoid the asocial-nerdy stereotype at the same time (Graham & Latulipe, 2003). Thus, computer science educators could offer more authentic and impactful projects in their courses by attracting clients from the community or industry to attract female students to engage more in programming. Educators could also form a female-friendly community in their institution similar to the Women Association for Computer Machinery (W-ACM) mentioned by Anne. Further, educators could utilize pair-programming and provide more communal environments in various programming activities. In pair-programming, one student will act as the driver (i.e., a programmer) and the other will be the navigator (i.e., a planner and debugger). Numerous studies have reported the benefit of such practice (Lui & Chan, 2006; Umapathy & Ritzhaupt, 2017; Williams et al., 2010).

Seventh, Anne was observed initiating discussion with the researcher during her problem-solving enterprise. While some students may be reluctant to seek assistance despite their learning difficulties (Dillon, 1988), Anne's behavior indicates good self-

regulation and perhaps, her competency in utilizing various co-regulation skills. Newman & Schwager (1995) argue asking for hints, similar to Anne's behavior, suggests "students' desire to try to work things out on their own as much as possible" (p. 369). Thus, computer science educators should learn and understand varying and distinct students' needs and avoid associating negative judgment with it. Further, educators should build a learning environment that may support those needs. For example, by developing a learning community or utilizing pair-programming.

Eighth, the findings suggest that all participants were cognizant of various problem types and were able to adjust their approach accordingly. This finding demonstrates that the participants possessed some attributes of expert problem-solvers (Glaser, 1992; Hoffman, 1996). However, it was unclear when the participants started to develop these skills, and thus granting a chance for a potential follow-up study. On the other hand, considering the importance of such skills, it might be beneficial to train students to identify problem characteristics as early as possible (e.g., during their first-year or K12 education). For example, the instructor could ask students to identify the number of issues, variables, or functions presented in the problems. The instructor could also challenge the students to categorize the problems based on its type (see Jonassen (2000, 2004, 2010) for a detailed discussion of various problem types) or Bloom's Taxonomy.

Ninth, the findings suggest that all participants revised their task interpretation during the problem-solving enterprise, especially when the problem was complex and had extensive requirements or constraints. Thus, by enhancing students' ability to identify

problem characteristics might also help them to be more accurate in determining the complexity of a problem, and then improve their awareness of having an incorrect initial task understanding. Further, it might also improve their probability of success in acquiring an accurate task interpretation during their problem-solving endeavor. Computer science educators could help students by familiarizing them with the growth mindset, such as making them aware that their abilities are not fixed but rather changeable given enough time and training (Dweck, 2006). Meanwhile, educators and researchers could design an intervention that may help accelerate students to acquire the accurate task interpretation.

Tenth, Anne's reaction towards creativity-related requirements in the third problem suggests that some students might not be confident with their creativity skill. Although creativity seems can only be assessed through the design artifacts, it is highly related to the design process and metacognitive knowledge (Christiaans & Venselaar, 2005). During their problem-solving enterprise, Jake, Rusty, and LStew were observed addressing creativity by tapping into their interests, preferences, experiences, and known best practices. Thus, computer science educators could encourage students to be more aware of their creative potential, and also encourage them to utilize it when solving course assignments. At the same time, educators could expose students to various creative products in computer science and give students a chance to learn from those.

Eleventh, this study identifies that being overwhelmed was one of the causes preventing students from self-regulating themselves properly. This phenomenon was evident in the third problem. Computer science educators might use this information to

encourage students to work on a complex problem in multiple stages. To make students aware of its benefit, educators might design a classroom design activity where the students tackle the same design problem for multiple days and reflect on their improvement each day.

Twelfth, the participants' problem-solving endeavors for the fifth problem suggest that overcoming confidence bias was not an easy task. Fischoff (1982)'s report suggests that providing external motivations has a meager impact on students' bias. On the other hand, Gigerenzer (1991) argues training students to distinguish single- and frequent-event confidences could lower their tendency to make biased decisions. However, this argument is not applicable in this study because, in the Discrete Mathematics course, students are frequently asked to analyze a set of numbers and develop a formula to generate the exact set. In this study, Rusty's experience suggests that being aware of the problem-solving stagnancy and that one might sometimes miss essential small details, could help overcoming the confidence bias. Thus, computer science educators might design a case study that could draw students' confidence bias, then help them to reflect on that and other occasions where their confidence bias occurs. Educators could also create a video of a biased-actor working on a problem while thinking aloud, present it in the class, ask the students to identify the actor's mistakes, and discuss their responses.

Thirteenth, this study reveals that the participants do not have a complete problem-solving approach for the fifth question. Rusty's explanation suggests that he had a biased perception about assignments in academic settings. Saulnier & Brisson (2018) also reported a similar finding in their study of using impactful and authentic problems in

course assignments. In their study of students' beliefs, McNeill, Douglas, Koro-Ljungberg, Therriault, & Krause (2016) reported that students expect course assignments to be more simple and straightforward compared to any real-world design tasks. Thus, these reports suggest a gap between students' perception of classroom and work field tasks, and that students might need more training in handling real-world design problems. Computer science educators could help by introducing more authentic design problems in the classroom and advising students to develop a versatile plan to solve it.

Fourteenth, the analysis reveals that some participants were self-regulating their emotion during the problem-solving enterprise. Thus, computer science educators could expose students to various emotion regulation strategies and help enhance that competency as early as possible.

## Recommendation for Future Studies

The researcher recommends other educational investigators to conduct direct or conceptual replication studies. As argued by Maksel & Plucker (2014) and Benson & Borrego (2015), replication studies are needed to verify whether particular educational findings are applicable in different settings. Such verifications could help to dismiss educational practitioners' and policies makers' doubts of the educational research results.

When future investigators conducting a replication study, the researcher advises them to utilize the verbal protocol or semi-structured interview for assessing students' initial task interpretation because the collected initial task interpretation survey responses in this study typically lack context and are sometimes hard to interpret. The investigators should also schedule their data collection and analysis cautiously when having more than

one unit of analysis because shifting between multiple analysis units is not easy and may disrupt the analysis process. During the coding process, it is critical to have at least two coders that have considerable experience in the research setting (i.e., computer programming) and are familiar with self-regulated learning theory because they will be proficient in identifying students' learning episodes and deducing students' intentions in each learning episode. Conducting a study in self-regulated learning requires a lot of self-regulation to understand students' behavior. The researcher found that having a discussion partner is beneficial, and suggests future investigators have at least one discussion partner.

The researcher realized there is a need for a systematic literature review to capture current knowledge on students' self-regulation in programming, and to reframe existing problem-solving, cognitive, and metacognitive studies related to computer programming using the self-regulated learning framework. A follow-up investigation can be directed to verify whether the reframed findings hold true.

The researcher also identifies seven possible follow-up educational investigations. First, this study describes how the participants' metacognitive knowledge inform their task interpretation and problem-solving approach. It will be beneficial to investigate the nature of students' metacognitive knowledge of typical problem-solving approaches and then address its deficiency, if any. Second, this study describes the influence of participants' confidence bias in their problem-solving endeavor. It will be beneficial to investigate the nature of confidence bias in course-related programming assignments and design an intervention to help students to conquer that challenge. Third, since this study

identifies some causes that prevent students from self-regulating themselves properly, a follow-up study designed to overcome these self-regulation challenges will be beneficial. Fourth, Ge et al. (2016) argue that students' self-regulation during a problem-solving endeavor can be categorized by space (i.e., problem- and solution-space) and that students have distinct self-regulation in each space. It would be interesting to assess students' self-regulation in both spaces and see their interplay, and then address its deficiencies if any. Fifth, the findings suggest that the participants displayed experts' behaviors. It will be beneficial to assess how those skills develop throughout their education. Sixth, the researcher observed that male and female students self-regulated themselves differently during the problem-solving process, in such that female students were observed engaging in emotion regulation more frequently compared to the male students. It would be interesting to assess how students' emotion regulation impacts their self-regulation in general while solving programming problems. Seventh, the researchers also observed that male and female spent different amount of time to programming, which might affect their expertise. Thus, a follow-up study to clarify this potential issue is needed.

# References

Office of Analysis, Assessment, and Accreditation, Utah State University. (2016). *Summaries by Department*. Logan, UT, USA.

Office of Analysis, Assessment, and Accreditation, Utah State University. (2017). *Enrollment Summary Fall 2017 Utah State University*. Logan, UT, USA. Retrieved from http://www.usu.edu/aaa/pdf/enroll_sum/Fall17Total.pdf

Abdillah, A., Nusantara, T., Subanj, S., Susanto, H., & Abadyo, A. (2016). The Students Decision Making in Solving Discount Problem. *International Education Studies*, *9*(7), 57. https://doi.org/10.5539/ies.v9n7p57

Ackermann, E. K. (1996). Perspective-Taking and Object Construction: Two Keys to Learning. *Constructionism in Practice: Designing, Thinking, and Learning in a Digital World*, 25–37.

Adams, J. C. (2007). Alice, middle schoolers & the imaginary worlds camps. *ACM SIGCSE Bulletin*, *39*(1), 307. https://doi.org/10.1145/1227504.1227418

Al-mehsin, S. A. (2017). Self-Efficacy and Its Relationship with Social Skills and the Quality of Decision-Making among the Students of Prince Sattam Bin Abdul-Aziz University. *International Education Studies*, *10*(7), 108. https://doi.org/10.5539/ies.v10n7p108

Alexander, P. a., Schallert, D. L., & Reynolds, R. E. (2009). What Is Learning Anyway? A Topographical Perspective Considered. *Educational Psychologist*, *44*(3), 176–192. https://doi.org/10.1080/00461520903029006

Alharbi, A., Henskens, F., & Hannaford, M. (2012). Student-Centered Learning Objects

to Support the Self-Regulated Learning of Computer Science. *Creative Education*, *03*(06), 773–783. https://doi.org/10.4236/ce.2012.326116

AlleyDog.com. (n.d.). Psychology Glossary. Retrieved January 1, 2017, from http://www.alleydog.com/glossary/psychology-glossary.php

Ambrosio, A. P., Almeida, L., Franco, A., Martins, S. W., & Georges, F. (2012). Assessment of self-regulated attitudes and behaviors of introductory programming students. In *Proceedings of the 42nd Frontiers in Education Conference - FIE'12* (pp. 1–6). https://doi.org/10.1109/FIE.2012.6462314

Anderson, J. R. (1980). *Cognitive psychology and its implications*. San Francisco: Freeman. Retrieved from http://www.informationr.net/ir/20-1/isic2/isic24.html#.VepxcRHBwXA

Anderson, J. R., & Jeffries, R. (1985). Novice LISP Errors: Undetected Losses of Information from Working Memory. *Human–Computer Interaction*, *1*(2), 107–131. https://doi.org/10.1207/s15327051hci0102_2

Anderson, J. R., & Skwarecki, E. (1989). The automated tutoring of introductory computer programming. *Communications of the ACM*, *29*(9), 842–849.

Apple Inc. (n.d.). Apple Inc. Official Website. Retrieved January 1, 2017, from http://www.apple.com/

Artino, A., & Stephens, J. (2007). Bored and frustrated with online learning? Understanding achievement emotions from a social cognitive, control-value perspective. In *Annual meeting of the Northeastern Educational Research Association*. Rocky Hill, CT.

Ashcraft, M. H. (1992). Cognitive arithmetic: A review of data and theory. *Cognition*, *44*(1–2), 75–106. https://doi.org/10.1016/0010-0277(92)90051-I

Bainbridge, L., & Sanderson, P. (2005). Verbal Protocol Analysis. In J. R. Wilsom & N. Corlett (Eds.), *Evaluation of Human Work, 3rd Edition* (3rd ed., p. 1048). CRC Press. Retrieved from https://books.google.com/books?id=dSmKYLp82b4C&pgis=1

Balcita, A. M., Carver, D. L., & Soffa, M. Lou. (2002). Shortchanging the Future of Information Technology: the Untapped Resource. *ACM SIGCSE Bulletin*, *34*(2), 32. https://doi.org/10.1145/543812.543825

Ball, L. J., Ormerod, T. C., & Morley, N. J. (2004). Spontaneous analogising in engineering design: a comparative analysis of experts and novices. *Design Studies*, *25*(5), 495–508. https://doi.org/10.1016/j.destud.2004.05.004

Bandura, A. (1977). *Social Learning Theory*. Oxford, UK: Prentice Hall.

Bandura, A. (1986). *Social Foundations of Thought and Action*. Prentice Hall.

Barak, M., Harward, J., Kocur, G., & Lerman, S. (2007). Transforming an Introductory Programming Course: From Lectures to Active Learning via Wireless Laptops. *Journal of Science Education and Technology*, *16*(4), 325–336. https://doi.org/10.1007/s10956-007-9055-5

Basit, T. (2003). Manual or electronic? The role of coding in qualitative data analysis. *Educational Research*, *45*(2), 143–154. https://doi.org/10.1080/0013188032000133548

Beaubouef, T., & Mason, J. (2005). Why the high attrition rate for computer science

students. *ACM SIGCSE Bulletin*, *37*(2), 103. https://doi.org/10.1145/1083431.1083474

Ben-Ari, M. (1998). Constructivism in Computer Science Education. *ACM SIGCSE Bulletin*, *30*(1), 257–261. https://doi.org/10.1145/274790.274308

Benson, L., & Borrego, M. (2015). The Role of Replication in Engineering Education Research. *Journal of Engineering Education*, *104*(4), 388–392. https://doi.org/10.1002/jee.20082

Bergin, S., Reilly, R., & Traynor, D. (2005). Examining the role of self-regulated learning on introductory programming performance. In *First International Workshop on Computing Education Research* (pp. 81–86). New York, New York, USA: ACM Press. https://doi.org/10.1145/1089786.1089794

Berglund, A., Daniels, M., & Pears, A. (2006). Qualitative research projects in computing education research: an overview. In *ACE '06 Proceedings of the 8th Australasian Conference on Computing Education* (pp. 25–33). Australian Computer Society, Inc. Retrieved from http://dl.acm.org/citation.cfm?id=1151875

Biochemical Society. (n.d.). What is biochemistry? Retrieved March 7, 2018, from http://www.biochemistry.org/?TabId=456

Boudreau, T., Tulach, J., & Unger, R. (2006). Decoupled Design: Building Applications on the NetBeans Platform. In *Companion to the 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications - OOPSLA '06* (p. 854). New York, New York, USA: ACM Press. https://doi.org/10.1145/1176617.1176734

Bransford, J. D., Brown, A. L., & Cocking, R. R. (1999). Learning: From Speculation to Science. In *How people learn: Brain, mind, experience, and school*. National Academy Press. Retrieved from http://www.nap.edu/read/9853/chapter/3

Breckner, R. E., Schlottmann, G. A., Beaulieu, N. M., LeMay, S. G., Nelson, D. R., Palchetti, J., & Benbrahim, J. (2001). *US7837556B2*. The United States. Retrieved from https://patents.google.com/patent/US7837556B2/en

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. … *of the 2012 Annual Meeting of* …. Retrieved from http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

Briggs, T. (2005). Techniques for active learning in CS courses. *Journal of Computing Sciences in Colleges*, *21*(2), 156–165. Retrieved from http://dl.acm.org/citation.cfm?id=1089053.1089075

Bruner, J. S. (1966). *Toward a Theory of Instruction*. Harvard University Press. Retrieved from https://books.google.com/books?hl=en&lr=&id=F_d96D9FmbUC&pgis=1

Brydges, R., & Butler, D. (2012). A reflective analysis of medical education research on self-regulation in learning and practice. *Medical Education*, *46*(1), 71–9. https://doi.org/10.1111/j.1365-2923.2011.04100.x

Bughin, J., Chui, M., & Manyika, J. (2010, August). Clouds, Big Data, and Smart Assets: Ten Tech-Enabled Business Trends to Watch. *McKinsey Quarterly*, 1–14.

Bui, Q. (2015). Will Your Job Be Done By A Machine? Retrieved May 25, 2015, from http://www.npr.org/sections/money/2015/05/21/408234543/will-your-job-be-done-by-a-machine

Bundy, A. (2007). Computational Thinking is Pervasive. *Journal of Scientific and Practical Computing*, *1*(2), 67–69. Retrieved from http://homepages.inf.ed.ac.uk/bundy/drafts/Chen_Li_Article.pdf

Burton, L. (2002). Methodology and methods in mathematics education research: Where is "The Why"? In S. Goodchild & L. English (Eds.), *Researching mathematics classrooms: A critical examination of methodology* (pp. 1–10). Westport, CT: Praeger.

Butler, D. L. (1995). Promoting Strategic Learning by Postsecondary Students with Learning Disabilities. *Journal of Learning Disabilities*, *28*(3), 170–190. https://doi.org/10.1177/002221949502800306

Butler, D. L. (1998). Metacognition and Learning Disabilities. In B. Y. L. Wong (Ed.), *Learning About Learning Disabilities* (2nd ed., pp. 277–307). Toronto: Academic Press.

Butler, D. L., & Cartier, S. C. (2004a). Learning in varying activities: An explanatory framework and a new evaluation tool founded on a model of self-regulated learning. In *Annual Conference of the Canadian Society for The Study of Education*. Toronto, ON. Retrieved from http://perso.crifpe.ca/~scartier/spip/IMG/pdf/Butler_and_Cartier_2004_.pdf

Butler, D. L., & Cartier, S. C. (2004b). Promoting Effective Task Interpretation as an Important Work Habit: A Key to Successful Teaching and Learning. *Teachers College Record*, *106*(9), 1729–1758. Retrieved from http://www.sfu.ca/~jcnesbit/EDUC220/ThinkPaper/ButlerCartier2004.pdf

Butler, D. L., & Cartier, S. C. (2005). Multiple Complementary Methods for Understanding Self-Regulated Learning as Situated in Context. In *American Educational Research Association, Annual Meeting* (pp. 11–15).

Butler, D. L., & Cartier, S. C. (2018). Case Studies as a Methodological Framework for Studying and Assessing Self-Regulated Learning. In D. H. Schunk & J. Greene (Eds.), *Handbook of Self-Regulation of Learning and Performance* (2nd ed., pp. 352–369). New York, New York, USA: Routledge.

Butler, D. L., Cartier, S. C., Schnellert, L., Gagnon, F., & Giammarino, M. (2011). Secondary students' self-regulated engagement in reading: researching self-regulation as situated in context. *Psychological Test and Assessment Modeling*, *53*(1), 73–105. Retrieved from https://www.researchgate.net/profile/Sylvie_Cartier/publication/50864533_Secondary_students_self-regulated_engagement_in_reading_researching_self-regulation_as_situated_in_context/links/54eca5a90cf27fbfd7713445.pdf

Butler, D. L., Schnellert, L., & MacNeil, K. (2015). Collaborative inquiry and distributed agency in educational change: A case study of a multi-level community of inquiry. *Journal of Educational Change*, *16*(1), 1–26. https://doi.org/10.1007/s10833-014-9227-z

Butler, D. L., & Winne, P. H. (1995). Feedback and Self-Regulated Learning: A Theoretical Synthesis. *Review of Educational Research*, *65*(3), 245–281. https://doi.org/10.3102/00346543065003245

Byun, T., & Lee, G. (2014). Why students still can't solve physics problems after solving

over 2000 problems. *American Journal of Physics*, *82*(9), 906–913. https://doi.org/10.1119/1.4881606

Carnegie Mellon University. (n.d.). Alice. Retrieved April 19, 2017, from http://www.alice.org/index.php

Carruthers, S., & Stege, U. (2013). On Evaluating Human Problem Solving of Computationally Hard Problems. *The Journal of Problem Solving*, *5*(2), 42–71. https://doi.org/10.7771/1932-6246.1152

Cartier, S. C., & Butler, D. L. (2004). Elaboration and validation of questionnaires and plan for analysis. In *Annual Conference of the Canadian Society for The Study of Education*. Toronto, ON.

