CHARACTERIZING STUDENT PROFICIENCY IN SOFTWARE MODELING IN TERMS OF FUNCTIONS, STRUCTURES, AND BEHAVIORS

by

Paul Josekutty Thomas

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Technology

West Lafayette, Indiana

May 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Alejandra J. Magana, Chair
Department of Computer and Information Technology

Dr. James L. Mohler
Department of Computer Graphics Technology

Dr. Marisa Exter
Department of Learning Design and Technology

Dr. Ida B. Ngambeki
Department of Computer and Information Technology

Dr. Paul Parsons
Department of Computer Graphics Technology

Approved by:
Dr. Kathryne A. Newton

To my friends, family, mentors, and most importantly to God:

Thank you for your love and support through everything.

ACKNOWLEDGMENTS

Throughout my journey of graduate school at Purdue University, I have been fortunate to meet people who have supported and encouraged me. Firstly, I would like to thank my advisor, Dr. Alejandra Magana for taking me under her wing and mentoring me. This dissertation would simply not exist if not for her guidance. I would also like to express my gratitude to Professor Victor Barlow for bringing me to Purdue University and offering me the opportunity to work as his teaching assistant. My experiences as a teaching assistant were instrumental in leading me to pursue this PhD. Dr. James Mohler, Dr. Ida Ngambeki, Dr. Marisa Exter, and Dr. Paul Parsons, I thank you for your patience and constructive criticism that helped me develop the competencies required to complete this PhD and develop the transferable skills that will serve me well following graduate school as well. I want to thank my committee for all their feedback and encourage throughout this research journey.

Thank you to my colleagues from the RocketEd research group. I enjoyed working with all of you and learning from all of you. I would like to thank Ying Ying Seah and Devang Patel for all their help. I especially want to thank Horane Holgate for being an ally, friend, and confidant from my very first year of graduate school. Special thanks to Carrie Sloma, Dee Bernhardt, Lancia Raja, Brandon Coventry, Thomas Sherringham, and all my friends at Purdue University and my church family at St. Thomas Aquinas who made graduate student life so much more enjoyable. Thank you for walking with me and supporting me through everything.

Thank you to my family for the love and for instilling in me the values I still hold dear. Finally, I thank God for the gift of faith and every blessing in my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Software modeling is an integral practice for software engineers especially as the complexity of software solutions increase. There is precedent in industry to model information systems in terms of functions, structures, and behaviors. While constructing these models, abstraction and systems thinking are employed to determine elements essential to the solution and how they are connected. However, both abstraction and systems thinking are difficult to put in practice and difficult to teach due to the, often, ill-structured nature of real-world IT problems. Unified Modeling Language (UML) is the industry standard for software modeling but unfortunately it is often used incorrectly and misunderstood by novices. This has also been observed in educational contexts where students encounter difficulty in employing the appropriate level of abstraction in modeling and programming contexts and not necessarily being able to view or treat software systems as being interconnected.

The researcher detailed a multi-methods approach, through the lens of pragmatism, towards understanding patterns of student proficiency with abstraction and software modeling in terms capturing the functional, structural, and behavioral aspects of an information system, as given by the Structures-Behaviors-Function framework. The quantitative strand involved the development of rubrics to analyze functional, structural, and behavioral models given by UML activity diagrams, class diagrams, and sequence diagrams, respectively. The subjects of this study were students enrolled in a sophomore-level systems analysis and design class. Descriptive analysis revealed patterns of modeling proficiency. Students were generally proficient in modeling the system in terms of functions but there was an overall drop-off in proficiency when modeling the system in terms of structures and behaviors. The results of the clustering analysis revealed underlying profiles of students based on abstract thinking and systems thinking ability. Two distinct clusters – high performing students and moderate performing students – were revealed with statistically significant differences between the groups in terms of abstract thinking and systems thinking ability. Further correlational analysis was performed on each cluster. The results of the correlational analyses pointed to significant positive associations between software modeling proficiency and the constructs of abstract thinking and systems thinking. Logistic regression analysis was then performed, and it could be inferred from the regression model that abstract thinking in terms of behaviors and systems thinking in terms of aligning sequence diagrams with activity diagrams were the most important predictors of high performance.

The qualitative strand of this study involved a case study approach using the think-aloud protocol centered around exploring how students utilized abstract thinking and systems thinking while constructing software models. The participants of this study were students who had completed the sophomore-level systems analysis and design course. Thematic analysis was utilized to identify themes of abstract thinking and systems thinking within the epistemic games of structural, functional, and process analyses. Two different approaches towards modeling information systems were identified and chronological visualizations for each approach were presented. Overall, it could be inferred from the results and findings of the study that the learning design of the sophomore-level course was successful in equipping students with the skills to proficiently model information systems in terms of functions. However, the students were not as proficient in modeling information systems in terms of structures and behaviors. The theoretical contribution of this study was centered around the application of the SBF framework and epistemic forms and games in the context of information systems. The methodological contributions pertain to the rubrics that were developed which can be used to evaluate software modeling proficiency as well as abstract thinking and systems thinking. Abstract thinking and systems thinking were successfully characterized in the context of information systems modeling. The results of this study have implications in computing education. The suggested instructional approaches and scaffolds can be utilized to improve outcomes in terms of structural and behavioral modeling proficiency.

CHAPTER 1. INTRODUCTION

This chapter provided an overview to this research study and to this manuscript. This chapter established significance within the existing literature about software modeling and software systems development. The significance of the research, the research questions, assumptions, limitations, and delimitations were discussed.

1.1 Background

Software modeling is prevalent in Information Technology (IT) industry and is widely used by software engineers and business analysts (Cernosek & Naiburg, 2004; Ho-Quang et al., 2017). Modeling enables IT professionals to better assess the requirements of complex software systems while simultaneously facilitating communication of these requirements in addition to technical and financial details to different stakeholders. Modeling can thus be utilized for to visualize software systems at different levels of detail (Cernosek & Naiburg, 2004). Moreover, software modeling is an important part of the software engineering discipline (ABET, 2016; Cernosek & Naiburg, 2004; Magana, Seah, & Thomas, 2017; Tamai, 2005). Therefore, it is necessary for those involved in the disciplines of software engineering or IT to have a working knowledge of how to create software models conforming to a standard and how to interpret these models (Boustedt, 2012). The process of constructing accurate software models requires the identification of details that are important to the system while ignoring those elements that are unnecessary or irrelevant (Kramer, 2007), this process is called abstraction or abstract thinking. It must also be recognized that software modules or components of a system do not exist in isolation and that the different software models constructed must also possess some degree of alignment between them since they are all based on the same requirements (Burgueño, Vallecillo, & Gogolla, 2018). The approach is called systems thinking as it forces a review of the relationship of various subsystems in a project (Brewer & Dittman, 2018). Therefore, designing information systems involves identifying essential details of the solution while omitting irrelevant details (Zehetmeier et al., 2019) and recognizing the connections between different aspects of the system. To represent these different interconnected aspects of an information system, modeling languages are used.

Unified Modeling Language (UML) was developed to serve as a standard of for representing software models (Engels and Groenewegen, 2000). UML can be and is often used to represent software systems in terms of functions, structures, and behaviors which provides a visual representation of its overall functionality and the inner workings that facilitate said functionality (Dennis, Wixom, & Tegarden, 2020; Robal, Viies, & Kruus, 2002). Unfortunately, UML is not used correctly in industry (Burgueño, Vallecillo, & Gogolla, 2018; Dobing & Parsons, 2006; Peneva, Ivanov & Tuparov, 2006) which has significant repercussions on the entire process of software development because software is often coded or programmed based on these models. Errors made during the construction of these models are the most expensive to correct at a later point in the development or maintenance process (Fernández-Sáez, Chaudron, & Genero, 2018; Queralt & Teniente, 2012).

In light of the challenges faced in industry, the Accreditation Board for Engineering and Technology (ABET) requires graduates of accredited programs to possess or exhibit “an ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs” (p.3). Previous studies have been conducted regarding the use of UML (Hadar & Hadar, 2006) and abstraction levels employed by students in programming classes (Bucci, Long, & Weide, 2001), however the proficiency of abstraction displayed by students during modeling, or the quality of the models produced were never formally evaluated.

The educational problem examined in this study was informed by what is observed in industry and across literature. The researcher took quantitative and qualitative approaches to evaluate student proficiency in modeling software systems in terms of functions, structures, and behaviors using UML diagrams. The quantitative approach involved the development rubrics to evaluate student proficiency at software modeling, especially abstraction and systems thinking ability. The qualitative approach examined the software modeling process exhibited by participants in terms of how they employ abstract thinking and systems thinking while constructing these models. The study was guided by the following research questions: i) To what extent to did students demonstrate proficiency in abstract thinking while analyzing software systems in terms of functions, structures, behaviors?; ii) To what extent to did students demonstrate proficiency in systems thinking while analyzing software systems in functional, structural, and behavioral representations?; iii) What were the characteristics or profiles of students in terms of abstraction and systems thinking ability?; iv) How did students use abstraction and systems thinking modeling software systems in terms of functions, structures, and behaviors?

1.2 Significance

The literature in the areas of systems developments, software modeling, and UML, did not delve into how abstract thinking and systems thinking play a role in the development of software models, especially in an educational context. I believed that by taking a multi-methods approach to this research, I could gain a formal understanding as to the areas of software modeling where students exhibit proficiency or, on the contrary, lack proficiency, while also gaining an insight into how students employ abstract thinking and systems thinking while constructing software models. The insights gained from this study can impact the learning interventions and assessment mechanisms that are employed in collegiate systems development courses.

1.3 Statement of Purpose

The purpose of this research was to identify and describe student proficiency in the various aspects of software modeling and to gain insights into how abstraction and systems thinking ability influenced proficiency. Furthermore, this research aimed to identify and describe how students employed abstract thinking and systems thinking while constructing software models. Understanding the patterns of proficiency and how students employed abstract thinking and systems thinking would provide insights into their strengths or weaknesses in software modeling. This could have further instructional and pedagogical implications in systems analysis courses.