Carver, C. S., & Scheier, M. F. (1990). Origins and functions of positive and negative affect: A control-process view. *Psychological Review*, *97*(1), 19–35. https://doi.org/10.1037/0033-295X.97.1.19

Case, J. M., & Light, G. (2011). Emerging Research Methodologies in Engineering Education Research. *Journal of Engineering Education*, *100*(1), 186–210. https://doi.org/10.1002/j.2168-9830.2011.tb00008.x

Cass, S. (2016). The 2016 Top Programming Languages. Retrieved May 6, 2017, from http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages

Chi, M. T. H., De Leeuw, N., Chiu, M.-H., & Lavancher, C. (1994). Eliciting Self-Explanations Improves Understanding. *Cognitive Science*, *18*(3), 439–477. https://doi.org/10.1207/s15516709cog1803_3

Christiaans, H., & Venselaar, K. (2005). Creativity in Design Engineering and the Role

of Knowledge: Modelling the Expert. *International Journal of Technology and Design Education*, *15*(3), 217–236. https://doi.org/10.1007/s10798-004-1904-4

Clance, P. R., & Imes, S. A. (1978). The imposter phenomenon in high achieving women: Dynamics and therapeutic intervention. *Psychotherapy: Theory, Research & Practice*, *15*(3), 241–247. https://doi.org/10.1037/h0086006

Clark, M. (2003). Computer Science: A hard-applied discipline? *Teaching in Higher Education*, *8*(1), 71–87. https://doi.org/10.1080/1356251032000052339

Cleaves, D. A. (1987). Cognitive biases and corrective techniques: proposals for improving elicitation procedures for knowledge-based systems. *International Journal of Man-Machine Studies*, *27*(2), 155–166. https://doi.org/10.1016/S0020-7373(87)80049-4

Cobb, P. (1994). Where Is the Mind? Constructivist and Sociocultural Perspectives on Mathematical Development. *Educational Researcher*, *23*(7), 13–20. https://doi.org/10.3102/0013189X023007013

Computer Hope. (n.d.). Free Computer Help and Information. Retrieved April 22, 2017, from http://www.computerhope.com

Cope-Watson, G., & Betts, A. S. (2010). Confronting otherness: An e- conversation between doctoral students living with the Imposter Syndrome. *Canadian Journal for New Scholars in Education*, *3*(1), 13.

Coutinho, S. A. (2007). The relationship between goals, metacognition, and academic success. *Educate*, *7*(1), 39–47. Retrieved from http://www.educatejournal.org/index.php/educate/issue/view/23

Creswell, J. W. (2012). *Qualitative Inquiry and Research Design: Choosing Among Five Approaches* (3rd ed.). SAGE Publications.

De Vaus, D. (2013). *Surveys in Social Research* (6th ed.). Routledge.

de Vries, M. F. R. K. (1990). The Impostor Syndrome: Developmental and Societal Issues. *Human Relations*, *43*(7), 667–686. https://doi.org/10.1177/001872679004300704

Denning, P. J. (2001). The Profession of IT: Who are We? *Communications of the ACM*, *44*(2), 15–19. https://doi.org/10.1145/359205.359239

Denning, P. J. (2003). Computer Science. In A. Ralston, E. D. Reilly, & D. Hemmendinger (Eds.), *Encyclopedia of Computer Science* (4th ed., pp. 405–419). Chichester, UK: John Wiley and Sons Ltd. Retrieved from http://dl.acm.org/citation.cfm?id=1074266

Denning, P. J. (2004). Great principles in computing curricula. *ACM SIGCSE Bulletin*, *36*(1), 336–341. https://doi.org/10.1145/1028174.971303

Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., & Young, P. R. (1989). Computing as a Discipline. *Communications of the ACM*, *32*(1), 9–23. https://doi.org/10.1145/63238.63239

Denning, P. J., & Freeman, P. A. (2009). The Profession of IT Computing's Paradigm. *Communications of the ACM*, *52*(12), 28. https://doi.org/10.1145/1610252.1610265

Dictionary.com, L. (n.d.). Dictionary.com. Retrieved April 22, 2017, from http://www.dictionary.com

Diethelm, I., Hubwieser, P., & Klaus, R. (2012). Students, teachers and phenomena:

educational reconstruction for computer science education. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research - Koli Calling '12* (pp. 164–173). New York, New York, USA: ACM Press. https://doi.org/10.1145/2401796.2401823

Dijkstra, E. W. (1989). On the cruelty of really teaching Computer Science. *Communications of the ACM*, *32*, 1398–1404.

Dillon, J. T. (1988). The Remedial Status of Student Questioning. *Journal of Curriculum Studies*, *20*(3), 197–210. https://doi.org/10.1080/0022027880200301

Dinsmore, D. L., Alexander, P. A., & Loughlin, S. M. (2008). Focusing the conceptual lens on metacognition, self-regulation, and self-regulated learning. *Educational Psychology Review*, *20*(4), 391–409. https://doi.org/10.1007/s10648-008-9083-6

Dreyfus, H. L., Drey-fus, S. E., & Zadeh, L. A. (1987). Mind over Machine: The Power of Human Intuition and Expertise in the Era of the Computer. *IEEE Expert*, *2*(2), 110–111. https://doi.org/10.1109/MEX.1987.4307079

Dweck, C. S. (2006). *Mindset: The New Psychology of Success*. Random House.

Eden, A. H. (2007). Three Paradigms of Computer Science. *Minds and Machines*, *17*(2), 135–167. https://doi.org/10.1007/s11023-007-9060-8

Engineering Accreditation Commission. (2003). Criteria for accrediting engineering program. *ABET Report E1 11/19*. Baltimore, Md. Retrieved from http://www.abet.org/criteria_eac.html

Engineering Education Department Utah State University. (2016). REU Site Program in Engineering Education. Retrieved August 31, 2016, from http://reu.usu.edu/

Ertmer, P. A., & Newby, T. J. (2013). Behaviorism, Cognitivism, Constructivism: Comparing Critical Features From an Instructional Design Perspective. *Performance Improvement Quarterly*, *26*(2), 43–71. https://doi.org/10.1002/piq.21143

Falkner, K., Szabo, C., Michell, D., Szorenyi, A., & Thyer, S. (2015). Gender Gap in Academia: Perceptions of Female Computer Science Academics. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '15* (pp. 111–116). New York, New York, USA: ACM Press. https://doi.org/10.1145/2729094.2742595

Falkner, K., Vivian, R., & Falkner, N. J. G. (2014). Identifying computer science self-regulated learning strategies. In *Proceedings of the 2014 conference on Innovation & technology in computer science education - ITiCSE '14* (pp. 291–296). New York, New York, USA: ACM Press. https://doi.org/10.1145/2591708.2591715

Febrian, A., Lawanto, O., & Cromwell, M. (2015). Advancing Research on Engineering Design using e-Journal. In *Frontiers in Education Conference (FIE), 2015. 32614 2015. IEEE* (pp. 1–5). El Paso, TX: IEEE. https://doi.org/10.1109/FIE.2015.7344191

Felder, R. M., & Soloman, B. A. (n.d.). Learning Styles and Strategies. Retrieved April 25, 2017, from http://www4.ncsu.edu/unity/lockers/users/f/felder/public//ILSdir/styles.htm

Fischer, R., & Hommel, B. (2012). Deep thinking increases task-set shielding and reduces shifting flexibility in dual-task performance. *Cognition*, *123*(2), 303–307. https://doi.org/10.1016/j.cognition.2011.11.015

Fischoff, B. (1982). Debiasing. In D. Kahneman, P. Slovic, & A. Tversky (Eds.), *Judgment under Uncertainty: Heuristics and Biases* (pp. 422–444). Cambridge, MA, USA: Cambridge University Press.

Fisher, A., & Margolis, J. (2002). Unlocking the clubhouse: the Carnegie Mellon experience. *ACM SIGCSE Bulletin*, *34*(2), 79. https://doi.org/10.1145/543812.543836

Forgas, J. P. (2000). Affect and information processing strategies: An interactive relationship. In *Feeling and thinking: The role of affect in social cognition* (pp. 253–280). Retrieved from http://philpapers.org/rec/FORAAI-2

Fowler, M., & Beck, K. (1999). *Refactoring: Improving the Design of Existing Code* (Illustrate). Addison-Wesley Professional.

Freeman, E., Bates, B., Sierra, K., & Robson, E. (2004). *Head First Design Patterns* (1st ed.). O'Reilly Media.

Froehlich, J., Chen, M. Y., Smith, I. E., & Potter, F. (2006). Voting with Your Feet: An Investigative Study of the Relationship Between Place Visit Behavior and Preference (pp. 333–350). Springer, Berlin, Heidelberg. https://doi.org/10.1007/11853565_20

Gal-Ezer, J., & Harel, D. (1998). What (else) should CS educators know? *Communications of the ACM*, *41*(9), 77–84. https://doi.org/10.1145/285070.285085

Gall, M. D., Gall, J. P., & Borg, W. R. (2007). *Educational Research: An Introduction*. (A. E. Burvikovs, Ed.) (8th ed.). USA: Pearson Education Inc.

Galpin, V. (2002). Women in computing around the world. *ACM SIGCSE Bulletin*, *34*(2),

94. https://doi.org/10.1145/543812.543839

Ge, X., Law, V., & Huang, K. (2016). Detangling the Interrelationships Between Self-Regulation and Ill-Structured Problem Solving in Problem-Based Learning. *Interdisciplinary Journal of Problem-Based Learning*, *10*(2). https://doi.org/10.7771/1541-5015.1622

geeksforgeeks. (n.d.). Geeks for Geek - A Computer Science Portal for Geeks. Retrieved April 2, 2018, from https://www.geeksforgeeks.org/

Geffner, H. (2014). Artificial Intelligence: From programs to solvers. *AI Communications*, *27*(1), 45–51. https://doi.org/10.3233/AIC-130581

Gigerenzer, G. (1991). How to Make Cognitive Illusions Disappear: Beyond "Heuristics and Biases." *European Review of Social Psychology*, *2*(1), 83–115. https://doi.org/10.1080/14792779143000033

Glaser, R. (1992). Expert knowledge and processes of thinking. In D. Halpern (Ed.), *Enhancing thinking skills in the sciences and mathematics* (pp. 63–75). Hillsdale, NJ, USA: Lawrence Erlbaum Associates, Inc.

Glass, R. L. (2006). Call It Problem Solving, Not Computational Thinking. *Communications of the ACM*, *49*(9), 3. Retrieved from http://search.ebscohost.com/login.aspx?direct=true&db=buh&AN=22043623&site=eds-live

Gonzalez, G. (2006). A systematic approach to active and cooperative learning in CS1 and its effects on CS2. In *ACM SIGCSE Bulletin* (Vol. 38, p. 133). New York, New York, USA: ACM Press. https://doi.org/10.1145/1124706.1121386

Google Inc. (n.d.-a). Android. Retrieved February 28, 2018, from https://www.android.com/

Google Inc. (n.d.-b). Google Developers Training - Android. Retrieved February 28, 2018, from https://developers.google.com/training/android/

Google Inc. (n.d.-c). Google Store. Retrieved January 1, 2017, from https://store.google.com/

Graham, S., & Latulipe, C. (2003). CS girls rock: sparking interest in computer science and debunking the stereotypes. *ACM SIGCSE Bulletin*, *35*(1), 322. https://doi.org/10.1145/792548.611998

Gronlund, N. E., Gronlund, E. N., & Waugh, C. K. (2013). *Assessment of Student Achievement*. *Assessment of Student Achievement* (10th ed.). Pearson.

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

Gul, R. B., & Ali, P. A. (2010). Clinical trials: the challenge of recruitment and retention of participants. *Journal of Clinical Nursing*, *19*(1–2), 227–233. https://doi.org/10.1111/j.1365-2702.2009.03041.x

Guzdial, M. (2008). Education Paving the Way for Computational Thinking. *Communications of the ACM*, *51*(8), 25. https://doi.org/10.1145/1378704.1378713

Guzdial, M., Johnson, R., Wampler, K., Kussmaul, C., Swanson, J., Humenn, P., & Lewchuk, M. (2015). What's the best way to teach computer science to beginners? *Communications of the ACM*, *58*(2), 12–13. https://doi.org/10.1145/2714488

Hadwin, A. (2006). Do your students really understand your assignments? *LTC Currents: Optimizing Learning Environments*, *11*(3), 8–9.

Hadwin, A. F., Jarvela, S., & Miller, M. (2011). Self-regulated, co-regulated, and socially shared regulation of learning. In B. J. Zimmerman & D. H. Schunk (Eds.), *Handbook of Self-Regulation of Learning and Performance* (pp. 65–84). New York, New York, USA: Routledge.

Hadwin, A. F., Oshige, M., Miller, M., & Wild, P. (2009). Examining Student and Instructor Task Perceptions in a Complex Engineering Design Task. In *The Sixth International Conference on Innovation and Practices in Engineering Design and Engineering Education*. Hamilton, ON, Canada: McMaster University.

Hagerott, S. G., & LaBanca, J. (2008). *US8924845B2*. The United States. Retrieved from https://patents.google.com/patent/US8924845B2/en

Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., & Hosking, A. L. (2009). A Multidisciplinary Approach Towards Computational Thinking for Science Majors. *ACM SIGCSE Bulletin*, *41*(1), 183. https://doi.org/10.1145/1539024.1508931

Hamp-Lyons, L., & Mathias, S. P. (1994). Examining expert judgments of task difficulty on essay tests. *Journal of Second Language Writing*, *3*(1), 49–68.

Havenga, M. (2015). The Role of Metacognitive Skills in Solving Object-Oriented Programming Problems: a Case Study. *TD: The Journal for Transdisciplinary Research in Southern Africa*, *11*(1), 133–147. Retrieved from https://journals.co.za/content/transd/11/1/EJC175923

Henderson, P. B. (2009). Ubiquitous computational thinking. *Computer*, *42*(10), 100–

102. https://doi.org/10.1109/MC.2009.334

Heppner, P. P., & Krauskopf, C. J. (1987). An information-processing approach to personal problem solving. *The Counseling Psychologist*, *15*, 371–447.

Hoffman, R. R. (1996). How can expertise be defined? Implications of research from cognitive psychology. In R. Williams, W. Faulkner, & J. Fleck (Eds.), *Exploring Expertise* (pp. 81–100). Edinburgh, Scotland: University of Edinburgh Press.

Howles, T. (2007). Preliminary Results of a Longitudinal Study of Computer Science Student Trends, Behaviors and Preferences. *Journal of Computing Sciences in Colleges*, *22*(6), 18–27. Retrieved from http://dl.acm.org/citation.cfm?id=1231097

Huawei Technologies Co., L. (n.d.). Huawei. Retrieved January 1, 2017, from http://consumer.huawei.com/

Institutional Review Board. (2011). *Investigator Handbook* (Version 2.). Logan, UT, USA: Utah State University.

Irani, L. (2004). Understanding gender and confidence in CS course culture. *ACM SIGCSE Bulletin*, *36*(1), 195. https://doi.org/10.1145/1028174.971371

Isomöttönen, V., & Tirronen, V. (2013). Teaching programming by emphasizing self-direction. *ACM Transactions on Computing Education*, *13*(2), 1–21. https://doi.org/10.1145/2483710.2483711

Johnson, S. D. (2008). Cognitive Analysis of Expert and Novice Troubleshooting Performance. *Performance Improvement Quarterly*, *1*(3), 38–54. https://doi.org/10.1111/j.1937-8327.1988.tb00021.x

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. (2004). *Learning to Solve Problems: An Instructional Design Guide*. (M. Davis, Ed.). John Wiley & Sons.

Jonassen, D. H. (2010). *Learning to solve problems: A handbook for designing problem-solving learning environments*. *Learning to Solve Problems: A Handbook for Designing Problem-Solving Learning Environments*. Routledge. https://doi.org/10.4324/9780203847527

Jones, B. F., & Idol, L. (2013). *Dimensions of Thinking and Cognitive Instruction*. Routledge.

Joo, Y.-J., Bong, M., & Choi, H.-J. (2000). Self-efficacy for self-regulated learning, academic self-efficacy, and internet self-efficacy in web-based instruction. *Educational Technology Research and Development*, *48*(2), 5–17. https://doi.org/10.1007/BF02313398

Kahneman, D. (2003). A Perspective on Judgment and Choice: Mapping Bounded Rationality. *American Psychologist*, *58*(9), 697–720. https://doi.org/10.1037/0003-066X.58.9.697

Kim, Y. (2011). The Pilot Study in Qualitative Inquiry: Identifying Issues and Learning Lessons for Culturally Competent Research. *Qualitative Social Work*, *10*(2), 190–206. https://doi.org/10.1177/1473325010362001

Kinnunen, P., & Malmi, L. (2006). Why students drop out CS1 course? In *Proceedings of the 2006 international workshop on Computing education research - ICER '06* (p.

97). New York, New York, USA: ACM Press. https://doi.org/10.1145/1151588.1151604

Ko, A. J., & Davis, K. (2017). Computing Mentorship in a Software Boomtown. In *Proceedings of the 2017 ACM Conference on International Computing Education Research - ICER '17* (pp. 236–244). New York, New York, USA: ACM Press. https://doi.org/10.1145/3105726.3106177

Koo, M., & Skinner, H. (2005). Challenges of Internet Recruitment: A Case Study with Disappointing Results. *Journal of Medical Internet Research*, *7*(1), e6. https://doi.org/10.2196/jmir.7.1.e6

Kori, K., Pedaste, M., Tonisson, E., Palts, T., Altin, H., Rantsus, R., … Ruutmann, T. (2015). First-year dropout in ICT studies. In *2015 IEEE Global Engineering Education Conference (EDUCON)* (pp. 437–445). IEEE. https://doi.org/10.1109/EDUCON.2015.7096008

Krauss, J. (2008). *Computer Science-in-a-Box: Unplug Your Curriculum*. (D. Burkhart & C. Stephenson, Eds.). Boulder, CO: The National Center for Women & Information Technology. Retrieved from https://www.ncwit.org/resources/computer-science-box-unplug-your-curriculum

Kumar, V., Winne, P., Hadwin, A., Nesbit, J., Jamieson-Noel, D., Calvert, T., & Samin, B. (2005). Effects of self-regulated learning in programming. In *Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05)* (pp. 383–387). IEEE. https://doi.org/10.1109/ICALT.2005.131

Lacey, T. A., & Wright, B. (2009). *Monthly Labor Review: Occupational Employment*

*Projections to 2018*.

Lapadat, J. C., & Lindsay, A. C. (1999). Transcription in Research and Practice: From Standardization of Technique to Interpretive Positionings. *Qualitative Inquiry*, *5*(1), 64–86. https://doi.org/10.1177/107780049900500104

Lawanto, O. (2010). Students' metacognition during an engineering design project. *Performance Improvement Quarterly*, *24*(2), 115–134.

Lawanto, O., Butler, D., Cartier, S. C., Santoso, H. B., Goodridge, W., Lawanto, K. N., & Clark, D. (2013). Pattern of Task Interpretation and Self-Regulated Learning Strategies of High School Students and College Freshmen during an Engineering Design Project. *Journal of STEM Education: Innovations and Research*, *14*(4), 15.

Lawanto, O., Butler, D., Cartier, S., Santoso, H. B., & Goodridge, W. (2013). Task Interpretation, Cognitive, and Metacognitive Strategies of Higher and Lower Performers in an Engineering Design Project: An Exploratory Study of College Freshmen. *International Journal of Engineering Education*, *29*(2), 459–475.

Lawanto, O., Cromwell, M., & Febrian, A. (2016). Students' Self-Regulation in Managing Their Capstone Senior Design Projects. In *123rd ASEE Annual Conference and Exposition*. New Orleans, LA, USA.

Lawanto, O., Goodridge, W. H., & Santoso, H. B. (2011). Task Interpretation and Self-Regulating Strategies in Engineering Design Project: an Exploratory Study. In *118th ASEE Annual Conference and Exposition*. Vancouver, British Columbia, Canada.

Lawanto, O., & Johnson, S. (2009). Student's cognitive self-appraisal, self-management, and the level of difficulty of an engineering design project: are they related? In

*American Society for Engineering Education Annual Conference*. Austin, TX.

Lawanto, O., Minichiello, A., Uziak, J., & Febrian, A. (2018). Students' Problem Understanding in Engineering Problem Solving (Under Review). *International Education Studies*.

Lee, K. D. (2014). *Foundations of Programming Languages*. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-13314-0

Leiviskä, K., & Siponen, M. (2013). Understanding Why IS Students Drop Out: Toward A Process Theory. In *ECIS 2013 Proceedings* (pp. 1–11). Retrieved from http://aisel.aisnet.org/cgi/viewcontent.cgi?article=1285&context=ecis2013_cr

Lenox, T. L., Woratschek, C. R., & Davis, G. A. (2008). Exploring declining cs/is/it enrollments. *Information Systems Education Journal, 6*(44), 11.

Leonard, N. R., Lester, P., Rotheram-Borus, M. J., Mattes, K., Gwadz, M., & Ferns, B. (2003). Successful Recruitment and Retention of Participants in Longitudinal Behavioral Research. *AIDS Education and Prevention*, *15*(3), 269–281. https://doi.org/10.1521/aeap.15.4.269.23827

Lewis, C. M., Anderson, R. E., & Yasuhara, K. (2016). "I Don't Code All Day": Fitting in Computer Science When the Stereotypes Don't Fit. In *Proceedings of the 2016 ACM Conference on International Computing Education Research - ICER '16* (pp. 23–32). New York, New York, USA: ACM Press. https://doi.org/10.1145/2960310.2960332

Lischner, R. (2001). Explorations: Structured Labs for First-Time Programmers. *ACM SIGCSE Bulletin*, *33*(1), 154–158. https://doi.org/10.1145/366413.364571

Litchfield, K., Javernick-Will, A., & Maul, A. (2016). Technical and Professional Skills of Engineers Involved and Not Involved in Engineering Service. *Journal of Engineering Education*, *105*(1), 70–92. https://doi.org/10.1002/jee.20109

Lucid Software Inc. (n.d.). Flowchart Symbols and Notation. Retrieved March 29, 2018, from https://www.lucidchart.com/pages/flowchart-symbols-meaning-explained

Lui, K. M., & Chan, K. C. C. (2006). Pair programming productivity: Novice–novice vs. expert–expert. *International Journal of Human-Computer Studies*, *64*(9), 915–925. https://doi.org/10.1016/j.ijhcs.2006.04.010

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

Madigan, E. M., Goodfellow, M., & Stone, J. A. (2007). Gender, perceptions, and reality: technological literacy among first-year students. *ACM SIGCSE Bulletin*, *39*(1), 410. https://doi.org/10.1145/1227504.1227453

Makel, M. C., & Plucker, J. A. (2014). Facts Are More Important Than Novelty. *Educational Researcher*, *43*(6), 304–316. https://doi.org/10.3102/0013189X14545513

MaxQDA. (n.d.). MaxQDA - N. 5: Intercoder agreement. Retrieved March 8, 2018, from https://www.maxqda.com/max12-tutorial/o-teamwork/n-5-intercoder-agreement

Mayer, R. (1996). Learners as Information Processors: Legacies and Limitations of Educational Psychology's Second Metaphor. *Educational Psychologist*, *31*(3), 151–161. https://doi.org/10.1207/s15326985ep3103&4_1

McNeill, N. J., Douglas, E. P., Koro-Ljungberg, M., Therriault, D. J., & Krause, I. (2016). Undergraduate Students' Beliefs about Engineering Problem Solving. *Journal of Engineering Education*, *105*(4), 560–584. https://doi.org/10.1002/jee.20150

Mercedes-Benz USA, L. (n.d.). SmartUSA. Retrieved January 1, 2017, from https://www.smartusa.com/

Miller, L. D., Soh, L. K., Chiriacescu, V., Ingraham, E., Shell, D. F., Ramsay, S., & Hazley, M. P. (2013). Improving Learning of Computational Thinking Using Creative Thinking Exercises in CS-1 Computer Science Courses. In *Proceedings - Frontiers in Education Conference, FIE* (pp. 1426–1432). https://doi.org/10.1109/FIE.2013.6685067

Miller, T. M., & Gerci, L. (2014). Improving metacognitive accuracy: How failing to retrieve practice items reduces overconfidence. *Consciousness and Cognition*, *29*, 131–140. https://doi.org/10.1016/j.concog.2014.08.008

MIT Media Lab. (n.d.). Scratch. Retrieved April 19, 2017, from https://scratch.mit.edu/

Newman, R. S., & Schwager, M. T. (1995). Students' Help Seeking During Problem Solving: Effects of Grade, Goal, and Prior Achievement. *American Educational Research Journal*, *32*(2), 352–376. https://doi.org/10.3102/00028312032002352

Ormrod, J. E. (2007). Behaviorism and Classical Conditioning. In J. E. Ormrod (Ed.), *Human Learning* (5th ed., pp. 32–47). Upper Saddle River, NJ: Allyn & Bacon.

Outlay, C. N., Platt, A. J., & Conroy, K. (2017). Getting IT Together: A Longitudinal Look at Linking Girls' Interest in IT Careers to Lessons Taught in Middle School

Camps. *ACM Transactions on Computing Education*, *17*(4), 1–17. https://doi.org/10.1145/3068838

Pallier, G., Wilkinson, R., Danthiir, V., Kleitman, S., Knezevic, G., Stankov, L., & Roberts, R. D. (2002). The Role of Individual Differences in the Accuracy of Confidence Judgments. *The Journal of General Psychology*, *129*(3), 257–299. https://doi.org/10.1080/00221300209602099

Paraskeva, F. (2007). Self-regulated learning strategies and computer self-efficacy in IT courses. In *Data Mining VIII: Data, Text and Web Mining and their Business Applications* (Vol. I, pp. 235–244). Southampton, UK: WIT Press. https://doi.org/10.2495/DATA070231

Parlante, N. (n.d.). CodingBat - code practice. Retrieved October 24, 2016, from http://codingbat.com/

Peixoto, F., Mata, L., Monteiro, V., Sanches, C., & Pekrun, R. (2015). The Achievement Emotions Questionnaire: Validation for Pre-Adolescent Students. *European Journal of Developmental Psychology*, *12*(4), 1–10. https://doi.org/10.1080/17405629.2015.1040757

Pekrun, R., & Perry, R. P. (2014). Control-Value Theory of Achievement Emotions. In R. Pekrun & L. Linnenbrink-Garcia (Eds.), *International Handbook of Emotions in Education* (pp. 120–141). Routledge. Retrieved from https://books.google.com/books?hl=en&lr=&id=8_UjAwAAQBAJ&pgis=1

Petticrew, M., & Roberts, H. (2006). *Systematic reviews in the social sciences: A practical guide*. Oxford, UK: Blackwell Publishing. Retrieved from

http://www.cebma.org/wp-content/uploads/Pettigrew-Roberts-SR-in-the-Soc-Sc.pdf

Pintrich, P. R. (2002). The role of metacognitive knowledge in learning, teaching, and assessing. *Theory into Practice*, *41*(4), 231–236.

Pintrich, P. R. (2003). A Motivational Science Perspective on the Role of Student Motivation in Learning and Teaching Contexts. *Journal of Educational Psychology*, *95*(4), 667–686. https://doi.org/10.1037/0022-0663.95.4.667

Pintrich, P. R. (2004). A conceptual framework for assessing motivation and self-regulated learning in college students. *Educational Psychology Review*, *16*(4), 385–407. https://doi.org/10.1007/s10648-004-0006-x

Pivkina, I., Pontelli, E., Jensen, R., & Haebe, J. (2009). Young Women in Computing: Lessons Learned from an Educational & Outreach Program. In *Proceedings of the 40th ACM technical symposium on Computer science education - SIGCSE '09* (Vol. 41, p. 509). New York, New York, USA: ACM Press. https://doi.org/10.1145/1508865.1509042

Polit, D. F., Beck, C. T., & Hungler, B. P. (2001). *Essentials of Nursing Research: Methods, Appraisal and Utilization* (5th ed.). Philadelphia, Pennsylvania, USA: Lippincott Williams & Wilkins.

Prescott, P., & Soeken, K. (1989). The Potential Uses of Pilot Work. *Nursing Research*, *38*(1), 60.

Rails Community. (2014). Rails - Web development that doesn't hurt. Retrieved from http://rubyonrails.org/

Renesse, C., & DiGrazia, J. (2018). Mathematics, Writing, and Rhetoric: Deep Thinking

in First-Year Learning Communities. *Journal of Humanistic Mathematics*, 24–63. https://doi.org/10.5642/jhummath.201801.04

Renumol, V. G., Janakiram, D., & Jayaprakash, S. (2010). Identification of Cognitive Processes of Effective and Ineffective Students During Computer Programming. *ACM Transactions on Computing Education*, *10*(3), 1–21. https://doi.org/10.1145/1821996.1821998

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., … Silver, J. (2009). Scratch: Programming for All. *Communications of the ACM*, *52*(11), 60. https://doi.org/10.1145/1592761.1592779

Rivera-Reyes, P. (2015). Students' Task Interpretation and Conceptual Understanding in Electronics Laboratory Work (Doctoral Dissertation). Logan, UT: Utah State University.

Rivera-Reyes, P., Lawanto, O., & Pate, M. L. (2016). Understanding Student Coregulation in Task Interpretation during Electronics Laboratory Activities. *International Education Studies*, *9*(7), 1. https://doi.org/10.5539/ies.v9n7p1

Rudolph, J., Niepel, C., Greiff, S., Goldhammer, F., & Kröner, S. (2017). Metacognitive confidence judgments and their link to complex problem solving. *Intelligence*, *63*, 1–8. https://doi.org/10.1016/j.intell.2017.04.005

Ruthmann, A., Heines, J. M., Greher, G. R., Laidler, P., & Saulters, C. (2010). Teaching computational thinking through musical live coding in scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education - SIGCSE '10* (p. 351). New York, New York, USA: ACM Press.

https://doi.org/10.1145/1734263.1734384

Ryan, R., & Deci, E. (2000). Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions. *Contemporary Educational Psychology*, *25*(1), 54–67. https://doi.org/10.1006/ceps.1999.1020

Saldana, J. (2008). An introduction to codes and coding. In *The Coding Manual for Qualitative Researchers* (pp. 1–31). SAGE Publications. Retrieved from https://scholar.google.com/scholar?q=An+Introduction+to+Codes+and+Coding&btnG=&hl=en&as_sdt=0%2C45#0

Samsung. (n.d.). Samsung. Retrieved January 1, 2017, from http://www.samsung.com/us/

Santoso, H. B. (2013). Computer Self-Efficacy, Cognitive Actions, and Metacognitive Strategies of High School Students While Engaged in Interactive Learning Modules (Doctoral Dissertation). Logan, UT: Utah State University. Retrieved from http://discover.lib.usu.edu/iii/encore/record/C__Rb3243218__Sharry budi santoso__Orightresult__X4?lang=eng&suite=cobalt

Santoso, H. B., Lawanto, O., Becker, K., Fang, N., & Reeve, E. M. (2014). High and Low Computer Self-Efficacy Groups and Their Learning Behavior from Self-Regulated Learning Perspective While Engaged in Interactive Learning Modules. *Journal of Pre-College Engineering Education Research (J-PEER)*, *4*(2), 11–28. https://doi.org/10.7771/2157-9288.1093

Saulnier, C. R., & Brisson, J. G. (2018). Design for Use: A Case Study of an Authentically Impactful Design Experience. *International Journal of Engineering Education*, *34*(2B), 769–779.

Schoenfeld, A. H. (1983). Episodes and Executive Decisions in Mathematical Problem Solving. In R. Lesh & M. Landau (Eds.), *Acquisition of Mathematics Concepts and Processes* (pp. 345–395). New York, New York, USA.

Seffah, A., Donyaee, M., Kline, R. B., & Padda, H. K. (2006). Usability measurement and metrics: A consolidated model. *Software Quality Journal*, *14*(2), 159–178. https://doi.org/10.1007/s11219-006-7600-8

Senske, N. (2011). A Curriculum for Integrating Computational Thinking. In *ACADIA Regional 2011 Parametricism* (pp. 91–98). Lincoln, NE. Retrieved from https://acadia.s3.amazonaws.com/paper/file/6G2VA4/AcadiaRegional_010.pdf

Shaft, T. M. (1995). Helping Programmers Understand Computer Programs: the Use of Metacognition. *ACM SIGMIS Database*, *26*(4), 25–46. Retrieved from http://dl.acm.org/citation.cfm?id=223280

Siddique, Z., Hardré, P. L., & Altan, D. (2015). Effects of a mechanical engineering design course on students' motivational features. *International Journal of Mechanical Engineering Education*, *43*(1), 44–74. https://doi.org/10.1177/0306419015581734

Sinatra, G. M., Broughton, S. H., & Lombardi, D. (2014). Emotions in Science Education. In R. Pekrun & L. Linnenbrink-Garcia (Eds.), *International Handbook of Emotions in Education* (pp. 415–436). Routledge.

Skinner, B. F. (1988). Whatever Happened to Psychology as the Science of Behavior? *Counselling Psychology Quarterly*, *1*(1), 111–122. https://doi.org/10.1080/09515078808251426

Smarthome. (n.d.). SmartHome Store. Retrieved January 1, 2017, from https://www.smarthome.com/

Teague, D. (2009). A People-First Approach to Programming. In *ACE '09 Proceedings of the Eleventh Australasian Conference on Computing Education* (pp. 171–180). Australian Computer Society, Inc. Retrieved from http://dl.acm.org/citation.cfm?id=1862737

TechTarget. (n.d.). What Is. Retrieved April 24, 2017, from http://whatis.techtarget.com

Teddlie, C., & Yu, F. (2007). Mixed Methods Sampling: A Typology With Examples. *Journal of Mixed Methods Research*, *1*(1), 77–100. https://doi.org/10.1177/2345678906292430

Tew, A. E., McCracken, W. M., & Guzdial, M. (2005). Impact of alternative introductory courses on programming concept understanding. In *Proceedings of the 2005 international workshop on Computing education research - ICER '05* (pp. 25–35). New York, New York, USA: ACM Press. https://doi.org/10.1145/1089786.1089789

The Joint Task Force on Computing Curricula. (2013). *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. *Practice*. https://doi.org/10.1145/2534860

Tigerfish. (n.d.). Transcription Style Guide. San Francisco, USA: Tigerfish. Retrieved from http://www.tigerfish.com/Transcription Style Guide Rev. 6.10.pdf

Toal, R. (n.d.). Programming Paradigms. Retrieved April 22, 2017, from http://cs.lmu.edu/~ray/notes/paradigms/

Trinh, V. Q., Chung, G. S., & Kim, H. C. (2012). Improving the Elder's Quality of Life

with Smart Television Based Services. *International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering*, *6*(7), 1794–1797.

Umapathy, K., & Ritzhaupt, A. D. (2017). A Meta-Analysis of Pair-Programming in Computer Programming Courses. *ACM Transactions on Computing Education*, *17*(4), 1–13. https://doi.org/10.1145/2996201

Unity Technologies. (2018). Unity 3D. Retrieved April 1, 2018, from https://unity3d.com

Utah State University. (n.d.). Computer Science - BS: A Catalog ACMS. Retrieved from http://catalog.usu.edu/preview_program.php?catoid=12&poid=9373#tt7849

Utah State University Office of Research and Graduate Studies. (n.d.). About IRB. Retrieved April 30, 2016, from http://rgs.usu.edu/irb/about/

van Teijlingen, E., & Hundley, V. (1998). The importance of pilot studies. *Nursing Standard : Official Newspaper of the Royal College of Nursing*, *16*(40), 33–36. https://doi.org/10.7748/ns2002.06.16.40.33.c3214

Vansteenkiste, M., Lens, W., Elliot, A. J., Soenens, B., & Mouratidis, A. (2014). Moving the Achievement Goal Approach One Step Forward: Toward a Systematic Examination of the Autonomous and Controlled Reasons Underlying Achievement Goals. *Educational Psychologist*, *49*(3), 153–174. https://doi.org/10.1080/00461520.2014.928598

Viera, A. J., & Garrett, J. M. (2005). Understanding interobserver agreement: the kappa statistic. *Family Medicine*, *37*(5), 360–363. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/15883903

Wang, J., Hejazi Moghadam, S., & Tiffany-Morales, J. (2017). Social Perceptions in

Computer Science and Implications for Diverse Students. In *Proceedings of the 2017 ACM Conference on International Computing Education Research - ICER '17* (pp. 47–55). Tacoma, Washington, USA: ACM Press. https://doi.org/10.1145/3105726.3106175

Weisser, M. (1991). The Computer for the Twenty-First Century. *Scientific American*, *3*(265), 94–104.

Whiting, L. S. (2008). Semi-structured interviews: guidance for novice researchers. *Nursing Standard*, *22*(23), 35–40. https://doi.org/10.7748/ns2008.02.22.23.35.c6420

Whittington, K. J. (2004). Infusing Active Learning Into Introductory Programming Courses. *Journal of Computing Sciences in Colleges*, *19*(5), 249–259. Retrieved from http://dl.acm.org/citation.cfm?id=1060081.1060111

Wiersema, J. A., & Licklider, B. L. (2009). Intentional Mental Processing: Student Thinking as a Habit of Mind. *Journal of Ethnographic & Qualitative Research*, *3*(2), 117–127.

Wigfield, A., & Eccles, J. (2000). Expectancy-Value Theory of Achievement Motivation. *Contemporary Educational Psychology*, *25*(1), 68–81. https://doi.org/10.1006/ceps.1999.1015

Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2010). In Support of Pair Programming in the Introductory Computer Science Course. *Computer Science Education*, *12*(3), 197–212. https://doi.org/10.1076/csed.12.3.197.8618

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, *49*(3), 33. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2008). Computational Thinking and Thinking About Computing. *Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences*, *366*(1881), 3717–25. https://doi.org/10.1098/rsta.2008.0118

Winne, P. H., & Perry, N. E. (2000). Measuring Self-Regulated Learning. In M. Boekaerts, P. R. Pintrich, & M. Zeidner (Eds.), *Handbook of Self-Regulation* (pp. 531–566). Orlando, Florida, USA: Academic Press.

Yin, R. K. (2009). *Case Study Research: Design and Methods*. Los Angeles, CA, USA: Sage Publications. Retrieved from http://discover.lib.usu.edu/iii/encore/record/C__Rb2559607__SCase study research: Design and methods__Orightresult__X4?lang=eng&suite=cobalt

Zimmerman, B. J., Heart, N., & Mellins, R. B. (1989). A Social Cognitive View of Self-Regulated Academic Learning. *Journal of Educational Psychology*, *81*(3), 329–339. https://doi.org/10.1037//0022-0663.81.3.329

# APPENDICES

# APPENDIX A. THE 2016 REU PROJECT DESCRIPTION

In this modern age, computers and smart devices are pervasive. It has been used to improve the quality of, for example, telecommunication, transportation, medical, and security services. Consequently, employers expect the next generation of workers to have some basic knowledge in applying these technological advancements to solve their problems. In other words, they are expected to have some computer science (CS) skills. Being aware of the importance of CS skills in the future, the states of Florida, Chicago, Utah, and California decided to incorporate CS-base courses in their respective K-12 curriculum through what commonly known as computational thinking. On the other hand, educational researchers in education have shown that students with better self-regulated learning (SRL) skills will excel in academic learning and problem solving compared to their counterparts. However, little has been known about students' SRL in programming design, one of the core activities in CS. This study aims to bridge that gap by assessing and describing CS students' SRL while they engaged in programming tasks. A qualitative case study will be conducted to three-to-four CS students who will be recruited from the CS department at Utah State University using the criterion sampling method. The participants will be asked to spend 2.5 hours to answer two programming questions, which will be audio and video recorded. Framed in Butler and Cartier's SRL model, the attribute, process, in-vivo, and pattern coding approaches will be applied to the transcribed data. Each participant will receive $25 and a personalized SRL profile as tokens of appreciation. A member checking activity will be conducted at the end of the data analysis process to validate research findings.