1.4 Research Questions

The research study was centered around the following questions:

1. To what extent to did students demonstrate proficiency in abstract thinking while analyzing software systems in terms of functions, structures, and behaviors?
2. To what extent to did students demonstrate proficiency in systems thinking while analyzing software systems in functional, structural, and behavioral representations?
3. What were the characteristics or profiles of students in terms of abstraction and systems thinking ability as evidenced by their system representations?

4. How did students use abstract thinking and systems thinking when modeling software systems in terms of functions, structures, and behaviors?

1.5 Assumptions

The following assumptions informed this research study:

1. There was a need to take quantitative and qualitative approaches to gain insights into the software modeling proficiency and behaviors of students.
2. Participants responded to the best of their ability in the qualitative study.

1.6 Dissertation Structure

This dissertation was divided into eight chapters:

1. Chapter 1 provided the background, significance, and introduced the research questions.
2. Chapter 2 covered the review of literature pertaining to abstract thinking, systems analysis and design, systems thinking, and UML along with their importance.
3. Chapter 3 provided a detailed over of the theoretical framework used as part of this research study.
4. Chapter 4 discussed the methodology employed in this study. This chapter contained a detailed discussion of the mixed methods approach taken for this study.
5. Chapters 5 detailed the results of the quantitative strand of this research study.
6. Chapter 6 detailed the results of the qualitative strand of this research study.
7. Chapter 7 discussed the results of both strands in the context of literature and details the implications.
8. Chapter 8 contained the conclusions of the study, the limitations, and recommendations for future research.

1.7 Summary

This chapter introduced the research study, including background, significance, and research questions. This chapter also provided an outline of the organization of the document. The next chapter covered the review of literature and explored the areas of abstract thinking, systems analysis and design, systems thinking, and UML.

CHAPTER 2. REVIEW OF LITERATURE

This chapter provided an overview of the literature pertaining to software modeling, abstract thinking, systems thinking, and unified modeling language (UML). The chapter discussed the importance of each term and existing work done in the area.

2.1 Software Modeling

System models can be used to describe the requirements of the software system. Models can also be used to remove or abstract irrelevant details that are not integral to the implementation of a potential solution (Engels & Groenewegen, 2000). A software architecture consists of interrelated components that are organized to focus on different views of the system, in effect serving as blueprints during the software development process (Eriksson, Börstler, & Borg, 2005; Mattsson, Fitzgerald, Lundell, & Lings, 2012; Robal, Viies, & Kruus, 2002). Software architecture has non-trivial effects on the understanding and maintenance of a software system (Bass et al., 2003).

In the context of software development, abstract systems modeling has its roots in the 1970s. They have been employed to ensure internal consistency of data and behaviors, as well as to bring about better alignment between the software solution and the real-world domain (Engels & Groenewegen, 2000). While models have been used primarily to describe software architecture, they can also describe business processes and other frameworks. (Larsen, 1999). The quality of an information system is often determined early in the software development process specifically during conceptual modeling and requirements specification (Queralt & Teniente, 2012). Furthermore, errors made during requirements specification and conceptual modeling permeate through the system as it is developed, thus making them more difficult and expensive to fix compared to those errors made during implementation. Software development often employs an approach referred to as model-driven engineering where models are often "combined, refined, translated, and integrated" (Balaban & Maraee, 2013, p.24).

An example of software modeling is use-case modeling. Use-case modeling is an approach towards requirements engineering that involves describing the functional requirements of a system through either narrative or graphical means (Whitten & Bentley, 2007). Use case narratives (Appendix B - Use Case Narrative Template) are used to detail the typical course of events that the system takes and then the alternate course of events that the system follows based on some predefined criteria. The alternate courses of events detail the system response for specific inputs that are not handled by the typical course of events. For example, in the use-case of handling user logins, a typical event is to validate the username and password of a user. An alternate course for this scenario will detail the steps for the system to follow if the user enters an invalid username or password. A system developed based on detailed use case narratives will be capable of handling diverse inputs with a degree of coherency and reliability.

In education, software modeling is viewed as an essential skill and as being necessary to provide a balanced education in the discipline of software engineering (Tamai, 2005). Software modeling allows students to be reflexive in their learning as they design and develop software solutions to solve a problem (Boustedt, 2012). It has been shown that the software modeling through modeling tools directly influence how students develop and utilize models (Burgueño, Vallecillo, & Gogolla, 2018). It must be noted that while students, in general, are able to construct and understand different views of an information system, they often do not understand the relationships between them (Burgueño, Vallecillo, & Gogolla, 2018). Teaching modeling is essentially the same as teaching abstraction because modeling requires learners to ignore irrelevant details while capturing properties and structure that are crucial to the design of the solution (Tamai, 2005). The literature ties software modeling to the construct of abstract thinking or abstraction.

2.2 Abstract Thinking

Abstract thinking or abstraction is often used interchangeably and has many different operational definitions across literature depending on the context in which it is viewed (Zehetmeier et al., 2019). Abstract thinking in general involves determining what details need to be focused on and what details can be ignored, and it can be applied at multiple levels (Hadar & Hadar, 2006; Kramer, 2007; Zehetmeier et al., 2019). Abstraction plays a key role in solving computing-related problems (Devlin, 2003; Kramer, 2007) and abstraction ability has been identified as integral to the software engineering process (Ghezzi, Jazayeri, & Mandrioli, 2002). Abstraction can involve drawing generalizations after removing unnecessary details (Kramer, 2007; Hill, Houle, Merritt, & Stix, 2008). Proficiency in abstract modeling is essential to all engineering disciplines as engineers are often required to “design, build, and reason about formal abstractions” (Devlin, 2003, p. 38). Abstraction skills also involve the ability to modify existing abstractions and create new abstractions (Bennedsen & Caspersen, 2008). Abstract thinking is especially important in the realm of systems analysis and design because it allows for constructing models that map to real-world constructs (Devlin, 2003).

2.2.1 Abstract Thinking in Systems Analysis and Design

Systems Analysis and Design refers to the methodologies used to describe existing real-world systems or those systems that have not been implemented yet (Wand and Weber, 1993). The analysis phase focuses on identifying the high-level functionality of a system and potential users of the system. The design phase determines how the system will operate and provides detailed specifications for the system architecture, user interfaces, database etc. (Dennis, Wixom, & Tegarden, 2020). This is often accomplished through models that are not only used for describing the requirements of the software system, but to also strip away or abstract irrelevant details that are not necessary for the potential implementation of the solution while highlighting those important to it (Engels & Groenewegen, 2000; Rijke, Bollen, Eysink, & Tolboom, 2018). In this manner, abstraction is responsible for converting something the real-world and concrete domain into a model that can then be mapped to a program in a specific programming language (Kramer, 2007). Systems analysis and design requires that various aspects of an information system be modeled such as the structural, behavioral, and functional aspects (Siau & Rossi, 2011; Dennis, Wixom, & Tegarden, 2020). Abstraction skills are essential in the construction of not just these models, but the designs, and implementations that are fit for the specific purpose at hand. Abstract thinking is also necessary for reasoning about abstractions in formal models or programs (Kramer, 2007) and it can be used for decomposing problems (Nguyen & Wong, 2001). Any kind of visualization that represents programming code is an abstraction (Engels & Groenewegen, 2000) and abstract models can be used to verify code (Clarke, Grumberg, & Long, 1994).

Figure 2.1 illustrates how abstraction is used to translate details from the problem space to the solution space while removing details that irrelevant to the solution.

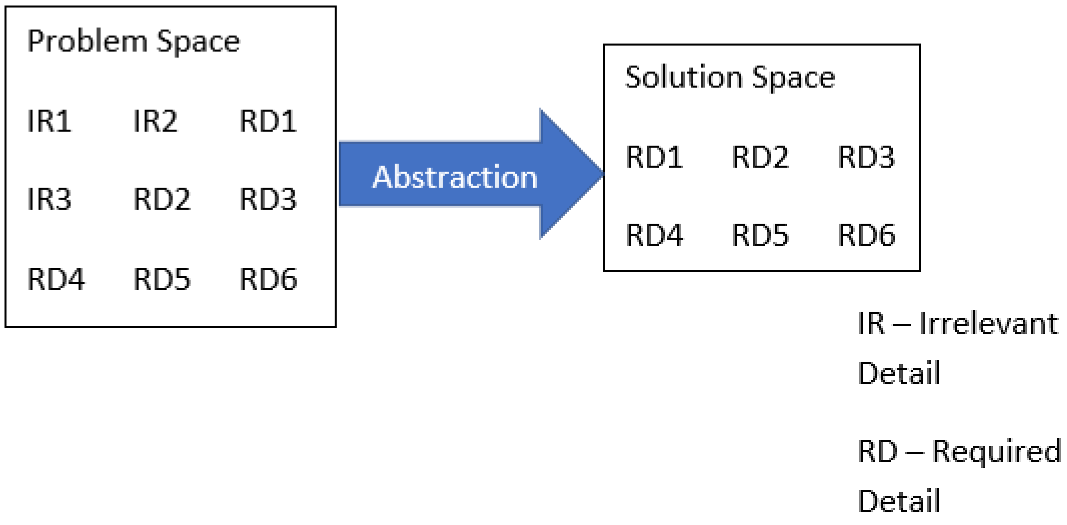


Figure 2.1. Illustration of abstraction

2.2.2 Characterizing and Measuring Abstraction

While literature generally comes to the consensus that there is value in not just teaching abstract thinking (Böttcher, Schlierkamp, Thurner, & Zehetmeier, 2016; Hill, Houle, Merritt, & Stix, 2008; Koppleman & van Dijk, 2010) but also in testing and measuring the abstraction ability possessed by learners (Hill et al., 2008; Kramer, 2007). The reality is that there are limited ways to test or measure abstract thinking reliably especially in the realm of information technology or modeling information systems (Kramer, 2007). This is exacerbated by the "general lack of explicit characterization and addressing of abstraction not to mention development of abstraction as a competence, the teaching and learning of abstraction ability" (Bennedsen & Caspersen, 2008, pg. 23). However, exploration of the literature revealed a few studies that characterized and operationalized abstract thinking.

In the realm of mathematics education, abstraction level is characterized as being dependent upon the complexity of the concept and the notion of process-object duality. Process-object duality refers to how relationships between objects or operations performed on objects are considered to be objects at a higher level of abstraction (Tall & Thomas, 2002). The field of computer science characterizes abstraction in two different ways: i) data abstraction – separating the implementation details from the logical properties of the data; and ii) procedural abstraction – separating the implementation details from the logical properties of a procedure (Walker, 1996).

Data abstraction involves the separation of behavior from implementation which allows easier modification and maintenance of code (Liskov, 1988). Once data abstraction is implemented, a set of objects can only be modified by or manipulated by fixed operations. Procedural abstraction refers to how programs can call or invoke a specific procedure to complete a task without having to care about the implementation of the function or procedure (Liskov, 1988). Procedural abstraction and data abstraction allow for the implementation of APIs which aid a programmer in developing software without necessarily being aware of specific implementation details of the data or procedure (Wing, 2008). The use of procedural or data abstraction allows for the programmer to remove oneself from the implementation details of objects or functions and focus on problem-solving. Perrenet (2010) offered a four-level view of describing algorithm abstraction, illustrated in Figure 2.2. The problem level is the highest level of abstraction where the algorithm is viewed as a black box, which takes an input and provides an output. This level of abstraction is best used for algorithm selection. At the object level, an algorithm is not associated with a specific programming language. Data abstraction is performed at this level and this allows for data structures and other representations of data to be changed without the affecting programs that use the data (Liskov, 1988). At the program level, an algorithm is viewed as a process and is associated with a specific programming language. Procedural abstraction is performed at this level where in a procedure or function call is used to accomplish a task. Effectively, the lines of code that performs the procedure call will be replaced by the procedure body (Morgan, 1988). The execution level is the lowest level of abstraction and provides the most specific implementation details. The study indicated that abstract thinking skills can be developed, and feedback is essential to the process. Higher levels of abstraction ability are usually associated with those who have attained an expertise in the area. Software models also serve the purpose of raising the level of abstraction involved in software development taking it from the lower levels such as execution to the program, object, or problem level (Mattsson, Fitzgerald, Lundell, & Lings, 2012). Object-oriented programming also necessitates a higher level of abstraction compared to procedural programming because of the data abstraction involved (Liskov, 1988; Sprague, & Schahczenski, 2002).

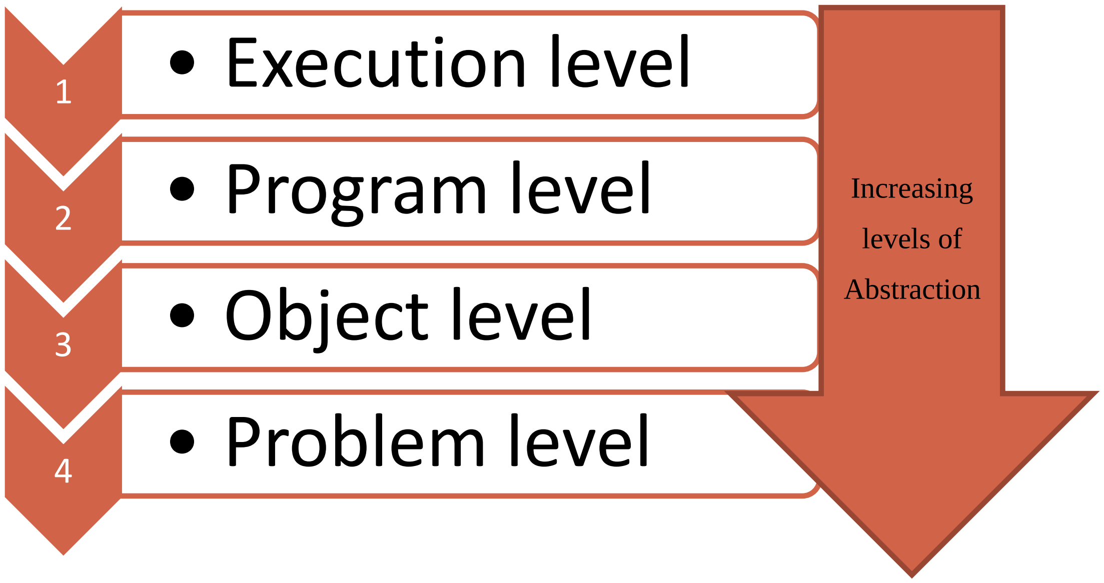


Figure 2.2. Algorithm abstraction levels

Studies conducted in the areas of computer science and object-oriented programming characterized abstraction ability as a general skill that is associated with the cognitive development of a person (Or-Bach & Lavy, 2004; Bennedsen & Caspersen, 2008). A theory of cognitive development (Adey & Shayer, 2006) defines eight stages of cognitive development ranging from Pre-Operational at the earliest stages to Formal generalization at the most advanced stages. There is a positive correlation between abstraction ability and age (Rijke, Bollen, Eysink, & Tolboom, 2018) indicating that one's abstraction ability is linked to their cognitive development which in turn is linked to one's age. Most undergraduate learners are at the Early Formal or Mature Formal levels of cognitive development with the implication being that they are capable of handling multiple variables and the relationships between them. It is also worth noting that studies have connected bilingualism to improved abstraction ability in the context of symbol mathematics (Mielicki, Kacinik, & Wiley, 2017). The study by Bennedsen and Caspersen (2008) sought to identify the relationship between abstraction ability - as operationalized by Adey and Shayer's theory of cognitive development, and performance in computer science. The computer science course in question focused on introducing learners to object-oriented programming. The data from 263 Computer Science students did not support the hypothesis that abstraction ability predicted performance in terms of academic achievement. The results were, in part, explained by the course placing an emphasis on coding as opposed to design. Coding or programming requires individuals to traverse from higher levels of abstraction to lower levels to implement the solution in a chosen programming language, often having to think at the execution, program, or object level (Perrenet, 2010). The introductory programming course in question may have also placed limited cognitive demands on the students since the goals of the course revolves around exposing students to programming concepts and conceptual modeling in object-oriented programming.

2.2.3 Summary

While the literature is vast about the importance of abstract thinking or abstraction, there is little work done regarding the formal testing of abstract thinking or abstraction skills specifically in the context of software modeling (Kramer, 2007). There are no pre-existing rubrics that can be used to measure abstract thinking in the context of software modeling. This has theoretical and methodological implications on the proposed study which focuses on characterizing student proficiency in software modeling in terms of functions, structures, and behaviors. The theoretical implications are centered around the two complimentary aspects of abstraction which involves stripping away irrelevant details as well as drawing generalizations (Hill, Houle, Merritt, & Stix, 2008; Kramer, 2007; Wing, 2008). There is no standard approach for characterizing or measuring either component of abstraction. The details that are to be stripped away or deemed irrelevant will differ from one view to another and this necessitates the development of rubrics to assess abstraction. The methodological implications on the study center around determining how to measure abstraction proficiency in the context of software modeling.

2.3 Systems Thinking

While abstract thinking is generally characterized as the process of removing unnecessary details, it is not focused on recognizing connections between different components. Systems thinking is broadly defined as the ability to see the interrelationships of components in a complex system (Senge, 1990; Stearman, 2000). Systems thinking or the systems approach forces a review of the relationship of various subsystems in a project (Brewer & Dittman, 2018). Systems thinking has also been defined as an approach towards integrating "…people, purpose, process and performance because it is a framework for seeing and working with the whole(s), rather than only the individual part, and for seeing the inter-relationships between parts…" (Godfrey, Deakin Crick, & Huang, 2014, p.113). This also allows practitioners of systems thinking to perceive possible consequences of changes made to one part of a system in terms of how it can affect other parts of the system or the broader environment (Wolstenholme, 2003).

Systems thinking has its origins in systems theory (Von Bertanlanffy, 1972) which stated that individual parts and processes cannot completely explain a specific phenomenon within a complex system (Von Bertanlanffy, 1972). System theory further stated that systems generally have a defined boundary and can be composed of components that interact with one-another achieve certain functions. These statements were made with respect to biological systems where it takes a coordination of individual parts and processes to enable certain phenomena. Since the 1960s, systems approaches derived from systems theory have been employed in numerous disciplines. Systems thinking has been identified as an essential skill necessary to solve complex and interdependent problems (Grohs et al., 2018), especially in disciplines like engineering (Lammi, 2011) and biology (Boersma, Waarlo, & Klaassen, 2011). The Scientific Thinking and Integrative Reasoning Skills (STIRS) framework, developed by the Association of American Colleges and Universities (AAC&U), advocates for systems analysis and systems approaches to be included as part of curriculum to equip learners with the skills to analyze complex systems and the interactions between them (Riegelman, 2016). Systems thinking approaches also help students, in the context of engineering, to discover the functions or purposes of a system and explain how these functions are achieved through different behaviors (Lammi, 2011). However, it must be noted that the skillset required to develop a systems architecture "…can not be achieved through rote learning or the cognitive application of pre-defined knowledge…" (Godfrey, Deakin Crick, & Huang, 2014, p.112). Learners must think critically to uncover knowledge that can be applied to a specific system. Therefore, the purpose of teaching systems thinking "…is to achieve competence rather than to acquire specialized subject knowledge…" (Godfrey, Deakin Crick, & Huang, 2014, p.112).

Systems thinking is also considered important in the space of design where designers are expected to have "…a special holistic overview spanning from technical, via socio-cultural aspects to economic aspects" (Sevaldson, 2011, p.3). Systems thinking enables designers to tackle complexity and deal with "wicked problems". Practicing systems thinking also allows designers to respond to changes while also increasing their understanding of frameworks specific to each user or client and the technology involved. Unfortunately, despite its importance systems thinking is not widely spread in the space of design and there is no uniform practice of systems thinking that has been developed (Sevaldson, 2011). System thinking also plays an important role in making management decisions as it forces decisions to be made while considering often conflicting interests and values (Ulrich, 1994) with the assertion being that design improvement is difficult if not impossible without holistic understanding of the system.

While there are a few ways to define systems thinking and its importance is clear, research into systems thinking, especially in the context of information systems, is sparse and this can be at least partially attributed to how the construct does not fit well with the typical positivist research done in the field (Alter, 2004). Checkland (1988) argues that the field of information systems has largely ignored the concept of systems thinking even through its various iterations as systematic thinking and systemic thinking. This can be explained by a preference for tool-focused thinking and the difficulty of defining systems thinking in practical terms (Alter, 2004). Systems thinking is also difficult to apply in real-world projects due to the often ill-structured nature of problems faced during their execution (Grohs et al., 2018; Yeo, 1993).