# APPENDIX B. THE 2016 REU PROJECT SCHEDULE

| Date | Activity | Outcomes |
|---|---|---|
| Week 1<br>06/06 - 06/10 | Seminars and training:<br>• Seminar "Self-Regulated Learning: What is it?"<br>• Seminar "A Brief Introduction to Qualitative Methods"<br>• Training: Institutional Review Board (IRB)<br><br>Introduction to research (in 'All Participants' folder):<br>• Searching for academic literatures: EBSCO and ERIC<br>• Best practice: research log book<br>• Taking notes: annotated bibliography<br>• File naming and version convention<br><br>Literature:<br>• Self-regulated learning (1 provided by mentor, 1 provided by you)<br>• Concept map (1 provided by mentor)<br><br>Debriefing:<br>• Issues/suggestions/resolutions<br>• Planning for next week | Each student:<br>• 2 summaries of seminars<br>• IRB training certificate<br>• Concept map of the seminar and literature |
| Week 2<br>06/13 - 06/17 | Seminar:<br>• Seminar "Curriculum and Research: Developing an Educational Research Question"<br>• Seminar "Educational Data Analysis with SPSS"<br><br>Learn programming:<br>• Complete: Light Bot stage 1 – 3 (http://lightbot.com/hocflash.html)<br>• Complete: Elsa Frozen puzzle 1 - 20 (http://code.org/api/hour/begin/frozen)<br>• SRL (task interpretation, planning, strategic action, and monitoring) activities note about your learning | Each student:<br>• 2 summaries of the seminars<br>• 1 screenshot which showed the completion of all Light Bot stages<br>• 1 screenshot which showed the completion of all Elsa Frozen puzzles<br>• Your programming-SRL note |

| Date | Activity | Outcomes |
|---|---|---|
| | Literature:<br>• Qualitative research methods (1 provided by mentor, 1 provided by you)<br>• Verbal protocol (1 provided by mentor)<br>• Application of verbal protocol (1 provided by you)<br><br>Debriefing:<br>• Issues/suggestions/resolutions<br>• Planning for next week | • Concept map of the literature |
| Week 3<br>06/20 - 06/24 | Seminar:<br>• Seminar "Responsible Research"<br><br>Getting familiar with verbal protocol:<br>• Watch videos about conducting a verbal protocol (1 provided by mentor, 1 provided by you)<br>• Discuss possible issues and its handling method on conducting verbal protocol in this research<br><br>Literature:<br>• Attribute of problem (1 provided by mentor)<br>• Transcription method (1 provided by mentor)<br>• Qualitative study in computer science education (1 provided by you)<br><br>Data collection preparation (provided by mentor):<br>• Discuss the research methodology<br>• Discuss the research question<br>• Discuss the research instrument<br>• Learn to use data collection tools | Each student:<br>• 1 summary of the seminar<br>• Concept map of the literature<br><br>Group<br>• Note about the research methodology<br>• List of possible issues and its handling method in verbal protocol |

| Date | Activity | Outcomes |
|---|---|---|
| | Debriefing:<br>• Issues/suggestions/resolutions<br>• Planning for next week | |
| Week 4<br>06/27 - 07/01 | Data collection and transcription:<br>• From 3 or 4 computer science students<br><br>Preparation for qualitative data analyses:<br>• NVivo9 for transcribing<br>• MaxQDA12 for coding<br><br>Debriefing:<br>• Issues/suggestions/resolutions<br>• Planning for next week | Group:<br>• 1 to 4 raw data<br>• 1 to 4 transcription data<br>• 1 to 4 signed informed consents |
| Week 5<br>07/05 - 07/08 | Data collection and transcription:<br>• From 3 or 4 computer science students<br><br>Literature:<br>• Qualitative data analyses (2 provided by mentor, 1 provided by you)<br>• Interrater reliability (1 provided by mentor)<br><br>Debriefing:<br>• Issues/suggestions/resolutions<br>• Planning for next week | Each student:<br>• Concept map of the literature<br><br>Group:<br>• 3 to 4 final raw data<br>• 3 to 4 final transcription data<br>• 1 to 4 signed informed consents |
| Week 6<br>07/11 - 07/15 | Phase 1 data analysis:<br>• Segmentation and coding: attribute and process<br>• Interrater reliability<br><br>Phase 2 data analysis preparation:<br>• Identify emergent strategies<br><br>Debriefing:<br>• Issues/suggestions/resolutions<br>• Planning for next week | Each student:<br>• Emergent strategies<br><br>Group:<br>• Phase 1: segment and coding data<br>• Phase 1: coding statistics<br>• Phase 1: interrater score |

| Date | Activity | Outcomes |
|---|---|---|
| Week 7<br>07/18 - 07/22 | Phase 2 data analysis:<br>• Coding: in-vivo, pattern<br>• Interpretation of the category<br>• Select examples of events or personal experiences<br><br>Literature:<br>• Computer science education (1 provided by mentor, 1 provided by you)<br><br>Debriefing:<br>• Issues/suggestions/resolutions<br>• Planning for next week | Each student:<br>• Concept map of the literature<br><br>Group:<br>• Phase 2: segment and coding data<br>• Phase 2: coding statistics<br>• Phase 2: interrater score<br>• Phase 2: interpretation |
| Week 8<br>07/26 - 07/29 | Data analysis:<br>• Interpretation<br><br>Member checking:<br>• 3 to 4 personalized SRL reports for each participants<br><br>Debriefing:<br>• Issues/suggestions/resolutions<br>• Planning for next week | Group:<br>• Final interpretation<br>• 3 to 4 personalized SRL report |
| Week 9<br>08/01 - 08/05 | Member checking:<br>• Revise findings based on member checking results<br><br>Documentation:<br>• Preparing research results presentation<br>• Develop a report of the analyzed data/findings | Group:<br>• Revised interpretation<br>• Research result presentation |
| Week 10<br>08/08 - 08/14 | At home research assignments:<br>Final report due on Friday, August 14th at 11:59 PM by email to Dr. Lawanto (olawanto@usu.edu) and Andreas (andreas.febrian@aggiemail.usu.edu) | Each student:<br>• Final REU report |

# APPENDIX C. THE 2016 REU RECRUITMENT PUBLICATION

Title: Research Participants Recruitment for CS Education Research

Content:

Courtenae Palmer,

My name is Andreas Febrian. I am a doctoral student in the Engineering Education Department. Yesterday we talked about disseminating information to CS undergraduate students; here is the information:

One of our REU summer projects is about assessing self-regulated learning of computer science students while engaged in programming design (see http://reu.usu.edu/projects.php#cP2). The goal of the study is to describe their task interpretation and planning strategies. We would love to recruit 3 to 4 undergraduate CS students who are willing to:

- Dedicate 2.5 hours in Logan between June 27 – July 8 to solve two programming design questions.
- Dedicate 15-30 minutes between July 26 – 29 to read a personalized report of his/her SRL and to comment about it (e.g., whether our interpretations were wrong or not). This can be done through a phone call, skype, or email.

Each participant will receive a $40 gift card and a personalized report of their SRL. Educational researchers found that students with higher self-regulation tend to perform better academically compared to their counterparts. The personalized SRL report can help students to identify their SRL strengths and weaknesses.

If you were interested in participating or had any questions, please contact me at andreas.febrian@aggiemail.usu.edu.

# APPENDIX D. THE 2016 REU DEMOGRAPHICS SURVEY

## Demographic Survey

You have agreed to participate in the REU 2016 Project #2. This survey is intended to collect demographic information about you, which includes basic and academic information. If you have any questions or concerns, please contact Andreas Febrian (andreas.febrian@aggiemail.usu.edu).

## Personal Information

Questions with asterisk (*) are mandatory.

Name*: _____

Nickname (research ID)*: _____

Please provide a name as your research identifier. It has to be at least four characters long and only contains alphabet (A-Za-z). You may also use your real name.

Gender*:

- o Male
- o Female

Your age*: _____

Ethnic:

- o African American
- o Asian-Pacific Islander
- o Caucasian
- o Hispanic
- o Native American
- o Other

Phone (with area code)*: _____

## Academic/Discipline Information

Questions with asterisk (*) are mandatory.

Current cumulative GPA (on a 4.00 scale)*: _____

Latest CS 1400 (Introduction to Computer Science--CS 1) grade*:

- o A
- o A-
- o B+
- o B
- o B-
- o C+
- o C
- o C-
- o Below C-

Please mark the all courses that you have passed with C- or better:

- ☐ MATH 1210: Calculus I (QL)
- ☐ CS 1410: Introduction to Computer Science--CS 2 (QI)
- ☐ CS 1440: Methods in Computer Science
- ☐ MATH 1220: Calculus II (QL)
- ☐ CS 2420: Algorithms and Data Structures--CS 3 (QI)
- ☐ MATH 3310: Discrete Mathematics
- ☐ CS 2410: Introduction to Event Driven Programming and GUI's
- ☐ CS 2610: Developing Dynamic, Database-Driven, Web Applications
- ☐ CS 3100: Operating Systems and Concurrency
- ☐ CS 3450: Introduction to Software Engineering (CI)
- ☐ CS 5000: Theory of Computability

- ☐ CS 5050: Advanced Algorithms
- ☐ MATH 2270: Linear Algebra (QI)
- ☐ CS 4700: Programming Languages
- ☐ CS 5300: Compiler Construction

Rate your interest in programming (0 - 10): _____

Please mark all programming paradigms that you are proficient in:

- ☐ Imperative (Procedural) Programming
- ☐ Functional Programming
- ☐ Logic Programming
- ☐ Object Oriented Programming
- ☐ Declarative Programming
- ☐ Visual Programming

Please estimate the number of hours you have spent in doing programming:

_____

Are there any additional factors that you feel have affected your programming abilities?

If so, what are they?

_____

_____

_____

_____

When do you want to meet with us?

Please select more than one.

- ☐ Thursday, June 30
- ☐ Friday, July 1
- ☐ Tuesday, July 5
- ☐ Wednesday, July 6
- ☐ Thursday, July 7
- ☐ Friday, July 8

What is the best time to meet on Tuesday, Wednesday, or Thursday?

Please select more than one.

- ☐ 09:00 AM - 11:30 AM
- ☐ 09:30 AM - 12:00 PM
- ☐ 10:00 AM - 12:30 PM
- ☐ 10:30 AM - 01:00 PM
- ☐ 11:00 AM - 01:30 PM
- ☐ 11:30 AM - 02:00 PM
- ☐ 12:00 PM - 2:30 PM
- ☐ 12:30 PM - 3:00 PM
- ☐ 01:00 PM - 3:30 PM
- ☐ 01:30 PM - 4:00 PM
- ☐ 02:00 PM - 04:30 PM

What is the best time to meet on Friday?

Please select more than one.

- ☐ 09:00 AM - 11:30 AM
- ☐ 09:30 AM - 12:00 PM
- ☐ 10:00 AM - 12:30 PM
- ☐ 10:30 AM - 01:00 PM

Initials*: _____

- ☐ I certify that all information given in this application packet is accurate and true to the best of my knowledge. I understand that submission of false information is grounds for immediate dismissal from this study.

# APPENDIX E. THE 2016 REU INTRODUCTION SCRIPT

**Researchers:** Hi! Thank you for coming in today, how are you doing?

**Participant**: Fine.

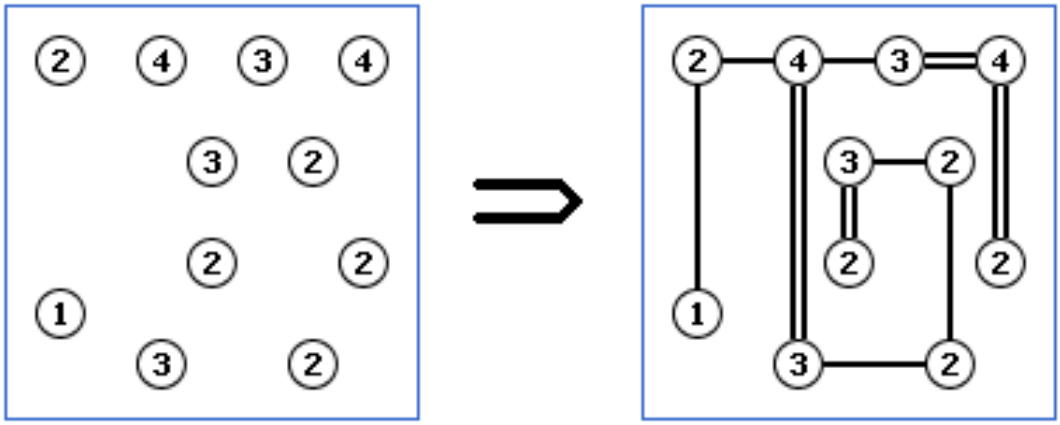
**Researchers**: Great! We have some chocolate here for you to eat throughout the session, feel free to take as much as you like.

Before we get started, we do want to remind you that we will be filming this session and will be using the audio and video recordings in our research. Here is the consent form, which we would like you to sign. Please take your time reading it and if you have any questions you would like to ask before you agree to participate, we will gladly answer them.

**Participant**: No, I have no questions, and yes, I will sign the form.

**Researchers**: Great! As you may already know, we are researching the self-regulating behaviors of computer science students, specifically those which occur during attempts to solve problems. To accomplish our research goals, we have several other students, similar to you, who either have already done or will soon do exactly what you are about to do today.

You will be providing us with a verbal protocol, or think-aloud, which means that as you work on the problems we give you, we would like you to speak your thoughts out loud as they come to you. We will demonstrate an example of verbal protocol using a simple bridges puzzle.




**Researchers**: As we demonstrated, please say every thought that goes through your head, no matter how small or irrelevant you think it is and speak loudly and clearly. If you are silent for a while, we may ask questions to help you stay focused, and/or to remind you that we need to hear your thoughts. Do you have any questions so far?

Participant: No questions.

**Researchers**: Good! Today, we will give you four problems total, two practice questions to get you used to the idea of thinking out loud, and two more questions after those. We will give you each problem one at a time, and we want hear how you work through the problem from beginning to end. Please take your time and be thorough. You may use as much paper as you need. Also, it is not important to get the "right answer". In the last two problems there is no "right answer" we are more interested in the way you work through the problems. Do you have any final questions before we begin?

**Participant**: No.

**Researchers**: Then here is your first practice problem.

# APPENDIX F. THE 2016 REU PERSONALIZED SRL REPORT

# DanielO Report

**Monitoring: Satisfying Requirements**

In the monopoly problem, you also went back and made sure all requirements were met several times. For example, you said “Let’s see… what else did they have? Castle, fortress, or inn. Alright now, what else should a space have?” This may be due to the length of the problem and all the specifications that were mentioned. This was done throughout the entirety of the problem. You later said “All right so, valid number of players here, valid number of players, table top, so what else should the game have?” This can also be seen as monitoring the task, since meeting all the requirements was your task interpretation.

**Monitoring: Monitoring of the Task**

While you were solving the monopoly problem, you reminded yourself that you were doing pseudo-code because the problem wasn’t asking you to go any further. You said things like “hmm, I mean, it is pseudo-code, so maybe I shouldn’t worry so much about that” and “this is pseudo-code of course, this is not how you write any of this, but I’m just writing it like this to make it easier to actually write down”. You recognized that your task was to create pseudo-code, but had to monitor yourself because it often felt like you wanted to go beyond that and write more accurate segments of code. An example of this is when you debate on whether a variable should be private or public. You say: “Um… I’m not sure if it should be public or private. I guess, public.”

Pseudo-code can be an informal a skeleton that will aid them in the design of the program. Keywords may not truly be important in the pseudo-code process because once

you are able to type, an IDE compiles for you and if there are any errors you can begin debugging. In your case, it seemed as if you wanted to make your pseudo-code as close to the real thing as possible so that when you actually start coding, the process will be as simple as possible. Since the problem never mentioned future coding, this is seen as your personal objective: to include keywords and make the pseudo-code as thorough as possible.

Another example of this is when you say “I think that I just realized I need a constructor, because yeah, game actually that’s not how you write constructors inside of classes when you do inherency files here I just make-- is called game, and that’s the constructor”. It should also be noted that your attention to detail in pseudo-code can be linked to observation bias. Maybe since we were observing you, you weren’t sure how much detail you should include for the purposes of our research?

**Monitoring: Instruments Used**

You were the only participant to ask whether a pencil could be used. We feel this is noteworthy because you provided the reasoning as, “It’s just if I get myself into a corner, I want to kind of wiggle out of it.” With this statement, you are aware of your own monitoring techniques. It shows that when you make a mistake, you are able to erase and start over, which is a good technique to employ in computer science.

**Strategic Action: Reading the Title**

Although it may not seem like a significant strategy, reading the title of a problem is an effective way to gain insight of what the problem will entail. You read the title to every question which means that you consider he title to be an integral part of each problem.

You also read the numbers in the title. For example, “Question four. Oh, four, question two. Okay. Monopoly in the middle ages”. Also you read, “So, the last standing man, ominous.” The addition of the word “ominous” to the title gives us the impression that you anticipate the problem will portray evil or harm. You draw this strictly after looking at the problem and reading the title. This not only shows that you read the title, but you strategically read the title by allowing yourself to anticipate characteristics of the problem, which is an important part of monitoring.

**Monitoring: Monitoring Interest Level**

Trough out your problem-solving procedure you verbalized how you felt about the problems. After reading several of the requirements for the monopoly problem, you said, “I don’t like monopoly”. We believe that your feelings toward a problem are external factors that affect your approach, so an interesting question to ask yourself is, “Would my approach and strategies used on this problem be different if I liked monopoly?” Later you say, “Yeah. That actually wasn’t as bad as I thought, okay I think that’s it.” This leads us to believe that you initially thought the problem would be more tedious.

When a problem is perceived as tedious, the interest level in that problem is likely to drop. When interest level drops, performance may not be as efficient in comparison to when you are truly engaged in the problem. In your case, it appears as though the initial feelings of dislike diminished once you completed the task. Personal Note: It could be effective to not only monitor your emotions before approaching the task, like you did, but if the emotions interfere with your objective, maybe monitor what you can do with those emotions to strategically accomplish your task.

**Strategic Approach: Skipping Parts of Problem**

“After you input your three values, the magic black box will output ‘true’ if your friendship is compatible and ‘false’ if it’s not compatible. That’s the algorithm.” Here, you are implicitly indicating that you will skip the code for now and come back to it later. This is a nonlinear approach that is focused on determining the task before going back and reading the code. This is strategic because once you determine the task, you can actually run through the code and know what you are looking for, which can same time in time-sensitive situations such as exams.

# Depend Report

**Task Interpretation: Sticking with Initial Task Interpretation**

In the problem called “the last man standing” you demonstrated a non-conventional understanding of the problem objective and a unique personal objective. You appeared to initially interpret the goal of the problem as…

“…using the algorithm, just find the perfect position of where he should stand. It should probably calculate, it should like simulate, the number of people at first and then already calculate, really quickly, because you don’t want to wait like a day because then you’re going to die. Then, he should be able to pick the position he wants so I’m going to have to look for a pattern and it has to be generic, like you can’t just hardcode.”

Here you explicitly describe your task interpretation, and show us that finding the “quickest” solution is something you feel is necessary despite not being asked to do so by the problem. Clearly, you noticed that the problem asked you to “simulate” the suicidal method in the code, but you interpreted this to mean that your code should simply take the number of people as an input, then calculate Josephus’ position in whatever way would be fastest. Later, you added that, “Right now, I’m just trying to find a pattern. A generic pattern that I can use.” This shows that you felt that you needed to find a strong pattern which your code could use to find Josephus’ position faster than if it simulated the whole suicidal process.