The use of systems thinking has also been associated with more positive outcomes in IT implementations, especially in healthcare (Rothschild et al., 2005). The definitions of systems thinking also suggest that it is essential for developing software systems by taking into consideration how different modules interact with another. Therefore, it is important that models are consistent and align with one-another, which is effectively systems thinking in practice. These relationships between these modules are also often captured in software models using UML (Boustedt, 2012; Dobing & Parsons, 2006; Eriksson, Börstler, & Borg, 2005; Hadar & Hadar, 2006; Robal, Viies, & Kruus, 2002). However, it must be noted that students do encounter a degree of difficulty in making connections between different UML diagrams (Burgueño, Vallecillo, & Gogolla, 2018). Unfortunately, literature pertaining to the measurement of systems thinking in the context of information systems education is sparse. This has theoretical and methodological implications on the proposed study which focuses on characterizing student proficiency in software modeling in terms of functions, structures, and behaviors. The theoretical implications pertain to how systems thinking should be operationalized in the context of software modeling and the methodological implications are centered around how systems thinking can be measured.

2.4 Characteristics of Object-Oriented Systems Analysis and Design

When developing software systems, two commonly used paradigms are that of structured development – centered around function and data – and object-oriented development (de Champeaux et al., 1990). The structured approach treats data as containers of information that are accessed and manipulated by functions. However, the object-oriented approach involves mapping real-world entities to objects where each object contains information and exhibit behaviors. Object-oriented analysis involves defining objects along with their attributes and behaviors based on applying abstraction to real-world entities. This requires both static and dynamic requirements to be captured (Hausmann, Heckel, & Taentzer, 2002).

The unified process is an object-oriented analysis methodology that possesses two major characteristics when designing systems and formally documenting requirements: (i) Use-case driven, and (ii) Architecture-centric (Dennis, Wixom, & Teagarden, 2020). The use-case driven analysis approach (UCDA), as the name implies, is centered around use cases where each use case is defined as "…a system usage scenario characteristic of a specific actor" (Regnell, Kimbler, & Wesslen, 1995, p.1). An actor refers to a group of users that interact with the system in a similar manner. The focus of UCDA is on the analysis and identification of use cases which helps reduce the complexity of requirement analysis. However, one main drawback of UCDA is that use cases are often simple in nature and only focus on one activity at a time. The generation of a loose collection of use cases that does not necessarily capture the holistic requirements of the system (Regnell, Kimbler, & Wesslen, 1995). There is also ambiguity in terms of how user cases should be described or what kind of events – internal vs external – should be focused on. By also focusing on the architecture-centric approach, the analysis and design process also takes into consideration overall system functionality as well as non-functional requirements, referred to as quality attributes, such as performance, scalability, and maintainability (Sangwan et al., 2008). The architecture centric approach first identifies the most important systemic properties and how they are linked to business goals by proposing three primary views a given system: the functional view, the structural view, and the behavioral view. The software architecture drives the specification, development, and documentation of the system. An integrated approach would combine the analysis and modeling activities associated with the use-case driven approach alongside the architecture-centric approach of determining quality attributes of the system (Sangwan et al., 2008). The Rational Unified Process (RUP) is an iterative object-oriented development framework that encompasses architecture creation as well as elaborate design, implementation, and testing (Kazman et al., 2004). Though RUP is often described as being use-case driven, it advocates for describing software systems using multiple interconnected views (Eriksson, Börstler, & Borg, 2005; Kazman et al., 2004; Robal, Viies, & Kruus, 2002). RUP also involves continuous refinement and testing in an iterative and incremental manner. Specifically, RUP involves the creation of a logical view that is focused on describing end-user functionality and a process view that describes how the functionality is implemented. This in turn necessitates the use of models to describe: (i) the functionality of the information system as viewed by the end user – functional view; (ii) the objects along with their corresponding attributes, behaviors, and relationships that implement the functionality – structural view; and iii) how the different objects interact with one-another, through messages and responses, to achieve the required functionality – behavioral view. Unified Modeling Language (UML) provides a common vocabulary and a set of diagrams to describe and analyze systems.

2.5 Unified Modeling Language

Unified Modeling Language (UML) was initiated and promoted by the industry to standardize the process of object-oriented conceptual modeling during software development (Aljumaily, Caudra, & Laefer, 2019; Engels and Groenewegen, 2000). UML, as its name implies, is a "general purpose modeling language" (Peneva, Ivanov, & Tuparov, 2006) that is not limited to modeling information systems but can be extended to visualize business processes as well. UML has its origins in the unification of Object Modeling Technique (OMT) pioneered by Grady Booch and Jim Rumbaugh with the Object-Oriented Software Engineering approach pioneered by Ivar Jacobson (Booch, 1999). UML as it is known today was direct successor to these methods and it underwent a standardization process under the supervision of the Object Management Group (Fowler, 2004).

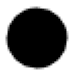
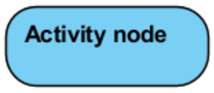
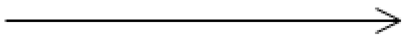
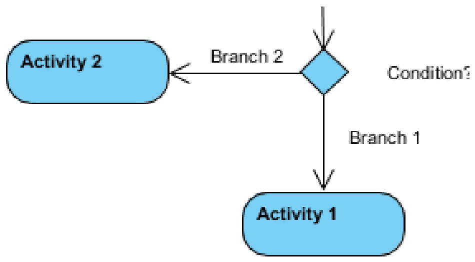
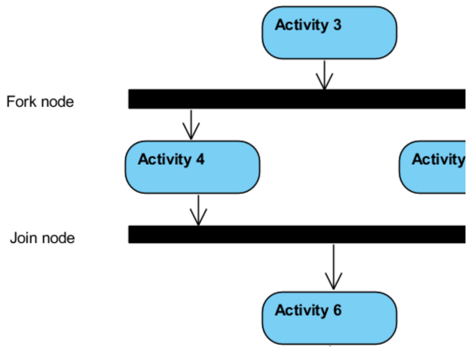
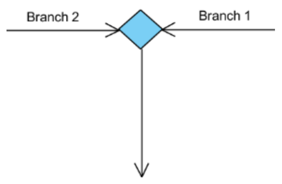
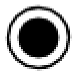
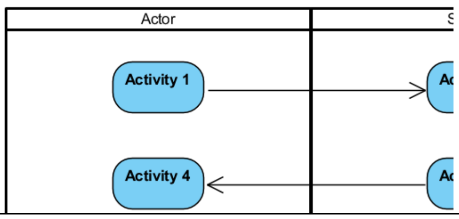
UML is used in software engineering education (Unkelos-Shpigel, Sheidin, & Kupfer, 2019) and has different diagrams that are used to be model different aspects of the information system, thereby providing different views of the information system (Balaban & Maraee, 2013; Boustedt, 2012; Dobing & Parsons, 2006). Therefore, UML can also be described as a visual language consisting of graphical symbols (Moody & van Hillegersberg, 2008). These graphical symbols are governed by a set of rules that are different for each diagram and the rules determine how these symbols can be combined. Through these different diagrams, UML also facilitates effective planning and visual communication (Ho-Quang et al., 2017). Using UML also allows to developers and analysts to create barriers of abstraction so that they can operate at the most appropriate level of abstraction for the specific stage of development that they are in. This in turn facilitates easier implementation of abstraction during software development (Hendrix et al., 2000).

UML in total provides 14 diagrams that can be used model information systems. However, some diagrams are used more than others (Ciccozzi, Malavolta, & Selic, 2019). Among the 14 diagrams offered by UML, activity diagrams, class diagrams and sequence diagrams are the most used widely in industry. Unfortunately, they are either poorly understood or are often incorrectly used (Dobing & Parsons, 2006; Peneva, Ivanov, & Tuparov, 2006). The use of these diagrams is driven by existing precedent for capturing functional, structural, and behavioral details of an information system using models (Dennis, Wixom, & Tegarden, 2020). The UML diagrams under consideration for this study is discussed below.

2.5.1 Activity Diagram

An activity diagram is a directed graph consisting of nodes and edges, and it can be used to represent the flow of control in an information system for a specific high-level function performed by the system. They are inspired by flowcharts can be used to specify the behavior of use cases in information systems (Eshuis, 2006). Table 2.1 details the different symbols typically used in an activity diagram (Dennis, Wixom, & Tegarden, 2020).

Table 2.1: Activity diagram symbols

Symbol	Symbol Name	Description
	Start node	A solid circle represents the start point for an activity diagram
	Activity	An activity or action refers to any step performed by the user or system and is represented by a rectangle with rounded corners
	Action flow	Action flows connect one activity to another and is represented by an arrow
 Activity 2 Branch 2 Condition? Branch 1 Activity 1	Decision	A decision is represented by a diamond and has one incoming action flow. The diamond is also labeled. A decision results in branching action flows that are labeled. with alternate paths.
 Activity 3 Fork node Activity 4 Activity Join node Activity 6	Synchronization bars	Synchronization bars consist of a fork node and a join node. A fork node splits one action flow into multiple concurrent flows. A join node follows a fork node to join multiple concurrent flows into a single action flow.
	Merge	A merge node brings together multiple control flows that are not concurrent.
	Final node	A solid circle nested inside another circle represents the end point of an activity diagram.
	Swimlanes	Swimlanes are used to group related activities.

The different symbols can be used together to depict the overall flow of a specific functionality offered by an information system. Figure 2.3 illustrates an activity diagram that models the functionality of a ticket reservation system. Swimlanes allow for the modeling of specific actions performed by the user and system. Activities are typically labeled in the format “verb-object” with each activity consisting of only one verb. Decision nodes are used to implement error-handling or unexpected scenarios such as the user selecting a travel date in the past or flights being unavailable. Merge nodes appear identical to decision nodes in that they are also represented by diamonds, but it differs from a decision node in terms of number of incoming action flows. Decision nodes have a single incoming action flow with two outgoing branches whereas merge nodes have multiple incoming actions flows and a single outgoing action flow.

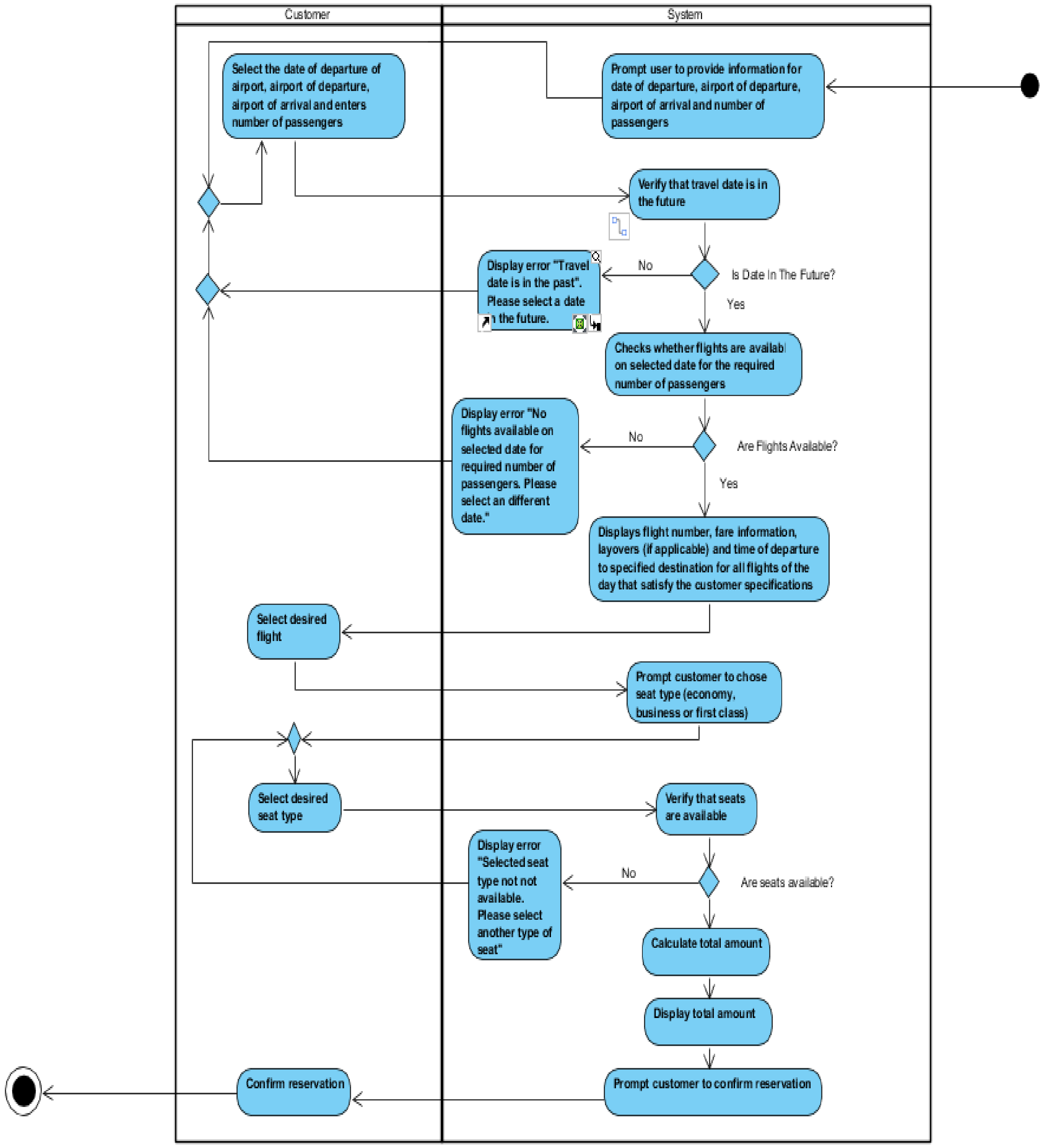

Customer
System
Select the date of departure of airport, airport of departure, airport of arrival and enters number of passengers
Prompt user to provide information for date of departure, airport of departure, airport of arrival and number of passengers
Verify that travel date is in the future
Display error "Travel date is in the past". Please select a date in the future.
No
Is Date In The Future?
Yes
Checks whether flights are availabl on selected date for the required number of passengers
Display error "No flights available on selected date for required number of passengers. Please select an different date."
No
Are Flights Available?
Yes
Displays flight number, fare information, layovers (if applicable) and time of departure to specified destination for all flights of the day that satisfy the customer specifications
Select desired flight
Prompt customer to chose seat type (economy, business or first class)
Select desired seat type
Verify that seats are available
Display error "Selected seat type not not available. Please select another type of seat"
No
Are seats available?
Calculate total amount
Display total amount
Confirm reservation
Prompt customer to confirm reservation

Figure 2.3. Activity diagram example

2.5.2 Class diagram

Class diagrams represent the objects involved in implementing a specific functionality offered by the information system. A class diagram consists of classes, associations, and a set of integrity constraints defining the cardinality of the associations between classes (Queralt & Teniente, 2012). It captures the various attributes and behaviors of each object and how they are related to one-another. Class diagrams are described as the most important of all UML diagrams (Moody & van Hillegersberg, 2008). Table 2.2 details the different symbols typically used in a class diagram (Dennis, Wixom, & Tegarden, 2020).

Table 2.2: Class diagram symbols

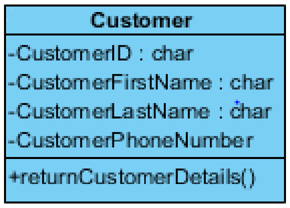
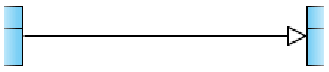
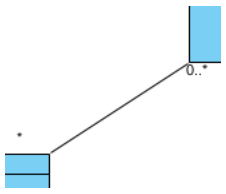
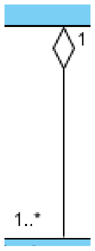
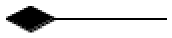
Symbol	Symbol Name	Description
 Customer -CustomerID : char -CustomerFirstName : char -CustomerLastName : char -CustomerPhoneNumber +returnCustomerDetails()	Class	Represents objects consisting of attributes and behaviors. Attributes are listed in the second partition below the name and methods are listed in the third partition. Attributes and behaviors have a visibility associated with them which determines how they can be accessed. The minus sign denotes that an attribute or behavior is private, and it can only be accessed within the class. The plus sign denotes that an attribute or behavior is public, and it can be accessed outside the class. A class can also show the specific data types of each attribute and the return type of each behavior.
	Inheritance	A relationship between two classes where a sub-class or child-class tends to derive attributes and behaviors from a super-class or parent-class.
	Association	Associations represent static relationships between classes and are represented by solid lines. Associations have a multiplicity denoting the number of instances of each class on either side of a relationship (one-to-one, one-to-many etc.)

Table 2.2 continued

	Aggregation	An aggregation denotes a relationship between two classes where one class is a part of the other class, but the two classes are not dependent on each other. A hollow diamond is used to represent this.
	Composition	A composition relationship is a special type of aggregation where the part class is destroyed when the whole class is destroyed. It is represented using a solid diamond.

The different symbols can be used together to depict the static structure of an information system. Figure 2.4 illustrates the static structure of an airline ticket reservation system. This class diagram depicts the objects along with their corresponding attributes, behaviors, and the relationships between the objects. This static structure enables the information system to offer the different functionalities that it has.

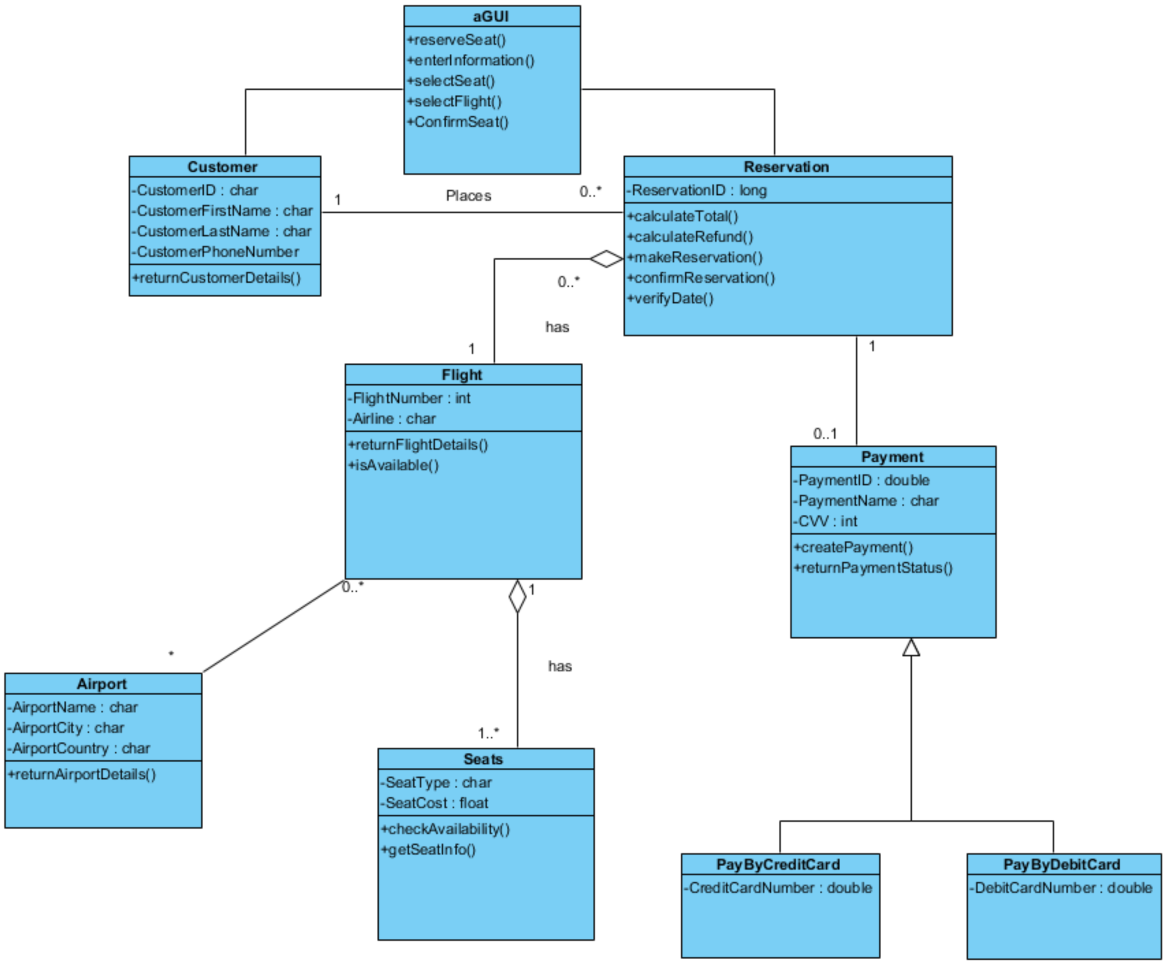
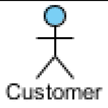
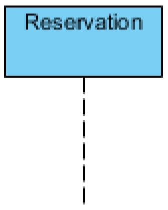
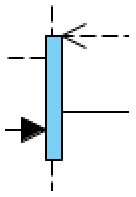
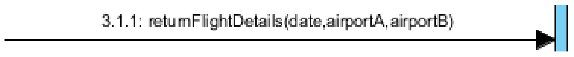
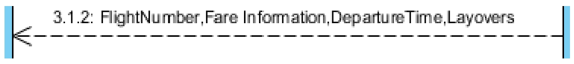


Figure 2.4. Class diagram example.