You stick to your plan of finding a pattern for 31 minutes. During this time, you questioned whether your solution was correct, but you never questioned whether you task understanding was correct. For example, you said, “Yeah, I think there’s a different way

to approach this that I'm not thinking of, but I kind of just want to do one more." This led to repetition, and to the realization that the problem-solving approach you were using may not have been the most efficient. Perhaps in the future, it would be useful for you to monitor your understanding of the task throughout your problem-solving process.

**Strategic Action: Marking for Organization**

Secondly, we noticed that you marked the papers, and used them for figuring, all to better organize your work. This was observed on multiple problems, such as in in the troubleshooting problem, where you said, "Okay. So I'm just going to underline where I think the error is." In the board game design problem, you used check marks to specify which constraints you had satisfied already, and question marks for those which you would return to later. In the "output prediction" problem, you used the extra paper to write down the results you got as you resolved each part of the complicated logic statements.

**Strategic Action: Not Reading the Title**

We also noticed that you often do not read the title of a problem at the start of the problem, most notable in the problem "monopolies in the middle ages" where you only made the connection once you had reached the end. You say "I just realized it says 'monopolies in the middle ages'. That explains why I was thinking monopoly the whole time". Perhaps, if you had read the title first, you would have gained more context about the problem's task and it could have helped to facilitate the problem-solving process.

**Strategic Action: Reading Silently**

You clearly demonstrate, in the initial practice problems, a strong preference for reading silently to yourself. However, you also give us reasons to believe that different things work for you. After the second practice problem, you admitted to us that, “I had to re-read the first paragraph like a couple of times to really understand what’s going on”. Also, when you tried reading the problem out loud, you would restate every sentence after reading it to check your understanding. This all tells us that your process for understanding problems is more involved than you may realize, and all the little things you do while interpreting a problem may help you develop more accurate interpretations, and may help you do so faster.

**Strategic Action: Linear Approach**

In the “monopolies in the middle ages” problem, after you feel you have fulfilled the majority of the requirements you go back and run through the list one by one. For example, you say, “They have to start with different items. I’m not really sure on this one, maybe… huh, I am going to put a question mark for there”. If you have satisfied the requirement for the game, you put a checkmark, if you have not, you put a question mark. This not only proves the aforementioned about your strategy to mark for organization, it also shows that you choose to solve a problem linearly. It helps you to keep track of what has been done and what you still need to work on.

# George Report

**Strategic Action: Reads Title**

Although this may seem like a common action, you'd be surprised that some of our other participants didn't read the title. Not only did you read the title every single time, you included the numbers of the problems and verbalized your initial impressions. For example, you said, "Umm… number 2. The second 2 seems redundant. Output prediction." Here there may not be any significance in the extra "2", but you still take notice of the numbers, which shows you pay great attention to detail. Also, you say "Monopolies in the middle ages. This feels more like an essay question. Still only one page though, not as bad as the bar exam." The length of the problem led you to believe this was similar to an essay question. When you did this, you were accessing prior knowledge. You are familiar with what essay questions looks like because you've encountered them before, so you are able to recognize an essay question according to its length. This is a useful strategy which we observed while you solved the problems.

**Strategic Action: Accessing Prior Knowledge**

Even for trivial approaches, you reminded us that you learned certain tactics in the past, and these past experiences led you to choose to employ a similar strategy at that particular moment. Perhaps this helps to reinforce your actions in your mind and provides you with a stronger foundation as you proceed with the problem. For example, you said, "Okay, well I'm going to start by reading the instructions because that's what I've learned is always the best think to start with." It can be assumed that we all mostly start by reading the instructions, but you credit this approach to your prior experience (which

is interesting). Also, as you are reading the problems, you mention, “Okay. I am starting to run through some of the programs I have already done in C++.” This proves you have experience in computer science and you understand that many times former programs can help us form the basics of a new program. Students who do not have prior knowledge in computer science do not have the memory bank that you do, and don’t have access such information. You do, and you are making notable use of it by using your prior knowledge when you know it will be advantageous. Another example of this is when you say, “Alright, well I think remembering back to the games I’ve designed in my other classes, I’m trying to decide if I want game objects to start with…” Again, you have a memory bank of games you’ve designed, so you are able to go through that memory bank and find an approach that would best suit this particular problem.

**Planning Strategies: Skipping Sections of the Problem**

During our observation, you strategically choose to skip sections of the problem. You say “After three days of meetings… the people in charge have agreed on same basic aspects of the game, which are… I’m going to skip the aspects again and see what I’m supposed to do with that information before I read it.” This was a particularly lengthy problem (the one you mentioned felt like an “essay question”), so perhaps it is usual for a computer science student to skip the mumbo-jumbo, and try to find out what the problem is truly asking. However, it was interesting to see that you still used this strategy even when the problem wasn’t lengthy. For one of the relatively short practice problems you say, “I’m going to skip the code and see what else I have to do before I go back and look at it, so that I know what I’m looking for.” You were the only participant who chose a nonlinear

approach to the problems. You are interested in knowing what the task is before you go back and read the details, which shows that you are a task oriented person and place a great deal of importance on the task.

**Task Interpretation: Taking the Task Literally**

The problems contain context to imitate some of the problems that are assigned in typical computer science courses. You take the context literally and make it your personal objective to fulfill the details mentioned in context. For example, “Yeah, the rest of the team could easily develop the rest of the game, so hopefully they can read my handwriting”. Other participants may simply see it as an unimportant problem on paper they will try to solve (while employing verbal protocol) to aid us in our research, but you consider everything mentioned in the instructions and view it as part of your duty to satisfy the requirements literally. We believe this will be a valuable strategy to use in real-life situations where you have to consider outside factors such as being the leader on a project assigned by your employer. Since you are concerned with fulfilling every requirement in classroom-given problems, you will be prepared to work in a team. You mention “They didn’t give very good instructions on those. But I don’t want to go ask my boss, because you know… and then you get fired, they want you to think.” This is a perfect example of the aforementioned role-playing that you demonstrated. Perhaps you understand that the purpose of a computer-science education is to be prepared for the work-field/industry.

**Strategic Action: Organizing Thoughts on Paper**

You make use of the scratch paper offered to you by using it to make sketches that help you organize your thoughts. Once your thoughts are on paper in the form of a diagram, the situation is clearer to you, and you are able to proceed from there. This is a strategy that is often taught during our earlier years of school, and one that some of us forget in our college education. It is especially important for students in the technical field such as engineering and computer science to have a strong foundation to work off of. A visual aid is a great example of building yourself a strong foundation which you can use strategically to your benefit. Here are some examples:

“So I’m going to use this sheet to just sketch out a little bit of where things are.”

“So we’ll put board over here and we put what it has and then we can… kind of like a backwards flowchart.”

# APPENDIX G. ONLINE APPLICATION FORM

## Consent Letter

[The consent letter place holder]

Are you willing to participate in the study*?

- o I WILL participate in this study
- o I WILL NOT participate in this study

## Screening

Thank you for your willingness to participate in this study. Please answer four screening questions to determine your eligibility to participate in this study.

Questions with an asterisk (*) are mandatory.

Your age*: _____

Are you a senior Computer Science students at USU*?

- o Yes
- o No

Current cumulative GPA (on a 4.00 scale)*: _____

Latest CS 1400 (Introduction to Computer Science--CS 1) grade*:

- o A
- o A-
- o B+
- o B
- o B-
- o C+
- o C
- o C-
- o Below C-

## Personal Information

Questions with an asterisk (*) are mandatory.

Name*: _____

Nickname (research ID): _____

Please provide a name as your research identifier. It has to be at least four characters long and only contains alphabet characters (A-Za-z). You may also use your real name.

Primary email address*: _____

Please provide your main email address. Further research communication will be delivered to this address.

Gender*:

- o Male
- o Female

Ethnicity:

- o African American
- o Asian-Pacific Islander
- o Caucasian
- o Hispanic
- o Native American
- o Other

Phone (with area code)*: _____

## Academic/Discipline Information

Questions with an asterisk (*) are mandatory.

Please mark all the courses that you have passed with a C- or better*:

- ☐ MATH 1210: Calculus I (QL)
- ☐ CS 1410: Introduction to Computer Science--CS 2 (QI)
- ☐ CS 1440: Methods in Computer Science
- ☐ MATH 1220: Calculus II (QL)
- ☐ CS 2420: Algorithms and Data Structures--CS 3 (QI)
- ☐ MATH 3310: Discrete Mathematics
- ☐ CS 2410: Introduction to Event Driven Programming and GUI's
- ☐ CS 2610: Developing Dynamic, Database-Driven, Web Applications
- ☐ CS 3100: Operating Systems and Concurrency
- ☐ CS 3450: Introduction to Software Engineering (CI)
- ☐ CS 5000: Theory of Computability

- ☐ CS 5050: Advanced Algorithms
- ☐ MATH 2270: Linear Algebra (QI)
- ☐ CS 4700: Programming Languages
- ☐ CS 5300: Compiler Construction
- ☐ Not Applicable

Rate your interest in computer programming (1-10): _____

Please mark all computer programming paradigms that you are proficient in:

- ☐ Imperative (Procedural) Programming
- ☐ Object Oriented Programming
- ☐ Visual Programming
- ☐ Functional Programming
- ☐ Logic Programming
- ☐ Declarative Programming
- ☐ Not Applicable - I do not know

Please estimate the number of hours you have spent in doing programming*:

_____

Are there any additional factors that you feel have affected your programming abilities?

If so, what are they?

_____

_____

What is your motivation to participate in this study?

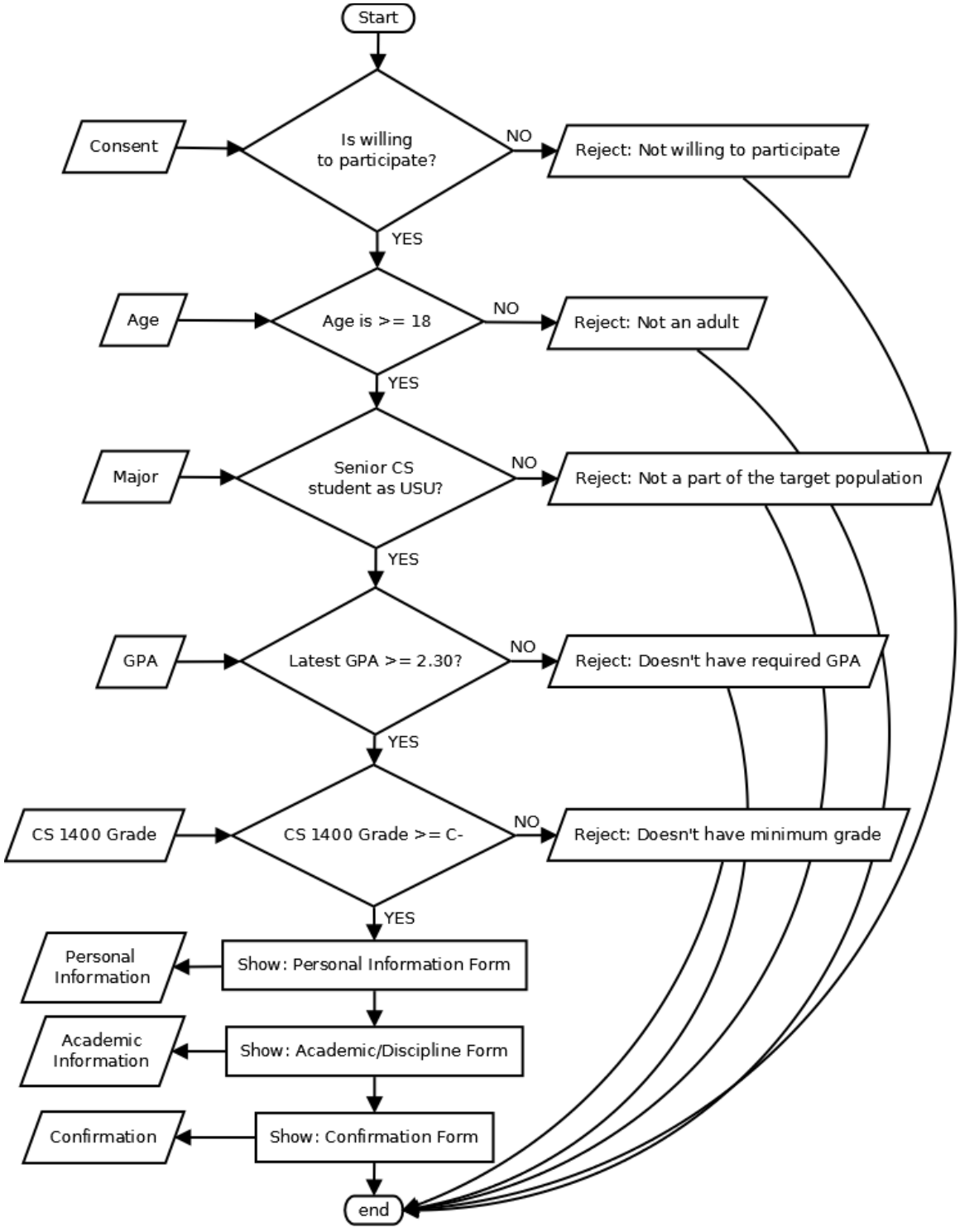
_____

_____

Initials*: _____

- [ ] I certify that all information given in this application packet is accurate and true to the best of my knowledge. I understand that submission of false information is grounds for immediate dismissal from this study.

# APPENDIX H. ONLINE APPLICATION SCREENING FLOWCHART


Start
Consent
Is willing to participate?
NO
Reject: Not willing to participate
YES
Age
Age is >= 18
NO
Reject: Not an adult
YES
Major
Senior CS student as USU?
NO
Reject: Not a part of the target population
YES
GPA
Latest GPA >= 2.30?
NO
Reject: Doesn't have required GPA
YES
CS 1400 Grade
CS 1400 Grade >= C-
NO
Reject: Doesn't have minimum grade
YES
Personal Information
Show: Personal Information Form
Academic Information
Show: Academic/Discipline Form
Confirmation
Show: Confirmation Form
end

# APPENDIX I. DEMOGRAPHICS SURVEY

Thank you for participating in this study. Please remember to keep your copy of signed informed consent in a safe place. We are in the process of pre-analyzing your data. We will contact you if we need some clarification. As part of this study, we need you to fill this demographic form. All questions are mandatory.

Your nickname (research ID): _____

Please provide your selected research identifier; please refer to the email if you forget.

Ethnicity:

- o African American
- o Asian-Pacific Islander
- o Caucasian
- o Hispanic
- o Native American
- o Other

Please mark all the courses that you have passed with a C- or better*:

- ☐ MATH 1210: Calculus I (QL)
- ☐ CS 1410: Introduction to Computer Science--CS 2 (QI)
- ☐ CS 1440: Methods in Computer Science
- ☐ MATH 1220: Calculus II (QL)
- ☐ CS 2420: Algorithms and Data Structures--CS 3 (QI)
- ☐ MATH 3310: Discrete Mathematics
- ☐ CS 2410: Introduction to Event Driven Programming and GUI's
- ☐ CS 2610: Developing Dynamic, Database-Driven, Web Applications
- ☐ CS 3100: Operating Systems and Concurrency
- ☐ CS 3450: Introduction to Software Engineering (CI)

- ☐ CS 5000: Theory of Computability
- ☐ CS 5050: Advanced Algorithms
- ☐ MATH 2270: Linear Algebra (QI)
- ☐ CS 4700: Programming Languages
- ☐ CS 5300: Compiler Construction
- ☐ Not Applicable

Rate your interest in computer programming (1-10): _____

Please mark all computer programming paradigms that you are proficient in:

- ☐ Imperative (Procedural) Programming
- ☐ Object Oriented Programming
- ☐ Visual Programming
- ☐ Functional Programming
- ☐ Logic Programming
- ☐ Declarative Programming
- ☐ Not Applicable - I do not know

Please estimate the number of hours you have spent in doing programming*:

_____

Are there any additional factors (personal or practical) that you feel have affected your programming abilities? If so, what are they?

_____

_____

Initials*: _____

- [ ] I certify that all information given in this application packet is accurate and true to the best of my knowledge. I understand that submission of false information is grounds for immediate dismissal from this study.

# APPENDIX J. PROBLEM SPACE MAP

# Question I

**Explicit:**

1. Task goal: Find two errors in a computer program
2. Requirements: The program must be able to select the greatest integer from three given values
3. Constraints: Not applicable
4. Instructions/standards: Not applicable

**Implicit:**

1. Relevant concepts:
    - Syntax error
    - Logic error
2. Knowledge:
    - Basic procedural programming language
    - Debugging procedure
3. Cognitive process:
    - Reading the provide code line-by-line
    - Understanding intMax algorithm:
        - It is a procedure
        - It receives three integer values
        - It returns one integer value
        - Variable max stores the greatest integer value

    - It compares all given values against max, and then rewrite max with the biggest value

- Finding errors in each line:
    - Misspelling: “Max” instead of “max”
    - Logic does not work as intended: using “==” instead of “=”
- Review the identified errors

# Question II

**Explicit:**

1. Task goal: Predict the program output for each input variation
2. Requirements:
    - Read given three inputs
    - Return a Boolean value
3. Constraints:
    - The first parameter is an integer between -10 to 10
    - The second parameter is an integer between -10 to 10
    - The third parameter is a Boolean value
4. Instructions/standards: Write each output in the provided box

**Implicit:**

1. Relevant concepts: Various procedural programming concepts
2. Knowledge: Basic procedural programming knowledge
3. Cognitive process:
    - Reading provided code line-by-line
    - Understanding the algorithm:
        - There is an if-statement that evaluate old_friend variable value
        - The return value depends on whether the given inputs are positive or negative
    - Simulate the program process line-by-line based on each given input variation:

    - Replace all variables with the associated input values line-by-line
    - Compute the result
    - Review the computation result
- Review whether all input variations have been simulated

# Question III

**Explicit:**

1. Task goal: Create a base class diagram of a digitalized modified monopoly game
2. Requirements:
    - There are 2 – 4 players
    - The player with most money after 20 turn wins
    - Each player needs to roll virtual dice to determine its movement
    - Each player must move each turn, and:
        - Each player can buy, sell, and improve building
        - Each player can use special ability
        - Each player can buy items in the shop
    - Character types:
        - King, Warrior, Merchant, and Thief
        - Each character type has unique special abilities
        - Each character type starts with different items and amount of money
    - The board has 30 spaces in a circle shape, where:
        - Some spaces have buildings
        - Some spaces have shops
        - Some spaces have special instructions
    - Building types:
        - Castle, Fortress, and Inn
        - Building’s properties change based on the improvement level

    - Each building can be bought and sold
    - Each building has special instructions which depend on its type and amount of improvement
  - Item types:
    - Sword, Potion, Horse, and others
    - Each item gives unique special benefits for each Character type
3. Constraints: The game ends after 20 turns
4. Instructions/standards:
  - Class diagram notation
  - Do not have to think about:
    - The game display or animation
    - The game play-testing
  - Improvise when possible

**Implicit:**

1. Relevant concepts: Various object-oriented concepts
2. Knowledge:
  - Object-oriented design
  - Class diagram notation
3. Cognitive process:
  - Developing overall understanding:
    - Identify the classes
    - Identify the classes' properties and their access levels

    - Identify possible user's interactions

- Identifying and selecting programming language: Java, C++, or others
- Developing class diagram, by defining:
    - Identified classes
    - Super-classes or approaches (e.g., to group all types of items)
    - Relationship between classes
    - Each class' properties, types, and access levels
    - Each class' methods, return types, and access levels
    - Getter and setter methods for all private properties in each class
    - Each character's special abilities
    - Each character's starting items and amount of money
    - Mechanics for setting up buildings
    - Each building type's properties
    - Mechanics for storing each building's owner
    - Mechanics for setting up spaces
    - Mechanics for setting up each building's special instructions
    - Mechanics for executing the special instruction
    - Mechanics for identifying the spaces where each player is on
    - Mechanics for rolling the dice and determining the number of dice
    - Mechanics for counting the turn
    - Mechanics for initializing all classes and the game loop
    - Mechanics for stopping the game