2.5.3 Sequence diagram

Sequence diagrams are used to capture the interactions between the different objects involved in implementing a certain functionality offered by the information system (Dennis, Wixom, & Tegarden, 2020). Sequence diagrams allow for capturing interactions between the user and system as well as the interactions between the various sub-systems that exist within the system. While activity diagrams capture the overall flow of control in an information system – including branching flows of control, sequence diagrams typically only detail the interactions between active objects for a specific function and outcome with an option to include the interactions for alternate scenarios. Sequence diagrams also contain a time-component that represented by the vertical axis. Table 2.3 details the symbols typically used in a sequence diagram.

Table 2.3: Sequence diagram symbols

Symbol	Symbol Name	Description
	Actor	An actor refers to any entity – a human user or another information system, that interactions with the information system being modeled.
	Lifeline	A lifeline depicts the lifetime of an object or actor. It is represented by rectangle connected to a dashed vertical line.
	Execution Occurrence	Execution occurrences are represented by thin rectangles drawn on lifelines. The top and bottom of this rectangle depict the initiation and completion of a specific operation.
	Message	A message defines a specific communication between two lifelines. It is represented using a solid arrow.
	Response	Like a message, a response also defines a specific communication between two lifelines. However, a response is produced to pass information back to the origin lifeline of a message.

The different symbols can be used together to model the various interactions that takes place between the structures of an information system to achieve a certain functionality. Figure 2.5 illustrates the interactions between objects that take place for a successful airline ticket reservation system.

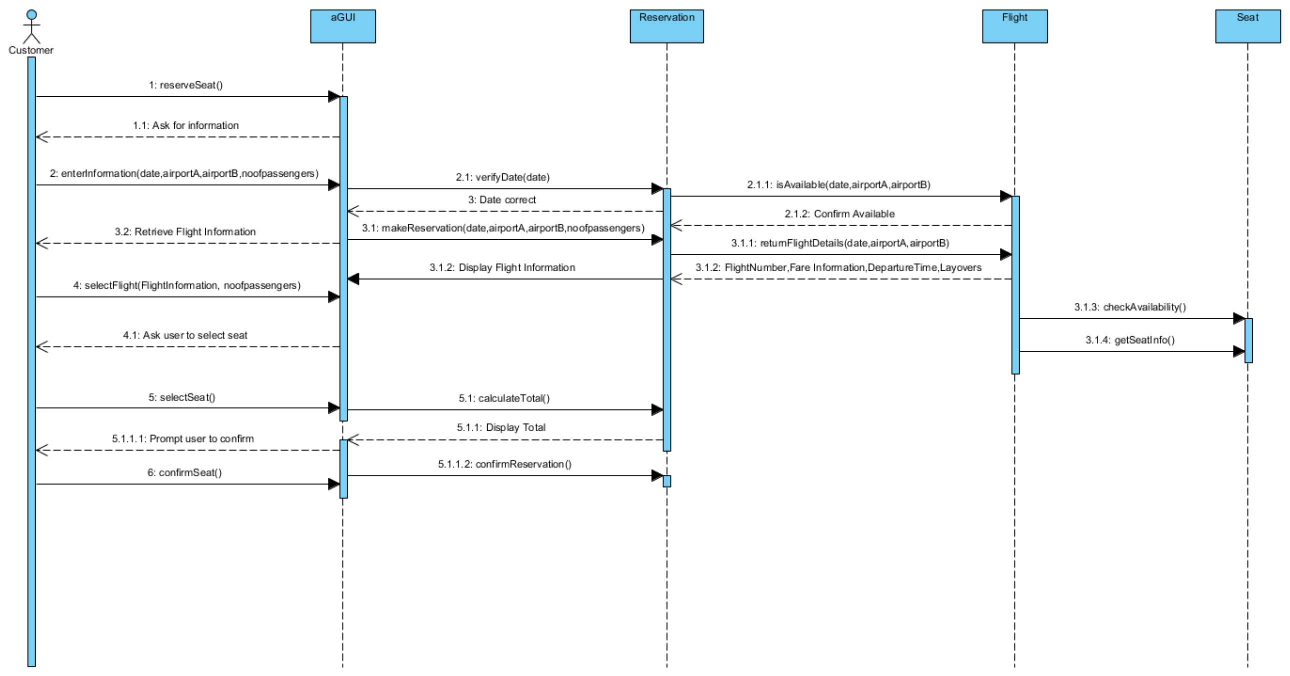

Customer
aGUI
Reservation
Flight
Seat
1: reserveSeat()
1.1: Ask for information
2: enterInformation(date,airportA,airportB,noofpassengers)
2.1: verifyDate(date)
2.1.1: isAvailable(date,airportA,airportB)
3: Date correct
2.1.2: Confirm Available
3.2: Retrieve Flight Information
3.1: makeReservation(date,airportA,airportB,noofpassengers)
3.1.1: returnFlightDetails(date,airportA,airportB)
3.1.2: Display Flight Information
3.1.2: FlightNumber,Fare Information,DepartureTime,Layovers
4: selectFlight(FlightInformation, noofpassengers)
3.1.3: checkAvailability()
3.1.4: getSeatInfo()
4.1: Ask user to select seat
5: selectSeat()
5.1: calculateTotal()
5.1.1: Display Total
5.1.1.1: Prompt user to confirm
5.1.1.2: confirmReservation()
6: confirmSeat()

Figure 2.5. Sequence diagram example.

2.6 Agile Approaches to Project Management and Scrum

UML found widespread use in the more traditional approaches to software development (Santos et al., 2016). In these approaches, requirements are modeled at the start of the development process before implementation begins. Examples of these models include the waterfall model or the V-model. These approaches require that all implementation details be incorporated in the planning stage right at the beginning. Unfortunately, these approaches have a drawback in that any misunderstandings or mistakes in the plan are compounded primarily due to human error (Sutherland & Schwaber, 2007). Contrary to this, rapid application software development processes, or agile processes, involve the execution of development tasks in an iterative fashion. These agile processes are adaptive in nature because it allows developers to incorporate “late changes in the specifications” (Abrahamsson et al., 2017, p. 12). Agile methodologies place an emphasis on iterative and incremental development while eliciting customer input continuously (Sutherland & Schwaber, 2007). However, there are certain agile development methodologies that incorporate elements of traditional structured software development approaches. Methodologies such as Scrum involve the documentation of requirements initially instead of directly beginning with the implementation. UML can be used to document, describe, and model these requirements which will allow project teams to be better prepared to implement the requirements in code (Santos et al., 2016). Scrum is described as an "…iterative and incremental approach to delivering object-oriented software" (Schwaber, 1997, p.2). Scrum has its origins in manufacturing, originally introduced with goal of creating usable results in a timeframe of weeks (Takeuchi & Nonaka, 1986). Scrum is a software development process that is best suited for smaller teams (Rising & Janoff, 2000). It involves planning phase where the team develops a preliminary architecture and during development, the architecture is modified as required. Development is done incrementally and iteratively in a series of phases called sprints where each sprint lasts anywhere from one to four weeks. Each sprint implements a tangible and usable product that implements at least one user interaction with the system, referred to as user story (Rising & Janoff, 2000). The team keeps track of all identified tasks in a list referred to as the backlog. A set of tasks from the backlog as selected based on priority for each sprint and following each sprint, the backlog is updated, and the remaining tasks are reprioritized. During each sprint, short daily meetings are held involving all team members to discuss progress made, any obstacles that were faced during development, and planned progress. The team is led by a Scrum master who is responsible for selecting user stories to be completed in a sprint, recording decisions made at Scrum meetings, facilitating communication, and tracking action items (Rising & Janoff, 2000). Scrum has been widely used in industry with many companies reporting improvements in productivity and morale following its adoption (Sutherland & Schwaber, 2007).

Agile methodologies have also been used in educational settings with an accompanying improvement in student project success (Umphress, Hendrix, & Cross, 2002). In addition, agile software development using Scrum resulted in improved course perceptions in the context of an undergraduate software engineering capstone course (Mahnic, 2012). Further studies have recommended the use of agile methodologies in educational contexts citing the benefits of improved collaboration, equipping students with practical experience, increased productivity, and improved learning outcomes (Coupal & Boechler, 2005; Kamthan, 2016; Rico & Sayani, 2009; Shukla & Williams, 2002). The relevance of agile approaches to project management and Scrum lies in the learning design of the course that this study is centered around and it is further explained in Chapter 4.

CHAPTER 3. THEORETICAL FRAMEWORK

3.1 Complex Systems Reasoning

Complex interconnected systems have become increasingly pervasive and reasoning about these complex systems is difficult due to the requirement of having to employ abstract thinking (Hmelo-Silver & Pfeffer, 2004). This difficulty in understanding complex systems can be explained by virtue of complex systems involving local interactions across several levels of organization (Ferrari & Chi, 1998). The relationships existing across these different levels in a complex system are not necessarily immediately obvious or intuitive (Wilenksy & Resnick, 1999). Reasoning about complex systems faces additional barriers in the form of often invisible dynamic phenomena existing around more readily visible or available structures (Feltovich et al., 1992; Hmelo, Holton, & Kolodner, 2000). A significant load is placed on working memory to simultaneously process all the interactions that are happening across multiple levels of a complex system. This can be attributed to the specifics of the mental simulation process and the inferences required to build a comprehensive mental model (Narayanan & Hegarty, 1998). Complex systems may also exhibit emergent properties or behavior that may not be fully attributed to any individual component or structure in the system (Wilensky & Resnick, 1999). Causality is also difficult to establish in the context of complex system due to the presence of numerous intermediate steps between the actual cause and the observed effect (Perkins & Grotzer, 2000). Prior knowledge can also impede reasoning about complex systems due to prevalent individual preferences towards centralized thinking and single causality (Wilensky & Resnick, 1999; Jacobson, 2001) whereas experts reasoning about complex systems often display "…decentralized thinking, multiple causes, and the use of stochastic and equilibration processes" (Hmelo-Silver & Pfeffer, 2004, p.129).

3.2 Epistemic Forms and Games

Literature posits several different approaches to reasoning about complex systems. One approach is that of epistemic forms and games (Collins & Ferguson, 1993; Sherry & Trigg, 1996; Morrison & Collins, 1995; Shimoda & Borge, 2016). Epistemic forms refer to target structures that can guide inquiry often from a systems-dynamics modeling perspective (Hmelo-

Silver & Pfeffer, 2004). Epistemic forms can be used to organize knowledge while illustrating the relationships between different concepts (Sherry & Trigg, 1996). Epistemic games refer to strategies used to analyze phenomena with the goal of filling out a specific epistemic form (Collins & Ferguson, 1993). They are referred to as epistemic games because the combination of rules, strategies, and moves associated with a specific representation is used to construct new knowledge. Epistemic games are reflective in nature and result in the generation of knowledge. They include often complex rules, constraints, and entry conditions that guide the construction of the epistemic form (Sherry & Trigg, 1996).

Epistemic games can be categorized as follows: (i) Structural analysis games; (ii) Functional analysis games; (iii) Process analysis games (Collins & Ferguson, 1993; Sherry & Trigg, 1996). Structure analysis games are often described as the simplest or easiest of the three categories of games to implement and examples include but are not limited to primitive-elements analysis or spatial decomposition (Collins & Ferguson, 1993; Sherry & Trigg, 1996). Spatial decomposition for instance involves breaking down an entity into non-overlapping components while specifying the relationships between these parts. Constraints for this game includes specifying the connections between components and the nature of these connections. The goal of primitive-elements analysis is to describe large phenomena as being composed of primitive elements combining to achieve it (Collins & Ferguson, 1993).

Functional analysis games are used to determine "…causal or functional structures that relate elements in a system" (Collins & Ferguson, 1993, p. 33). Examples of functional analysis games include:

- Critical-event analysis: Focuses on the series of events that led to a specific critical event or the consequences the follow a critical event once it has occurred.
- Cause-and-effect analysis: It is a variant of critical-event analysis that draws a distinction between triggers (also referred to as causes) and preconditions – some condition that must be true for an effect to occur. An effect can also serve as a trigger for one or more new effects.
- Problem-centered analysis: Breaks down an event stream into problems and actions required to solve those specific problems. These solutions result in primary and secondary effects with secondary effects often being new problems that require solutions.

- Form and function analysis: Distinguishes between the form of an object and its functionality or purpose.

Process analysis games are used to describe the internal behavior of a system (Sherry & Trings, 1996). These games, and the forms associated with it, are used to analyze dynamic phenomena (Collins & Ferguson, 1993). Process analysis games are described as being the most complex and difficult. Systems-dynamics models consist of basic elements that are connected by positive or negative links occasionally with feedback loops existing within the system. Situation-action models are characterized by rules that model action to be taken based on conditions being satisfied. Situations are influenced by environmental changes as well as by actions taken by individual agents.

These three categories align with elaboration theory which advocates for structuring knowledge in terms of concepts, procedures, and theories (Reigeluth & Stein, 1983; Reigeluth, 2018). These three knowledge structures – conceptual, procedural, and theoretical – answer the questions of what a system is, how a system works, and why a system works the way it does (Sherry & Trings, 1996). Epistemic forms and games guide inquiry by providing the inquirer with constraints and context. Mastering epistemic games provides one with the ability use various epistemic forms to make sense of various phenomena (Collins & Ferguson, 1993). This introduces the concept of epistemic fluency that is defined as "…the ability to identify and use different ways of knowing, to understand their different forms of expression and evaluation, and to take the perspective of others who are operating within a different epistemic framework" (Morrison & Collins, 1995, p. 40). Epistemic fluency is the ability to organize knowledge into different patterns while making sense of a problem in different ways (Sherry & Trigg, 1996). Epistemic fluency develops through "…social interactions with other members of a community of practice, including those who are at least slightly more expert at playing these game" (Morrison & Collins, 1995, p. 43). Those who exhibit epistemic fluency will be able to use different epistemic games to determine the function of system, the structure of the system in terms of its interrelated components, and the behaviors or processes of the system in terms of how the structures accomplish the function (Collins & Ferguson, 1993; Morrison & Collins, 1995).

Literature widely demonstrates that experts in various domains organize their knowledge based on deep principles of their discipline (Chi, Feltovich, & Glaser, 1981). One approach towards organizing knowledge is in terms of visible structures of the system, however this has the drawback of not describing or modeling underlying functions (Perkins & Grotzer, 2000). Reasoning about complex systems requires individuals to create a network of concepts pertaining to the domain in question which can then be used to represent the micro and macro level interrelationships among its various structures (Hmelo-Silver & Pfeffer, 2004). The inquiry into epistemic forms and games provided precedent for employing structural analysis, functional analysis, and process analysis to describe and understand complex systems (Collins & Ferguson, 1993; Sherry & Trigg, 1996; Morrison & Collins, 1995; Shimoda & Borge, 2016). The Structure-Behavior-Function theory accounts for the dynamic nature of the numerous interconnected levels that exist in complex systems (Hmelo-Silver & Pfeffer, 2004) and align well with the structural, functional, and process analyses detailed earlier. The SBF framework and its application in software modeling has been discussed below.

3.3 SBF Framework

The SBF framework, sometimes denoted as Functions-Behaviors-Structures (FBS), has its roots in cognitive science and has been successfully applied to many diverse domains (Gero, 1990; Gero & McNeill, 1998; Gero & Kannengiesser, 2004; Hmelo-Silver & Pfeffer, 2004; Lammi, 2011). The SBF framework aligns well with the epistemic games of structural, process, and behavioral analyses (Collins & Ferguson, 1993) as well as with existing approaches for modeling information systems using functional, structural, and behavioral views (Dennis, Wixom, & Tegarden, 2020). The use of multiple interconnected views to describe software systems was popularized by the Rational Unified Process (RUP). RUP advocates for the use of a logical view that is focused on describing end-user functionality and a process view that describes how the functionality is implemented (Eriksson, Börstler, & Borg, 2005; Robal, Viies, & Kruus, 2002), which necessitates the use of models to describe: i) the functionality of the information system as viewed by the end user – functional view; ii) the objects along with their corresponding attributes and behaviors that implement the functionality – structural view; and iii) how the different objects interact with one-another to achieve the required functionality.

This approach to information systems development aligns well with the SBF framework and the SBF framework can be used to reason about the functional roles of structural elements in complex systems (Hmelo-Silver & Pfeffer, 2004). This can be done by describing the purpose of individual subcomponents of a system and how they work together to bring about a certain functionality. Structures can refer to the artefacts or objects themselves and the relationships between them (Krutchen, 2005) or to individual elements of a system (Hmelo-Silver & Pfeffer, 2004). Functions can refer to the purpose of a design artefact or the exact reason for why specific components exist. Behaviors describe the actions or processes of an artefact or object and how the structures of the system interact to implement a certain function. The SBF framework has been used successfully in interactive learning environments to create knowledge representations of complex systems (Vattam et al., 2001). It must be noted that learners tended to struggle with differentiating between behaviors and functions of a system because the functional aspects focus on often tangible outcomes while behavioral aspects typically include intrinsic mechanisms that are often difficult to represent (Hmelo-Silver & Pfeffer, 2004; Lammi, 2011; Vattam et al., 2011). Also noteworthy was how functional aspects of a system are often implicit and difficult for novices to infer (Chi, De Leeuw, Chiu, & Lavancher, 1994). Structures of complex systems are most readily observed by novices or beginners. However, experts typically use behaviors and functions to organize their knowledge pertaining to a specific system. This could be explained by how the overall functionality of a system is accomplished by combining the different behaviors and structures. This discrepancy between experts and novices could also be attributed to how novices may encounter difficult in connecting phenomena observed at microlevel to those observed at a macrolevel or vice-versa (Penner, 2000).

The SBF framework has been successfully used in physics, medicine, engineering and history (Hmelo-Silver & Pfeffer, 2004; Lammi, 2011) but it has not been applied in information systems context. Given that the SBF framework has been used for complex systems and information systems are modeled in terms of functions, structures, and behaviors, there is reason to believe that the SBF framework can be applied successfully in the context of software modeling and information systems. It can be posited that the UML serves as the epistemic form and the SBF framework serves as the epistemic game for performing structural, process, and functional analysis of information systems. Information systems can be complex and require those developing them to understand the various relationships and connections between the micro and macro elements of the system. Similar to biological systems, the functionality of information systems tends to be implicit and there are interactions between the different components of the system. Thus, understanding the structures, behaviors, and functions of an information system becomes an integral part of software modeling and indirectly an important part of the software development process.