    - Mechanics for declaring the winner
- Defining each class' constructor
- The game plays improvements*:
    - Selecting a winner if two or more players have the same amount of money
    - Mechanics for specifying the number of players
    - Mechanics for attacking other players
    - Mechanics for attacking other areas
    - Using Mercian Twister instead of typical-random method for the dice
- The player and character classes' improvements*:
    - Adding player's name
    - Adding player's stats
    - Adding levels to characters' special abilities
    - Mechanics for items enhancement to characters' stats
    - Mechanics for Items enhancement to characters' special abilities
- The building class's improvement*:
    - Adding maintenance cost
    - Adding building levels
    - Mechanics for utilizing buildings' type and improvement
    - Mechanics for handling changes of buildings' types after an upgrade
    - Mechanics for ordering multiple upgrades
- Reviewing identified classes, properties, method names, and access level

- Reviewing identified relationship between classes

*) Examples

## Question IV

**Explicit:**

1. Task goal: Developing an algorithm that can calculate the sum of three given integers but will stop when 13 is found
2. Requirements: Accept three given integers
3. Constraints: If one of the values is 13, then that value and the values after it will not count toward the sum
4. Instructions/standards: Nothing specific

**Implicit:**

1. Relevant concepts: Various procedural programming concepts
2. Knowledge: Basic procedural programming knowledge
3. Cognitive process:
    - Understanding the expected behavior based on provided examples
        - Reading the examples
        - Simulating calculation procedure based on the examples
        - Simulating calculation procedure based on given input variations
    - Developing general model of the algorithm
    - Writing the algorithm:
        - Declaring a variable to store the total sum

    - Initializing the total sum variable with zero
    - Ensuring the function will return the total sum
    - For each given input:
        - Check if the input value is 13
        - If it is, return the current total sum
        - If it is not, add the input value to the total sum
    - Deciding mechanics for writing the if-else statements
- Reviewing the proposed algorithm using the examples

# Question V

**Explicit:**

1. Task goals: Developing a pseudo-code to simulate given situation and determine the best position for Josephus
2. Requirements:
    - The program starts by asking the number of people
    - All people stand in circle facing the center (i.e., the sword)
    - The person in the north most position start killing the person to its left (clockwise):
        - The first person represented in the program is the person at the north most position
        - The people can be assigned numerically clockwise
    - Repeat the following until only one person remains:
        - Pass the sword to the next living person on its left (clockwise)
        - The person with sword then kill the person to its left (clockwise)
    - Return the last position
3. Constraints: There are only 3 to 40 people
4. Instructions/standards: Nothing specific

**Implicit:**

1. Relevant concepts: Procedural programming concepts
2. Knowledge: Basic procedural programming knowledge

3. Cognitive process:
    - Understanding the expected behavior from the provided examples:
        - Reading the example
        - Bridging the example and given suicidal-procedure
    - Developing patterns by generating more examples:
        - Choosing the number of people
        - Simulating given suicidal-procedure
        - Finding patterns:
            - Must not be in the even position
            - In each turn half of the people disappear
            - The total number of people in each turn has a behavioral impact
    - Testing identified patterns
    - Identifying and selecting programming approach: imperative, object-oriented, or others
    - Identifying and selecting programming language: Java, C/C++, or others
    - Writing the algorithm based on the identified pattern[1]
    - Writing the algorithm by following the provided procedure[2]:
        - Creating a procedure: its name and return type
        - Selecting the best data type to represent the people
            - Primitive: Array, Pointers
            - Object: Linked List, Stack, Queue, Vector

    - Reading the number of people from the user
    - Storing the number of people as an integer
    - Initializing the people using the selected data type as the reference
    - Declaring and initializing variable to point to the person who hold the sword
    - Identifying, selecting, and implementing the best way to repeat the suicidal-procedure
        - Loop: for or while
        - Recursive + required parameters
    - Implementing the suicidal-procedure inside the repeater (e.g., loop)
        - Connecting the current situation to the next when reach the last person when the number of people is odd (i.e., put the first person at the end)
    - Identifying and implementing the best way to store the updated people list
        - Using existing people list
        - Creating new people list (e.g., when using Array)
    - If using recursive, identify the best condition that will stop the recursion
    - Reviewing the variable names and types
- Reviewing the algorithm

[1, 2]) mutually exclusive

# APPENDIX K. PROGRAMMING PROBLEM CHARACTERISTICS

## Characteristics of Question I: Locating the Errors

| | | |
|---|---|---|
| Structure | : | Structured |
| Complexity | : | There are two issues to solve, one function, and four variables within a dynamic subsystem. |
| Required knowledge | : | Foundation of Programming |
| Cognitive skills | : | Level 5.1: Evaluate – Checking: Detecting internal inconsistency within a process which required using factual, conceptual, and metacognitive knowledge. |
| Type | : | Troubleshooting |
| Author(s) | : | Coding Bat (http://codingbat.com) with some modifications by Andreas Febrian and the 2016 REU students. |
| Difficulty | : | 2.30 out of 10 with standard deviation of 1.25 |

## Characteristics of Question II: Outputs Prediction

| | | |
|---|---|---|
| Structure | : | Structured |
| Complexity | : | There are seven issues, one function, and three variables within a dynamic subsystem. |
| Required knowledge | : | Foundation of Programming |
| Cognitive skills | : | Level 3.1: Apply – Executing: Applying a procedure to a familiar task which required using factual, conceptual, procedural, and metacognitive knowledge. |
| Type | : | Algorithmic |
| Author(s) | : | Coding Bat (http://codingbat.com) with some modifications by Andreas Febrian and the 2016 REU students. |
| Difficulty | : | 3.88 out of 10 with standard deviation of 3.09 |

## Characteristics of Question III: Monopoly in the Middle-Ages

| | | |
|---|---|---|
| Structure | : | Ill-Structured |
| Complexity | : | There are at least 18 issues, 24 functions, and 22 variables within a dynamic system. |
| Required knowledge | : | Foundation of Programming, Object Oriented Programming |
| Cognitive skills | : | Level 6.2: Create – Planning: Devising steps to accomplish certain task which required using factual, conceptual, procedural, and metacognitive knowledge. |
| Types | : | Design: designing a system |
| Author(s) | : | Collaborative work between the 2016 REU students and |

| | | |
|---|---|---|
| | | Andreas Febrian. |
| Difficulty | : | 6.88 out of 10 with standard deviation of 2.47 |

## Characteristics of Question IV: Algorithm Generation

| | | |
|---|---|---|
| Structure | : | Structured |
| Complexity | : | There are three issues, one function, and, at least, three variables within a dynamic subsystem. |
| Required knowledge | : | Foundation of Programming |
| Cognitive skills | : | Level 6.3: Create – Producing: Making a product for a specific purpose which required using factual, conceptual, procedural, and metacognitive knowledge. |
| Types | : | Design: designing an algorithm |
| Author(s) | : | Coding Bat (http://codingbat.com) with some modifications by Andreas Febrian and the 2016 REU students. |
| Difficulty | : | 3.00 out of 10 with standard deviation of 1.50 |

## Characteristics of Question V: The Last Standing Man

| | | |
|---|---|---|
| Structure | : | Structured |
| Complexity | : | There are at least five issues, one function, and 4 to 41 variables within a dynamic subsystem. |
| Required knowledge | : | Foundation of Programming |
| Cognitive skills | : | Level 6.3: Create – Producing: Making a product for a specific purpose which required using factual, conceptual, procedural, and metacognitive knowledge. |
| Types | : | Design: designing an algorithm |
| Author(s) | : | Herika Hayurani with major modification by Andreas Febrian and the 2016 REU students. |
| Difficulty | : | 6.56 out of 10 with standard deviation of 1.94 |

# APPENDIX L. PROGRAMMING PROBLEM

## Question I: Locating the Errors

You are teaching an introductory course in C++ programming to a group of high school students. You give them an assignment in which they are to provide a function that will select the greatest integer among three given values. The students have freedom in how they choose to write their function as long as it works properly. One of your students thinks the assignment is too easy and turns it in to you before it is due, much sooner than you expected. You're surprised, but you take the paper anyway and check it for correctness:

```
1.   public int intMax(int a, int b, int c){
2.        int max = a;
3.        if(Max < b){max = b;}
4.        if(max < c){max == c;}
5.        return max;
6.   }
```

You notice that the code contains two errors. What are the errors?

## Question II: Outputs Prediction

There exists a magic black box that takes in three values to verify whether a friendship between two individuals is compatible. The first value should be an integer of value -10 through 10 chosen by the first person in the friendship. The second value should be an integer of value -10 through 10 chosen by the second person in the friendship. Finally, the third value should be a Boolean value. If the two individuals have been friends for more than three years, this Boolean value will be TRUE, but if they have been friends for less than three years, the Boolean value will be FALSE. After you input your three values, the magic black box will return TRUE if the friendship is compatible, or FALSE if the friendship isn't compatible.

Please carefully read the code for the magic black box below:

```
1.   public boolean blackBox(int a, int b, boolean old_friend){
2.        if(old_friend){return a < 0 && b < 0;}
3.        return (a < 0 && b > -1) || (a > -1 && b < 0);
4.   }
```

Using the algorithm in the code above, determine the compatibility output for each statement in the table below:

| No. | Statement | Answer |
|---|---|---|
| 1. | blackBox(5, -5, FALSE) | |
| 2. | blackBox(-6, 6, FALSE) | |
| 3. | blackBox(-5, 6, TRUE) | |
| 4. | blackBox(-5, -5, TRUE) | |

## Question III: Monopoly in the Middle-Ages

The game company that you work for has decided to develop a digital version of a classic board game. You have been assigned as their system designer. You are informed that other experts are in charge of the animation and play-testing, so these are not part of your duties. After three days of meetings, the people in charge have agreed on some basic aspects of the game, which are:

1. The game is meant to be played by either two, three, or four players.
2. Each player chooses to play as any one of the following characters: King, Warrior, Merchant, or Thief. Each character has unique special abilities, and starts with different items and different amounts of money.
3. The game board will consist of 30 spaces where players can land, arranged in a circle. On some spaces, there are buildings which can be bought and sold. On other spaces, there are shops where players can buy items. In addition, some spaces have special instructions that players must follow when they land there.
4. In the original board game, movement is determined by rolling dice, so you must develop an equivalent virtual method of determining the number of spaces each player moves on his or her turn.
5. On their turn, each player must move and they can choose to do any of the following: buy the building on the space they are on, sell any building they own, spend money to improve buildings they own, or use one of their character's special abilities.

6. Items give special benefits to the player. Items include the following: Sword, Potion, Horse, etc. The effects of the item will be different for each character type.
7. There are three different kinds of buildings: Castle, Fortress, and Inn. These buildings have different properties depending on how much the owner has spent on improving them.
8. When a player lands on a space with a building owned by someone else on it, then that player must follow certain special instructions, determined in part by the type of building, and also by the amount of improvements paid for by the owner.
9. The goal is to have the most money after each player has taken 20 turns.

As a system designer, you have been asked to create a complete base for this game that will allow the rest of the team members to easily develop the rest of the game. You have been told to use object oriented design, and specifically you must provide a detailed class diagram, which will accommodate all the given objectives and constraints. Your company has also requested that you go beyond the listed requirements when appropriate and use your creativity to produce a thorough and extensive design.

## Question IV: Algorithm Generation

In Western culture, there is an irrational fear surrounding the number 13. For example, according to the Stress Management Center and Phobia Institute in North Carolina, more than 80 percent of high rise buildings in the U.S. don't have a thirteenth floor. Because you believe in this superstition, you want to create a "Lucky Sum" method. The method (shown below) will return the sum of three given integer values. However, if one of the values is 13 then that value and the values to its right will not count toward the sum. So for example, if b is 13, then both b and c do not count.

```
1.   public int luckySum(int a, int b, int c){
2.        ...
3.   }
```

Here are three examples to show the method behavior:

| No. | Statement | Answer |
|---|---|---|
| 1. | luckySum(1, 2, 3) | 6 |
| 2. | luckySum(1, 2, 13) | 3 |
| 3. | luckySum(1, 13, 3) | 1 |

## Question V: The Last Standing Man

It was the time when Rome had conquered most of Europe. A religious group decided to rebel against the Roman Empire. The religious leader's call for actions inspired their young believers, one of whom was Josephus. He was a bright mathematician and historian with unshakable belief in justice and the power of their God. Long story short, after several months of fighting, the rebel group was being pushed outside of the city. Out of hundreds, only 3 to 40 people remained. They knew that their days were numbered. They had two options: to die at the hands of their comrades (suicide was not an option if they wanted to go to Heaven) or to be tortured by the Romans. After a long discussion, their leader decided that they would:

- Throw away all their swords, except one, which would be placed on the ground.
- Stand in a circle, around the single sword, with everyone facing the center.
- The person who was standing in the north-most position in the circle then took the sword.
- Repeat the following procedure:
    - The person with the sword killed the person on his left (clockwise).
    - That person then passed the sword to the next (living) person on his left (clockwise).
    - This process should be repeated until there was only one man left.

Josephus was not yet ready to die. He was a historian; he wanted to immortalize their (and his) story; he had to be the last person alive.

Your task is to develop a pseudo-code to simulate this depressing suicidal method and determine where Josephus should stand. Your pseudo-code will start by asking for the number of people in the group. You can represent each person with a number: start from one (1) which is assigned to the person who initially stands in the north-most position, and then assign the rest of the numbers clockwise from that person. You have to simulate each step and then determine Josephus' position. For example:

```
Number of people in the group: 5
1, 2, 3, 4, 5
3, 4, 5, 1
5, 1, 3
3, 5
3
```

Josephus should stand at the 3rd position.

# APPENDIX M. PROGRAMMING PROBLEM SOLUTION

## Solution for the Question I: Locating the Errors

This problem asks the participant to identify and locate two errors within a given code snippet. The two errors are in line three (see the ‘Max’ variable) and four (see the comparison syntax, ‘==’); also see the texts marked with red color below:

```
1.    public int intMax(int a, int b, int c){
2.         int max = a;
3.         if(Max < b){max = b;}
4.         if(max < c){max == c;}
5.         return max;
6.    }
```

In C++ and most programming (not scripting) languages, variable name is case sensitive, which means ‘max’ and ‘Max’ are two different variables. Since variables have to be declared prior usage, there are only four variables declared in the code snippet above, which are ‘a’, ‘b’, ‘c’, and ‘max’. In other word, variable ‘Max’ was never declared; variable ‘Max’ does not exist within the program. Therefore, the program has an error in line three, where it tried to access undeclared variable ‘Max’. This is the first error.
The second error is in line four, where the student tried to assign the value of variable ‘c’ to ‘max’. By following the logic of the program, it is clear that the student’s intent was to store the biggest value in variable ‘max’ (see the return statement at line five). In other words, in line four, instead of using the comparison syntax (‘==’), the student should use the assignment syntax (‘=’). Please note that the comparison syntax will compare the value of variable ‘max’ to ‘c’, which will return, a Boolean value, TRUE if both are the

same or FALSE if otherwise. On the other hand, the assignment syntax will put the value of variable ‘c’ to variable ‘max’.

## Solution for Question II: Outputs Prediction

This problem asks the participant to determine the outputs of four statements through evaluating a given ‘blackBox(int, int, bool)’ function. This problem only has a correct solution but can be approached in multiple ways, for example by using Boolean tables or creating an abstraction for each Boolean expression. Both approaches will be explained in the next section. The correct solutions for this problem are as follow:

| No. | Statement | Answer |
|---|---|---|
| 1. | `blackBox(5, -5, FALSE)` | TRUE |
| 2. | `blackBox(-6, 6, FALSE)` | TRUE |
| 3. | `blackBox(-5, 6, TRUE)` | FALSE |
| 4. | `blackBox(-5, -5, TRUE)` | TRUE |

**Using a Boolean Table**

In this section, the first statement (i.e., ‘blackBox(5, -5, FALSE)’) will be used to illustrate the Boolean table approach. By applying the value of variable ‘old_friend’ (i.e., FALSE) to the third line in the code snippet, one can determine that the Boolean expression in line three can be skipped and go to line four. Using the Boolean table approach, one will get:

| Step | Evaluation |
|---|---|
| Boolean expression in line four | (a < 0 && b > -1) \|\| (a > -1 && b < 0) |
| Replace each variable with its correspondent value | (5 < 0 && -5 > -1) \|\| (5 > -1 && -5 < 0) |
| Evaluate all innermost comparisons | (FALSE && FALSE) \|\| (TRUE && TRUE) |
| Evaluate all 'AND' statements in each section | (FALSE) \|\| (TRUE) |
| Evaluate the 'OR' statement | TRUE |

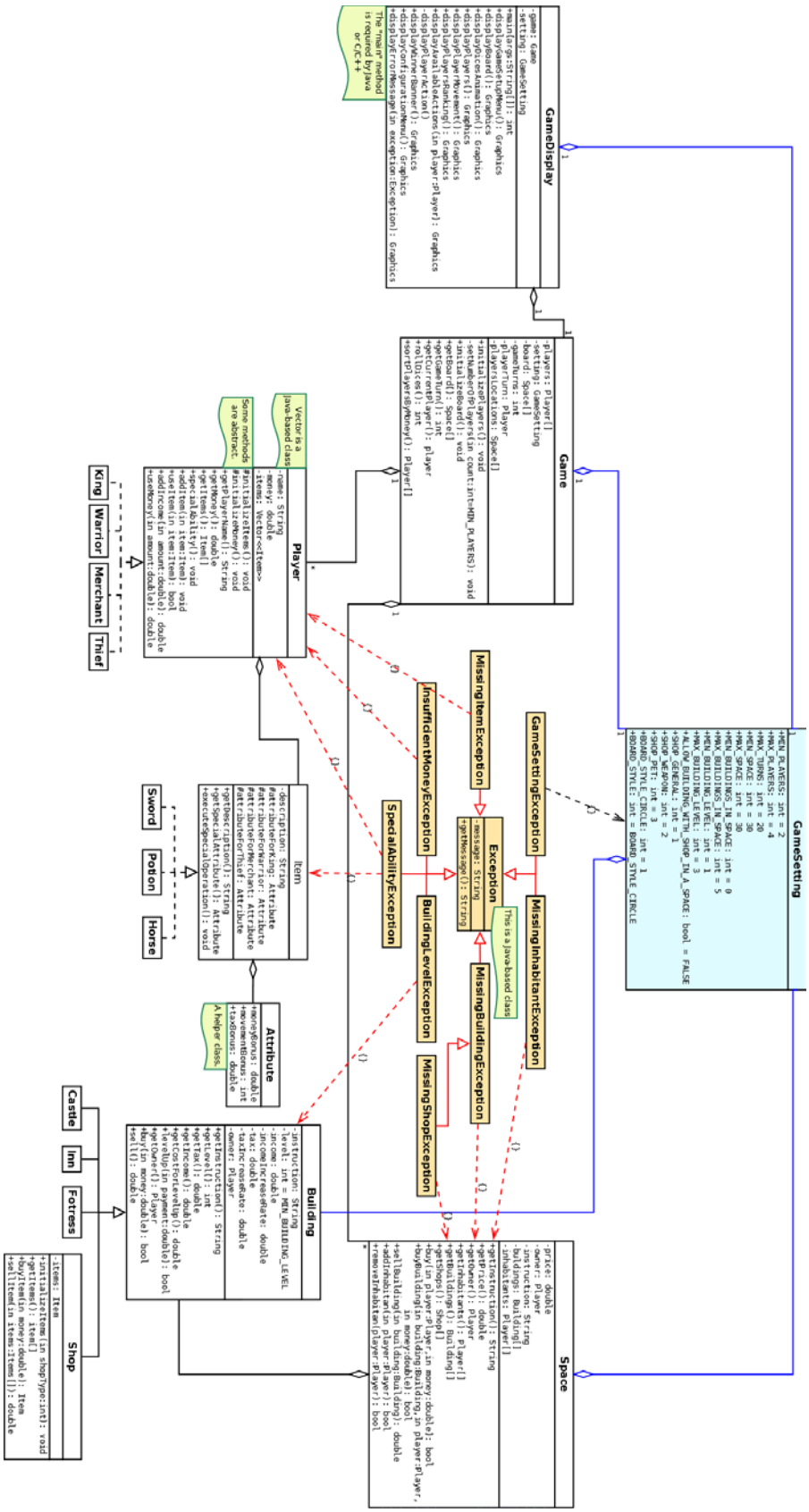
Therefore, the first statement, 'blackBox(5, -5, FALSE)', will yield TRUE.

**Using Abstraction of Boolean Expressions**

There are two Boolean expressions in this problem, which are in line three and four. The Boolean expression in line three will only be evaluated if the value of variable 'old_friend' was TRUE. The Boolean expression in line three is 'a < 0 && b < 0', which means if variable 'a' is less than zero and variable 'b' is less than zero, then the return will be TRUE, otherwise the return will be FALSE. In other words, this expression will only return TRUE if both variables 'a' and 'b' are negatives. Using this knowledge, we can infer that the third and fourth statements will yield FALSE and TRUE respectively.

## Solution for Question III: Monopoly in the Middle-Ages

This problem asks the participant to design an object oriented system based on given goals and constraints. Below is one of the possible solutions which utilize various object oriented concepts in the Java programming language.


GameDisplay
Game
GameSetting
Player
Space
Building
Item
Attribute
Shop
Exception
MissingItemException
InsufficientMoneyException
SpecialAbilityException
BuildingLevelException
MissingInhabitantException
MissingBuildingException
MissingShopException
GameSettingException
King
Warrior
Merchant
Thief
Sword
Potion
Horse
Castle
Inn
Fotress

## Solution for Question IV: Algorithm Generation

This problem asks the participant to design a function based on given criteria and constraints. Although the utilization of ‘IF’ statements are necessary, the solution for this problem is not unique. Here are three of them.

**First Solution**

This solution utilized the in-line ‘IF’ syntax where all statements will be executed.

```
1.   public int luckySum(int a, int b, int c) {
2.        int count = (a == 13) ? 0 : a;
3.        count += (a == 13 || b == 13) ? 0 : b;
4.        count += (a == 13 || b == 13 || c == 13) ? 0 : c;
5.        return count;
6.   }
```

**Second Solution**

This solution utilized nested ‘IF’ approach where not all statements will be executed; it depends on the values of variables ‘a’, ‘b’, and ‘c’.

```
1.   public int luckySum(int a, int b, int c) {
2.        int count = 0;
3.        if(a != 13){
4.             count += a;
5.             if(b != 13){
6.                  count += b;
7.                  if(c != 13){
8.                       count += c;
```

```
9.                  }
10.             }
11.       }
12.       return count;
13.   }
```

**Third Solution**

This solution utilized the combination of ‘IF’ and ‘return’ syntaxes where not all statements will be executed; it depends on the values of variables ‘a’, ‘b’, and ‘c’.

```
1.    public int luckySum(int a, int b, int c) {
2.          int count = a + b + c;
3.          if(a == 13){return 0;)
4.          if(b == 13){return a;}
5.          if(c == 13){return a + b;}
6.          return count;
7.    }
```

## Solution for Question V: The Last Standing Man

This problem asks the participant to determine the best location for Josephus so he can escape death. Here are three examples of the possible solutions.

**Using Arrays**

Although it is not straight forward, arrays can be used as a solution to this problem. Below is an example of such solution (please note, this solution assumed that the first index of an array is started with one (i.e., 1), not zero (i.e., 0)).

```
int best_position = 0; // store the best location for Josephus
// store the number of the Josephus’ comrade
int people = in(<std_in>);
// <std_in> means ask input from the user
// store Josephus’ comrades’ information
Array members[] = new Array[people];
// fill the array with integers from 1 to ‘people’
initialize(members, people);
int sword_position = 1; // the sword starting position
while(members.length > 1){
      print(members); // print all array elements
      if(sword_position % 2 == 1) // if it is odd number
           zeroingEvenIndexedData(members);
      else // if it is even number
           zeroingOddIndexedData(members);
      // adjust sword position based on the number of people
      sword_position = (members.length % 2 == 0) ? 1 : 2;
      // create new array by removing all zeros in ‘members’
      members = recreateMembersArray(members);
}
print(best_position);
```

Consequently, methods ‘initialize(Array, int)’, ‘zeroingEvenIndexedData(Array)’, ‘zeroingOddIndexedData(Array)’, and ‘recreateMembersArray(Array)’ should also be implemented.

**Using Queue**

Queue is a dynamic data type. Unlike array, it can add and adjust its length on the fly (i.e., when the program runs). Queue uses the FIFO (First In, First Out) principle, which means if one inserted (pushed) 3, 2, 1 to the queue, one would get 3, 2, and 1 when one ejects (pops) the queue three times. Most programming languages provide a built-in queue. Here is an example of such a solution:

```
int best_position = 0; // store the best location for Josephus
// store the number of the Josephus' comrade
int people = in(<std_in>);
// <std_in> means ask input from the user
Queue members = new Queue();
// fill the Queue with integers from 1 to 'people'
initialize(member, people);
while(members.size() > 1){
      print(members);
      // move the person who hold the sword to the back
      member.push(members.pop());
      member.pop(); // kill the next person
}
print(best_position);
```

Consequently, methods 'initialize(Queue, int)' and 'print(Queue)' should also be implemented.

**Using Double Linked List**

A Double Linked List is a dynamic data type. Unlike array, it can adjust (add and remove) its length on the fly (i.e., when the program runs). Unlike Queue or Stack, the Double Linked List does not follow the FIFO (First In, First Out) or the LIFO (Last In, First Out) principles. Each item in the list will be connected to the two other items, either on its right or left. Here is an example of such a solution:

```
int best_position = 0; // store the best location for Josephus
// store the number of the Josephus' comrade
int people = in(<std_in>);
// <std_in> means ask input from the user
DoubleLinkedList head<Integer> = new LinkedList<Integer>();
DoubleLinkedList head = initialize(head, people);
while(member.size() > 1){
      print(head);
      LinkedList temp = head.next;
      // remove connection to the next person
      // (kill the next person)
      head.next = temp.next; temp.next.prev = head;
      temp.next = null; temp.prev = null;
      // give the sword to the next living person.
      head = head.next;
}
print(best_position);
```

Consequently, methods ‘initialize(LinkedList, int)’ and ‘print(DoubleLinkedList)’should also be implemented.

# APPENDIX N. RESEARCH SCHEDULE

The dissertation study activity overview is presented in Figure N-1, and the detailed schedule is presented in Table N-1.

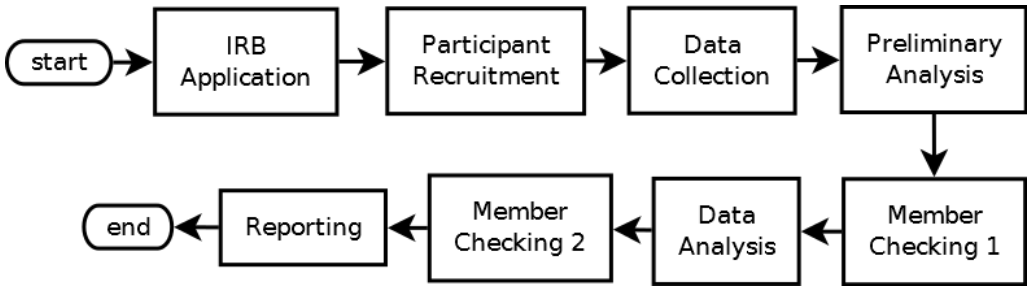



*Figure N-1. Research Activities Overview*

*Table N-1.*

Research Schedule

| **Research Activity** | **Month** | | | | | | |
|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| IRB Application | X | X | | | | | |
| Participant Recruitment | | X | X | | | | |
| Participant Selection | | | X | | | | |
| Data Collection | | | X | X | | | |
| Preliminary Analysis | | | X | X | | | |
| Member Checking 1 | | | X | X | | | |
| Data Analysis | | | | X | X | X | |
| Member Checking 2 | | | | | | X | |
| Reporting | | X | X | | X | X | X |

## APPENDIX O. IRB APPROVAL


# Institutional Review Board

USU Assurance: FWA#00003308


**Exemption #2**

## Certificate of Exemption

FROM:

Melanie Domenech Rodriguez, IRB Chair

Nicole Vouvalis, IRB Administrator

| | |
|---|---|
| To: | Oenardi Lawanto, Andreas Febrian |
| Date: | August 29, 2017 |
| Protocol #: | 8659 |
| Title: | Senior Computer Science Students' Task And Revised Task Interpretation While Engaged In Programming Endeavor |

The Institutional Review Board has determined that the above-referenced study is exempt from review under federal guidelines 45 CFR Part 46.101(b) category #2:

> Research involving the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures or observation of public behavior, unless: (a) information obtained is recorded in such a manner that human subjects can be identified, directly or through the identifiers linked to the subjects: and (b) any disclosure of human subjects' responses outside the research could reasonably place the subjects at risk of criminal or civil liability or be damaging to the subjects' financial standing, employability, or reputation.

This exemption is valid for three years from the date of this correspondence, after which the study will be closed. If the research will extend beyond three years, it is your responsibility as the Principal Investigator to notify the IRB before the study's expiration date and submit a new application to continue the research. Research activities that continue beyond the expiration date without new certification of exempt status will be in violation of those federal guidelines which permit the exempt status.

As part of the IRB's quality assurance procedures, this research may be randomly selected for continuing review during the three year period of exemption. If so, you will receive a request for completion of a Protocol Status Report during the month of the anniversary date of this certification.

In all cases, it is your responsibility to notify the IRB prior to making any changes to the study by submitting an Amendment/Modification request. This will document whether or not the study still meets the requirements for exempt status under federal regulations.

Upon receipt of this memo, you may begin your research. If you have questions, please call the IRB office at (435) 797-1821 or email to irb@usu.edu.

The IRB wishes you success with your research.

# APPENDIX P. PERSONALIZED TASK INTERPRETATION REPORTS

# Jake's Task Interpretation Report

Hi Jake,

Thank you for your participation in the study on students' task interpretation and its revision during a programming endeavor. I am happy to share a personalized self-regulated learning report related to your task interpretation for Monopoly in the Middle-Ages (i.e., creating a class diagram) and the Last Standing Man (i.e., writing pseudocode to simulate a depressing suicidal method and determine where Josephus should stand) problems.

First, a summary of self-regulation and task interpretation. Self-regulation is a common sense, such that before solving a programming problem, you need to understand the problem itself, then you make a plan to solve it according to your understanding, then you execute your plan, then you monitor your progress and approach, and adjust your strategies as needed. The heart of your self-regulation is your task interpretation or understanding of the problem. Task interpretation is crucial because when solving a programming problem (or any task), your approach to solve it is informed by your understanding of that problem. Therefore, if you have an incorrect task interpretation, you may end up using wrong strategies or even fail to solve the problem. Fortunately, your task interpretation evolves during the learning or problem-solving endeavor.

Overall, you have shown an excellent performance- and mastery-driven (e.g., utilizing various programming best practices) self-regulation skills that mimics the experts' behaviors during the data collection, such as:

- You were capable of adjusting your problem-solving approach according to the problem type.
- You were competent in identifying the problem-goal, requirements, constraints, and relevant knowledge and skills.
- Given enough time, you were competent in identifying and extrapolating the problem requirements and constraints.
- You were proficient in identifying the most appropriate problem-solving steps based on your understanding of the problem.
- You were able to balance your two drivers (i.e., performance and mastery) during the problem-solving endeavor.

When solving the third problem, you were observed using entity-relationship diagram notation instead of the class diagram. Although this decision was a performance-oriented accommodation that enabled you to solve the problem during the data collection, this decision prevented you from addressing some design details, such as specifying mechanics for declaring the winner or determining the access level (e.g., public or private) of the classes' properties and methods. This might be an area for consideration during future problem-solving endeavors.

Some possible improvements are observed based on your approach to solving the Last Standing Man problem. You were observed to inaccurately interpret the goal of this problem, in such you did not seem able to identify that the problem asked to simulate the given procedure and provide a print out of each program state. As a result, you were drawing strategies from inaccurate experience (i.e., Discrete Mathematics) and utilizing

inappropriate problem-solving approach. It was plausible you were overconfident in the relationship between this problem and the Discrete Mathematics problems, which then prevented you from checking whether the association itself was correct; this phenomenon is also known as confidence bias. Unfortunately, aside from improving your self-awareness and self-monitoring, the literature does not suggest any other strategies to overcome it. However, you might want to reflect on Rusty's experience:

> Out of four participants, Rusty was the only student who defeated his confidence bias when working on the Last Standing Man problem. Rusty's awareness on the stagnancy of his problem-solving endeavor, and he often misses essential small details when interpreting a problem, inspired him to question whether his task understanding was accurate.

Please let me know your comments on this report. Also, let me know if you want to read an elaborated analyses of your problem-solving endeavor.

# Rusty's Task Interpretation Report

Hi Rusty,

Thank you for your participation in the study on students' task interpretation and its revision during a programming endeavor. I am happy to share a personalized self-regulated learning report related to your task interpretation for Monopoly in the Middle-Ages (i.e., creating a class diagram) and the Last Standing Man (i.e., writing pseudocode to simulate a depressing suicidal method and determine where Josephus should stand) problems.

First, a summary of self-regulation and task interpretation. Self-regulation is common sense, in that before solving a programming problem, you need to understand the problem itself, then you make a plan to solve it according to your understanding, then you execute your plan, then you monitor your progress and approach, and adjust your strategies as needed. The heart of your self-regulation is your task interpretation or understanding of the problem. Task interpretation is crucial because, when performing any task such as solving a programming problem, your approach to solve it is informed by your understanding of that problem. Therefore, if you have an incorrect task interpretation, you may end up using ineffective strategies and failing to solve the problem. Fortunately, your task interpretation evolves during the learning or problem-solving endeavor.

Overall, you have shown an excellent performance- and mastery-driven (e.g., utilizing various design pattern) self-regulation skills that mimic the experts' behaviors during the data collection, such as:

- You were capable of adjusting your problem-solving approach according to the problem type.
- You were competent in identifying the problem-goal, requirements, constraints, and relevant knowledge and skills.
- Given enough time, you were competent in identifying and extrapolating the problem requirements and constraints.
- You were proficient in identifying the most appropriate problem-solving steps based on your understanding of the problem.
- You were able to balance your two drivers (i.e., performance and mastery) during the problem-solving endeavor.

Out of four participants, you were the only student who could correctly interpret the Last Standing Man problem. Similarly to other participants, you were observed to inaccurately interpret the goal of this problem as you did not identify that the problem asked you to simulate the given procedure and provide a print out of each program state. As a result, you were drawing strategies from inaccurate experience (i.e., Discrete Mathematics) and utilizing inappropriate problem-solving approach. However, your awareness on the stagnancy of your problem-solving endeavor, and that you often miss essential small details when interpreting a problem, inspired you to question whether your task understanding was accurate.

When solving the third problem, you were observed addressing the details of special abilities, mechanics for virtual dice, Items benefit for the Characters, and limiting the number of players, board spaces, and turns but forgot to integrate them in your class

diagram. Theoretically, it was possible that your extensive problem-solving engagement combined with the limited working memory space, made you forget these design details. Such situation can be mitigated by being more sensitive to your intermediate design decisions and improving your self-monitoring and note-taking skills.

A possible improvement is observed based on your approach to solving the Last Standing Man problem. You were observed assuming you could identify useful patterns. Your approach suggests you did not consider the follow-up actions if you were not able to find the pattern; this might be an area of consideration for you. Further, it might be beneficial to enrich your known problem-solving approaches, not only for this problem type but others, so you do not have to improvise when your problem-solving attempt seems not to be working.

Please let me know your comments on this report. Also, let me know if you want to read more detailed analyses of your problem-solving activities.

# Anne's Task Interpretation Report

Hi Anne,

Thank you for your participation in the study on students' task interpretation and its revision during a programming endeavor. I am happy to share a personalized self-regulated learning report related to your task interpretation for Monopoly in the Middle-Ages (i.e., creating a class diagram) and the Last Standing Man (i.e., writing pseudocode to simulate a depressing suicidal method and determine where Josephus should stand) problems.

First, a summary of self-regulation and task interpretation. Self-regulation is a common sense, such that before solving a programming problem, you need to understand the problem itself, then you make a plan to solve it according to your understanding, then you execute your plan, then you monitor your progress and approach, and adjust your strategies as needed. The heart of your self-regulation is your task interpretation or understanding of the problem. Task interpretation is crucial because when solving a programming problem (or any task), your approach to solve it is informed by your understanding of that problem. Therefore, if you have an incorrect task interpretation, you may end up using wrong strategies or even fail to solve the problem. Fortunately, your task interpretation evolves during the learning or problem-solving endeavor.

Overall, you have shown an excellent performance-driven self-regulation skill that mimic the experts' behaviors during the data collection, such as:

- You were capable of adjusting your problem-solving approach according to the problem type.

- You were competent in identifying the problem-goal, requirements, constraints, and relevant knowledge and skills.
- Given enough time, you were competent in identifying and extrapolating the problem requirements and constraints.
- You were proficient in identifying the most appropriate problem-solving steps based on your understanding of the problem.

Although you acknowledged that creativity is not one of your strengths, investing some effort to enrich your programming style, known algorithms, known design patterns, and various problem-solving approaches might be beneficial as studies suggest a close relationship between computer programming and creativity.

Aside from creativity, some possible improvements are observed based on your approach to solving the Last Standing Man problem. You were observed to inaccurately interpret the goal of this problem, in such you did not seem able to identify the problem asked to simulate the given procedure and provide a print out of each program state. As a result, you were drawing strategies from inaccurate experience (i.e., Discrete Mathematics) and utilizing inappropriate problem-solving approaches. It was plausible you were overconfident in the relationship between this problem and the Discrete Mathematics problems, which then prevented you from checking whether the association itself was correct; this phenomenon is also known as confidence bias. Unfortunately, aside from improving your self-awareness and self-monitoring, the literature does not suggest any other strategies to overcome it. However, you might want to reflect on Rusty's experience:

> Out of four participants, Rusty was the only student who defeated his confidence bias when working on the Last Standing Man problem. Rusty's awareness on the stagnancy of his problem-solving endeavor, and he often misses essential small details when interpreting a problem, inspired him to question whether his task understanding was accurate.

Further, related to your approach to solving this problem, you were observed assuming you could identify useful patterns. Your approach suggests you did not consider the follow-up actions if you were not able to find the pattern; this might be an area of consideration for you. Further, it might be beneficial to enrich your known problem-solving approaches, not only for this problem type but others, so you do not have to improvise when your problem-solving attempt seems not to be working.

Please let me know your comments on this report. Also, let me know if you want to read an elaborated analyses of your problem-solving endeavor.

# LStew's Task Interpretation Report

Hi LStew,

Thank you for your participation in the study on students' task interpretation and its revision during a programming endeavor. I am happy to share a personalized self-regulated learning report related to your task interpretation for Monopoly in the Middle-Ages (i.e., creating a class diagram) and the Last Standing Man (i.e., writing pseudocode to simulate a depressing suicidal method and determine where Josephus should stand) problems.

First, a summary of self-regulation and task interpretation. Self-regulation is common sense, in that before solving a programming problem, you need to understand the problem itself, then you make a plan to solve it according to your understanding, then you execute your plan, then you monitor your progress and approach, and adjust your strategies as needed. The heart of your self-regulation is your task interpretation or understanding of the problem. Task interpretation is crucial because, when performing any task such as solving a programming problem, your approach to solve it is informed by your understanding of that problem. Therefore, if you have an incorrect task interpretation, you may end up using ineffective strategies and failing to solve the problem. Fortunately, your task interpretation evolves during the learning or problem-solving endeavor.

Overall, you have shown an excellent performance- and mastery-driven (e.g., utilizing various design pattern) self-regulation skills that mimic the experts' behaviors during the data collection, such as:

- You were capable of adjusting your problem-solving approach according to the problem type.
- You were competent in identifying the problem-goal, requirements, constraints, and relevant knowledge and skills.
- Given enough time, you were competent in identifying and extrapolating the problem requirements and constraints.
- You were proficient in identifying the most appropriate problem-solving steps based on your understanding of the problem.
- You were able to balance your two drivers (i.e., performance and mastery) during the problem-solving endeavor, except for the last problem (i.e., the Last Standing Man).

When solving the third problem, you were observed addressing the mechanics to store building's owner and identify the player's location on the board but forgot to integrate them in your class diagram. Theoretically, it was possible that your extensive problem-solving engagement combined with the limited working memory space, made you forget these design details. Such situation can be mitigated by being more sensitive to your intermediate design decisions and improving your self-monitoring and note-taking skills.

Some possible improvements are observed based on your approach to solving the Last Standing Man problem. You were observed to inaccurately interpret the goal of this problem as you did not identify that the problem asked to simulate the given procedure and provide a print out of each program state. As a result, you were drawing strategies

from inaccurate experience (i.e., Discrete Mathematics) and utilizing inappropriate problem-solving approach. It was plausible you were overconfident in the relationship between this problem and the Discrete Mathematics problems. This overconfidence may have prevented you from checking whether the association itself was correct; this phenomenon is also known as confidence bias. Unfortunately, aside from improving your self-awareness and self-monitoring, the literature does not suggest any other strategies to overcome this. However, you might want to reflect on Rusty's experience:

> Out of four participants, Rusty was the only student who defeated his confidence bias when working on the Last Standing Man problem. Rusty's awareness on the stagnancy of his problem-solving endeavor, and he often misses essential small details when interpreting a problem, inspired him to question whether his task understanding was accurate.

Further, related to your approach to solving this problem, you were observed assuming you could identify useful patterns. Your approach suggests you did not consider the follow-up actions if you were not able to find the pattern; this might be an area of consideration for you. Further, it might be beneficial to enrich your known problem-solving approaches, not only for this problem type but others, so you do not have to improvise when your problem-solving attempt seems not to be working.

Please let me know your comments on this report. Also, let me know if you want to read more detailed analyses of your problem-solving activities.

# CURRICULUM VITAE

**Andreas Febrian**

http://id.linkedin.com/pub/andreas-febrian/32/597/b0

Researcher ID: C-2716-2016, OrcID: 0000-0003-0746-242X

## Formal Educations

| Year | GPA | Information |
|---|---|---|
| 2014 – 2018 | 3.94 | Doctoral in Engineering Education at Utah State University (http://www.eed.usu.edu/)<br>Dissertation: Senior Computer Science Students' Task and Revised Task Interpretation while Engaged in Programming Endeavor |
| 2008 – 2010 | 3.40 | Master in Computer Science at Universitas Indonesia (http://www.cs.ui.ac.id/)<br>Thesis: Implementation of Al-Fath Robot, Movement Limit For Charged Robot, and Main Spread 2 in Simulation of Robots for Odor Sources Localization. |
| 2003 – 2008 | 2.90 | Bachelor in Computer Science at Universitas Indonesia (http://www.cs.ui.ac.id/)<br>Thesis: Evaluation of Architecture Component Selection on a Single Bus Microprocessor. |

## Teaching Experience

| Semester | Course | Institution |
|---|---|---|
| 2013 Fall | Algorithm and Programming 1<br>Mobile Programming | Faculty of Information Technology, Universitas YARSI |
| 2013 Spring | Algorithm and Programming 2<br>Web Programming | Faculty of Information Technology, Universitas YARSI |
| 2012 Fall | Algorithm and Programming 1<br>Mobile Programming | Faculty of Information Technology, Universitas YARSI |

## Research Experience

| Period | Experience |
|---|---|
| January 2014 – July 2018 | Cognitive and Metacognitive Activities in Engineering Design Education - Conducted at Utah State University |
| August 2017 - April 2018 | Senior Computer Science Students' Task and Revised Task Interpretation while Engaged in a Programming Endeavor - Conducted at Utah State University |
| June – August 2016 | Self-Regulated Learning of Computer Science Students While Engaged in Programming Design - Conducted at Utah State University |
| June – August 2016 | Experiences of Undergraduate Students in Engineering Education Research - Conducted at Utah State University |
| April – August 2015 | The Role of Self-Regulation in Problem-Solving Activities using Computational Thinking Strategies: A Preliminary Study - Conducted at Utah State University |

| Period | Experience |
|---|---|
| April – August 2014 | Developing a Self-Regulation Survey Instrument for Problem Solving in Engineering: An Exploratory Mixed-Methods Design - Conducted at Utah State University |
| August – December 2013 | Developing Non-Conventional (IT-based) Instructions for Mobile Programming Course - Conducted at Universitas YARSI |
| August 2008 – September 2010 | Developing Swarm Robots for Gas Leakage and Boom Source Localization in an Industrial Environment - Conducted at Universitas Indonesia |

## Academic Experiences

| Period | Experience |
|---|---|
| August 1, 2012 – December 11, 2013 | Research and Teaching Faculty at Faculty of Information Technology, Universitas YARSI. |
| June – August 2012 | Teaching Assistant Coordinator for Introduction to Digital Systems at Faculty of Computer Science, Universitas Indonesia. |
| September 2011 – January 2012 | Teaching Assistant Coordinator and External Affairs Officer at "Lembaga Asisten," Faculty of Computer Science, Universitas Indonesia. |
| February 7 - June 3, 2011 | Teaching Assistant Coordinator for Introduction to Digital System, Faculty of Computer Science, Universitas Indonesia. |
| February – June 2010 | Teaching Assistant for Web Design and Programming, Faculty of Computer Science, Universitas Indonesia. |
| February 1 - July 31, 2010 | Technical Assistant for Accreditation and Curriculum Revision, Faculty of Computer, Science Universitas Indonesia. |
| September 2008 – January 2009 | Teaching Assistant for Introduction to Digital Systems, Faculty of Computer Science, Universitas Indonesia. |

| Period | Experience |
|---|---|
| June 2008 – August 2008 | Teaching Assistant for Web Design and Programming, Faculty of Computer Science, Universitas Indonesia. |
| August 29, 2005 - January 6, 2006 | Teaching Assistant for Introduction to Digital Systems, Faculty of Computer Science, Universitas Indonesia. |
| August 30, 2004 - January 7, 2005 | Teaching Assistant for Introduction to Digital Systems, Faculty of Computer Science, Universitas Indonesia. |

## Training

| Date | Title and Organizer |
|---|---|
| February 6, 2018 | Preparing for a Career Outside of Academia<br>Utah State University |
| March 6, 2016 | Broader Impacts: How to Include them in My Proposal<br>Utah State University |
| February 10, 2016 | Data Management Workshop<br>Utah State University Research of Graduate Studies |
| November 18, 201 | Prepare for Your Career in Academia<br>Utah State University Research of Graduate Studies |
| October 21, 2015 | Agile Teaching and Learning<br>A workshop in the 2015 Frontiers in Education Annual Conference |
| October 14, 2015 | 3 Most Effective Tactics to Improve Your Teaching<br>Utah State University Research of Graduate Studies |
| September 24, 2015 | Getting Started as a Successful Proposal<br>Utah State University Research of Graduate Studies |
| March 19, 2015<br>1:00 PM - 2:00 PM | How to Design Stunning Poster<br>Utah State University Research of Graduate Studies |

| Date | Title and Organizer |
|---|---|
| October 13 - November 3, 2012 | Training Java™ Enterprise Edition<br>Brainmatics - IT Training and Consulting |
| July 26 - 30, 2010 | Low-Cost Wireless Computer Networking<br>Faculty of Computer Science Universitas Indonesia and International Center for Theoretical Physics |
| August 4 - 15, 2009 | DAAD International summer School ”Computational Logic and Its Applications”<br>DAAD, Fasilkom UI, and Technische Universität Dresden |
| June 12 - 24, 2008 | brew™ Application Seminar and Workshop<br>Qualcomm, mobile8, and Jatis Mobile |

## Professional Experience

| Period | Information |
|---|---|
| February 2017 – August 2018 | Flash developer, Engineering Education Department, Utah State University. |
| May 2015 – August 2018 | Scholarship in Science, Technology, Engineering, and Mathematics (S-STEM) website and online application developer, Engineering Education Department, Utah State University (see http://s-stem.usu.edu). |
| February 2014 - November 2016 | REU Site Program in Engineering Education website and online application developer, Engineering Education Department, Utah State University (see http://reu.usu.edu). |
| November 2011 - May 2013 | Member of e-Indonesian Government Interoperability Framework (e-IGIF) Developer Team and a freelance translator of United Nations Asian and Pacific Training Center First Premier Series. |

| Period | Information |
|---|---|
| October - December 2010 | Technical Committee of 2010 International Conference on Computer Science and Information Systems, Faculty of Computer Science, Universitas Indonesia. |
| February 2010 | Formal Method Laboratory website developer, Faculty of Computer Science, Universitas Indonesia |
| July 2 – October 31, 2007 | System information developer, Universitas Indonesia. |
| 2005 | Instructor for "A Day Workshop: Developing a Dynamic and Interactive Website" |

## Leadership and Service Experience

| Period | Role and Organization |
|---|---|
| November 21, 2017 | Speaker at Universitas YARSI, Jakarta, Indonesia. Topic: “Educational Research” |
| October 2016 – October 2017 | President of a non-profit religious organization |
| 2015 – 2016 | Webmaster of Student Chapter of American Society of Engineering Education, Utah State University |
| March – August 2015 | Committee member of a non-profit religious organization |
| 2014 – 2015 | Secretary of Student Chapter of American Society of Engineering Education Utah State University |
| February 7, 2013 | Instructor for LaTeX Tutorial for Instructors, Faculty of Information Technology, Universitas YARSI. |
| January 29, 2013 | Instructor for LaTeX Tutorial for Students, Faculty of Information Technology, Universitas YARSI. |
| November 13, 2011 | Instructor for LaTeX Tutorial, Faculty of Computer Science Universitas Indonesia. |

| Period | Role and Organization |
|---|---|
| April 16, 2010 – now | Member of Iluni Faculty of Computer Science, Universitas Indonesia |
| August 2005 –2006 | Coordinator of Academic Mentorship, Faculty of Computer Science, Universitas Indonesia. |
| 2004 –2005 | Member of Department of Kreasi Mahasiswa (Students Creativity) of Universitas Indonesia Student Body<br>Deputy of Pengembangan Masyarakat (Social Development) of Faculty of Computer Science Universitas Indonesia Student Body<br>Member of Pengembangan Sumber Daya Manusia (Human Development) Department of Student Religious Organization |

## Professional Affiliations and Services

| Date/Period | Organization |
|---|---|
| 2014 – 2018 | Member of American Society of Engineering Education<br>Interest: Computers in Education, Computing & Information Technology, Multidisciplinary Engineering, Software Engineering<br>Constituent Committee, Student, Community Engagement Division, Engineering Leadership Development Division |
| 2014 – 2018 | Member of Student Chapter of American Society of Engineering Education, Utah State University |
| April 25, 2015 | Volunteer for inspirational class organized by ‘Indonesia Mengajar’ |

# Publications

## Books

Jatmiko, W., **Febrian, A.,** Jovan, F., Eka Suryana, M., Sakti Alvisalim, M., & Insani, A. (2010). Swarm Robot dalam Pencarian Sumber Asap (in English: Swarm Robots for Odor Source Localization). Depok, West Java, Indonesia: UI Press.

Jatmiko, W., **Febrian, A.**, & Salsabila, S. (2010). Robot Lego Mindstorm: Teori and Praktek (in English: Lego Mindstorm Robot: Theory and Application). Depok, West Java, Indonesia: UI Press.

## Journals

Lawanto, O., Minichiello, A., Uziak, J., & **Febrian, A**. (2018). Students' Problem Understanding in Engineering Problem Solving (Under Review). International Education Studies.

Purnomo, Jati, D. M., Arinaldi, A., Priyantini, D. T., Wibisono, A., & **Febrian, A.** (2016). Implementation of Serial and Parallel Bubble Sort on FPGA. Jurnal Ilmu Komputer Dan Informasi, 9(2), 113–120.

Diana, N. E., Nurmaya, & **Febrian, A.** (2016). Understanding Student Emotion in Video-Based Learning: Case Study - Programming Course. Journal of Teaching and Education, 5(1), 641–647. Retrieved from http://universitypublications.net/jte/0501/pdf/U5K394.pdf

**Febrian, A.**, & Murni, A. (2012). Usulan Pengembangan Sistem BALITAROT untuk Mendukung Perencanaan Berkelanjutan (In English: BALITAROT System Development Proposal to Support Sustainable Planning). Jurnal Sistem Informasi, 5(2), 121–131.

Jatmiko, W., Pambuko, W., **Febrian, A.**, Mursanto, P., Kusumoputro, B., Sekiyama, K., & Fukuda, T. (2010). Ranged subgroup particle swarm optimization for localizing multiple odor sources. International Journal of Smart Sensing and Intelligent Systems, 3(3).

**Conferences**

**Febrian, A.**, Lawanto, O., Peterson-Rucker, K., Melvin, A., & Guymon, S. E. (2018). Does Everyone Use Computational Thinking? - A Case Study of Art and Computer Science Majors. In Proceedings of the 2018 ASEE Annual Conference & Exposition (p. 15). Salt Lake City, UT, USA.

Lawanto, O., & **Febrian, A.** (2017). Students' Self-Regulated Learning Deficiencies During the Capstone Design Course. In 7th World Engineering Education Forum 2017 (pp. 1–5). Kuala Lumpur, Malaysia: IEEE. Retrieved from http://www.weef2017.org/

Lawanto, O., & **Febrian, A.** (2017). Students' Self-Regulation in a Senior Capstone Design Context: A Comparison between Mechanical and Biological Engineering Design Projects. In Proceedings of the 2017 ASEE Annual Conference & Exposition (p. 22). Columbus, Ohio, USA.

Lawanto, O., & **Febrian, A.** (2017). Students' Self-Regulation in Senior Capstone Design Projects. In Proceedings of the 2017 ASEE Annual Conference & Exposition. Columbus, Ohio, USA.

Lawanto, O., & **Febrian, A.** (2016). Student Self-Regulation in Capstone Design Courses: A Case Study of Two Project Teams. In Frontiers in Education Conference (FIE). IEEE

Lawanto, O., Cromwell, M., & **Febrian, A.** (2016). Students' Self-Regulation in Managing Their Capstone Senior Design Projects. In 123rd ASEE Annual Conference and Exposition. New Orleans, LA, USA.

**Febrian, A.**, Lawanto, O., & Cromwell, M. (2015). Advancing Research on Engineering Design using e-Journal. In Frontiers in Education Conference (FIE), 2015. 32614 2015. IEEE (pp. 1–5). El Paso, TX: IEEE. http://doi.org/10.1109/FIE.2015.7344191

Lawanto, O., Cromwell, M., & **Febrian, A.** (2015). Engineering Design Journey and Project Management. In Frontiers in Education Conference (FIE), 2015. 32614

2015. IEEE (pp. 1–5). El Paso, TX: IEEE. http://doi.org/10.1109/FIE.2015.7344208

Ortiz, C., Pham, B., Lawanto, O., & **Febrian, A.** (2014). Developing a Survey Instrument to Measure Problem Perception, Task Interpretation, and Planning Strategies within Self – Regulated Learning, a Work in Progress. In 2014 SACNAS National Conference. Los Angeles, California, USA.

Satwika, I. P., Habibie, I., Ma'sum, M. A., **Febrian, A.**, & Budianto, E. (2014). Particle Swarm Optimation based 2-Dimensional Randomized Hough Transform for Fetal Head Biometry Detection and Approximation in Ultrasound Imaging. In 2014 International Conference Advanced Computer Science and Information Systems (ICACSIS) (pp. 468–473). IEEE.

**Febrian, A.**, Matul, I. E., & Agus, S. (2011). Building Automation Tools to Calculate Trichloroethylene Level in Human Liver Using ? Case Study: Images of White Mouse Liver. In *22nd MHS2011 & Micro-Nano Global COE*.

Ferdian, J., S, D. R. Y., Sakti, A. M., Jatmiko, W., Fanany, M. I., **Febrian, A.**, Sekiyama, K., & Fukuda, T. (2010). Real Multiple Mobile Robots Implementation of PSO Algorithm for Odor Source Localization. In International Conference on Advanced Computer Science and Information Systems 2010 (pp. 253 –258). Bali, Indonesia.

Jatmiko, W., Fanany, M. I., **Febrian, A.**, Pambuko, W., Nugraha, A., Alvissalim, M. Sakti, Jovan, F., … Fukuda, T. (2010). Modified PSO Algorithm for Odor Source Localization Problems: Progress and Challenge. In International Conference on Advanced Computer Science and Information Systems 2010 (pp. 231–238). Bali, Indonesia.

Jatmiko, W., Nugraha, A., Pambuko, W., Kusumoputro, B., & **Febrian, A.** (2009). Localizing Multiple Odor Sources in Dynamic Environment Using Niche PSO with Flow of Wind Based on Open Dynamics Engine Library. In Second International Conference on IT Application and Management. Retrieved from http://www.wseas.us/e-library/transactions/systems/2009/32-562.pdf

Mursanto, P., **Febrian, A.**, Jatmiko, W., & Yuniarto, A. (2009). Performance Evaluation of Single Bus Microprocessor Architecture. In International Conference on Advanced Computer Science and Information Systems 2009. Depok, West Java, Indonesia.

**Workshop**

Lawanto, O., Butler, D. L., **Febrian, A.**, & Froyd, J. E. (2017). Self-regulated Learning in Engineering Design Projects Workshop. Columbus, Ohio, USA: 2017 ASEE Annual Conference & Exposition.

**Theses**

**Febrian, A.** (2010). Implementasi Model Robot Al-Fath, Pembatasan Gerak Robot Bermuatan, dan Main Spread 2 Pada Perangkat Simulasi Robot Pencari Sumber Asap (In English: Implementation of Al-Fath Robot, Movement Limit For Charged Robot, and Main Spread 2 in Simulation of Robots for Odor Sources Localization). Depok, West Java, Indonesia: Faculty of Computer Science Universitas Indonesia.

**Febrian, A.** (2009). Evaluasi Pemilihan Komponen Arsitektur Terhadap Kinerja Mikroprosesor Bus Tunggal (In English: Evaluation of Architecture Component Selection on a Single Bus Microprocessor). Depok, West Java, Indonesia: Faculty of Computer Science Universitas Indonesia.

**Others**

Jatmiko, W., Nugraha, A., Pambuko, W., & **Febrian, A.** (2010). User Guide: Manual Simulator Robot Pencari Sumber Asap (Version 1.12). Depok, West Java, Indonesia.

## Grants, Awards, and Honors

| Date | Information |
|---|---|
| February 2017 | Graduate Researcher of the Year, Engineering Education Department, Utah State University |
| October 2016 | Travel Grant of $500 from Utah State University Engineering Education Department for attending Frontier in Education Annual Conference in Erie, Pennsylvania. |
| October 2016 | Travel Grant of $300 from Utah State University Graduate School for attending Frontier in Education Annual Conference in Erie, Pennsylvania. |
| October 2015 | Travel Grant of $300 from Utah State University Graduate School for attending Frontier in Education Annual Conference in El Paso, Texas. |
| December 2013 | Research Grant of 10 million IDR from Universitas YARSI.<br>Title: Developing an Emotion Detection System to Improve Students' Mental Health - A Case Study in Programming Course |
| September 2013 | Development Grant of 25 million IDR from the Indonesian Higher Education Department.<br>Title: Developing Non-Conventional (IT-based) Instructions for a Mobile Programming Course |
| April 2013 | Research Grant of 95 million IDR from the Indonesian Higher Education Department.<br>Title: Developing a Smart Programming Learning Environment based on Affective Computing |
| December 8, 2009 | Best Session Presentation at the International Conference on Advanced Computer Science and Information Systems 2009 |