3.4 Implications

For this study, the UML diagrams were the epistemic forms that guide the inquiry and description of an information system. Use-case narratives and UML activity diagrams can be used to capture the interactions between the user and the information system. This aligns with the epistemic game of functional analysis. The use-case narratives and activity diagrams serve as the epistemic form to capture the functional aspects of the information system. Learners had to employ the epistemic games of critical-event analysis, cause-and-effect analysis, and problem-centered analysis to determine the triggers and preconditions for specific use cases while delineating how the system and user will interact in various scenarios. UML class diagrams can be used to represent the objects of the system which aligns with the epistemic game of structural analysis. Class diagrams serve as the epistemic form to capture the structural aspects of the information system. Learners had to employ spatial decomposition and primitive-element analysis to determine what the objects are and the nature of the relationships between them for the system given in the specific problem statement. UML sequence diagrams can be used to capture the interactions between objects of an information system which aligns with the epistemic game of process analysis. Sequence diagrams serve as the epistemic form to capture the behavioral aspects of the information system. The internal behavior of the information system can be described by capturing the messages and responses between objects. To proficiently model functional, structural, and behavioral aspects of an information system, learners will have to display a high degree of epistemic fluency. This UML diagrams were evaluated quantitatively, and the details were discussed in Section 4.4.5. The epistemic games of functional analysis, structural analysis, and process analysis were analyzed qualitatively to determine how participants employed these games.

The implications of the theoretical framework influenced how functions, structures and behaviors were operationalized in the context of information systems and how UML models were utilized to capture these details. Table 3.1 illustrates how the UML diagrams aligned with each of the elements of the SBF framework and the different epistemic games (Collins & Ferguson, 1993; Dennis, Wixom, & Tegarden, 2020; Hmelo-Silver & Pfeffer, 2004).

Table 3.1: Alignment between Epistemic Games, UML Models, and SBF Constructs

Epistemic Game	SBF Construct	Construct Definition	UML Model (Epistemic Form)	Study Definition
Functional analysis	Functions	Specific purpose or services that the system must provide	Use-Case Narratives and Activity Diagrams	Flow of control between the user and system, and within the system as it executes a specific functionality.
Structural analysis	Structures	Specific components of a system	Class Diagrams	Attributes and behaviors of each object and how the objects relate to one-another.
Process analysis	Behaviors	How the components of a system work together to achieve a specific purpose or functionality	Sequence Diagrams	The interactions between the different objects, in the form of messages and responses, involved in implementing a certain functionality.

CHAPTER 4. METHODOLOGY

This chapter described the research design and the various procedures that were employed for data collection and analysis. The rationale behind the selection of the research paradigm and research methods were discussed. This chapter further detailed research context and participant details. This study was approved by institutional review board with protocol numbers 1709019656 and IRB-2019-393.

4.1 Research Paradigm

Literature points to various research paradigms associated with educational research such as: (i) Postpositivism; (ii) Constructivism; (iii) Critical Theory; and (iv) Pragmatism. While this is in no way meant to be an exhaustive list, each paradigm is distinct regarding the following parameters (Creswell & Plano Clarke, 2018; Jones, Torres, & Arminio, 2014):

- Ontology – the nature of reality
- Epistemology – the relationship between the researcher and what is being researched
- Axiology – the role of values
- Methodology – the research process
- Rhetoric – the language of research

The primary consideration was the alignment of the research paradigm with the research area being examined – namely that of systems analysis, software modeling, abstraction, and systems thinking; and which paradigm would serve as the most useful lens for exploring the research questions of this study. Constructivism and critical theory were quickly excluded because of the apparent lack of alignment with the research area of information systems modeling. Following this, the approaches of postpositivism and pragmatism were considered due to alignment with the research area in terms of epistemology, axiology, and methodology. Ultimately, pragmatism was chosen as the most suitable lens for this study as it allows for adopting the approaches that work best to answer the research questions at hand. In the realm of systems analysis there are often multiple correct answers or solutions while not precluding the existence of wrong answers which aligns well the philosophical assumptions of pragmatism which point to the existence of a singular reality or multiple realities (Jones, Torres, & Arminio, 2014). Pragmatism also emphasizes the use and interpretation of data collected through multiple methods in manner that works best to answer the research questions that are being explored (Creswell & Plano Clark, 2018).

4.2 Research Design

Mixed methods research that combines quantitative and qualitative data has been used to great effect in the recent past various fields including those of education and social sciences (Creswell, 2014). When employing a mixed-methods design, the following four factors must be considered: (i) theoretical perspective; (ii) strategy priority; (iii) implementation sequence; and (iv) point of data integration (Terrell, 2012). Theoretical perspective refers to whether the study is directly or indirectly based on a theory. Priority of strategy refers to which data is considered to be more important in the overall context of the study or if they are to be given equal importance. The implementation sequence refers to the order of data collection – whether quantitative or qualitative data is collected first. The point of data integration refers to the exact point of the study where the data from the two phases are integrated and discussed. This can be done at collection, analysis, or interpretation (Ivankova, Creswell, & Stick, 2006; Terrell, 2012). At first glance, an explanatory mixed-methods approach would be best suited to accomplish the goals of this research (Creswell & Plano Clark, 2018). The explanatory sequential design first involves the collection of quantitative data. The results from the analysis of this data provides a general or high-level picture of the problem. The qualitative follow-up provides explanations for the general picture. However, this approach requires that both phases of the study be performed on the same sample so as to facilitate integration and this encounters pragmatic concerns related to subject recruitment. The same limitation applies to a triangulation design (Ivankova & Creswell, 2009). Adhering to any of the traditional mixed methods research designs would not be suitable for the goals of this study.

An alternative to this would be adopt a multimethod or multiple methods approach that incorporates multiple forms of data collection with the goal of addressing the research questions (Anguera et al., 2018). The differences between mixed methods and multimethod research designs are subtle yet significant even though the terms are sometimes used interchangeably in literature (Stange, Crabtree, & Miller, 2006). Mixed methods approaches mandate the integration of quantitative and qualitative components to mix the often complementary information that they carry while multimethod studies are driven by an overall research goal which is achieved by the use of complimentary methodologies (Anguera et al., 2018).

Like mixed methods studies, multimethods studies can have varying degrees of emphasis placed on the quantitative and qualitative aspects of the study (Anguera et al., 2018). The notation QUANT + QUAL indicates that both methods have equal emphasis (Creswell & Plano Clark, 2018). QUANT + qual indicates that the emphasis is unequal with more emphasis being placed on the quantitative methods whereas quant + QUAL indicates unequal emphasis with more emphasis being placed on qualitative methods. A triangulation design was adopted for this study that comprised of two distinct strands– a quantitative strand that was followed by a qualitative study denoted by QUANT + qual. Figure 4.1 illustrates the overall structure and format of the study while providing details about each step.

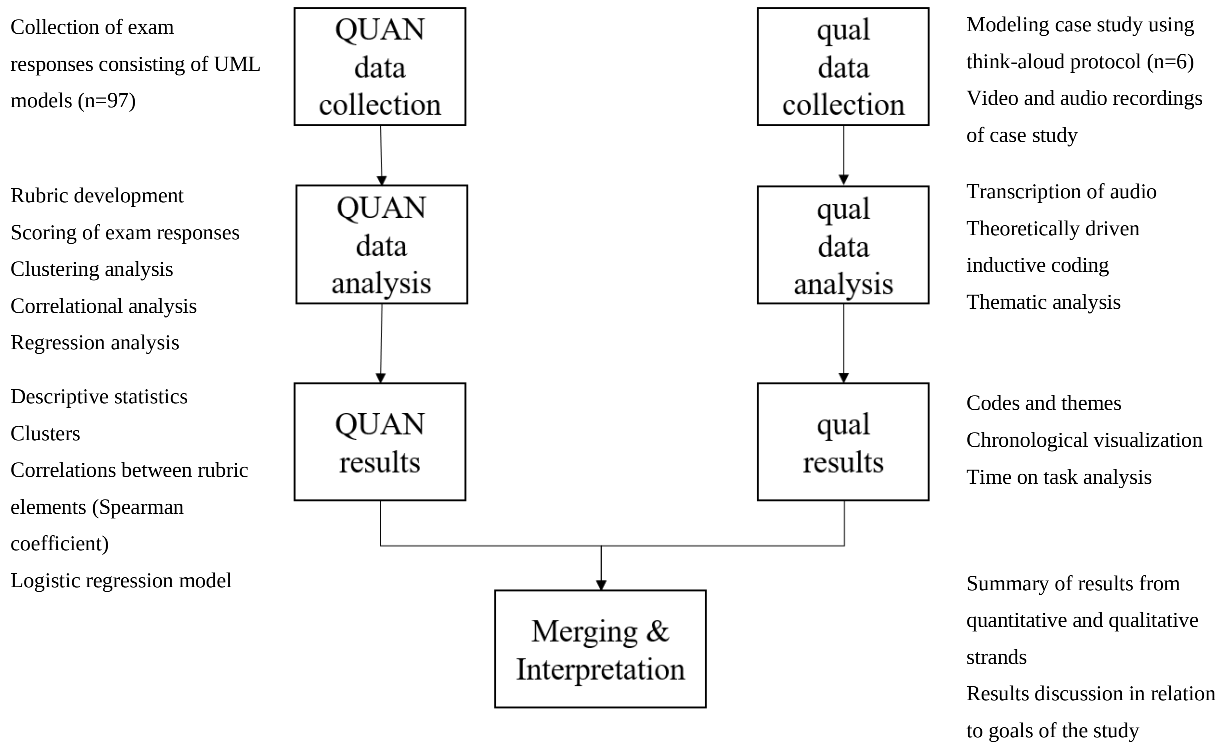

Collection of exam responses consisting of UML models (n=97)
QUAN data collection
qual data collection
Modeling case study using think-aloud protocol (n=6)
Video and audio recordings of case study
Rubric development
Scoring of exam responses
Clustering analysis
Correlational analysis
Regression analysis
QUAN data analysis
qual data analysis
Transcription of audio
Theoretically driven inductive coding
Thematic analysis
Descriptive statistics
Clusters
Correlations between rubric elements (Spearman coefficient)
Logistic regression model
QUAN results
qual results
Codes and themes
Chronological visualization
Time on task analysis
Merging & Interpretation
Summary of results from quantitative and qualitative strands
Results discussion in relation to goals of the study

Figure 4.1: Flow chart for research design

The quantitative strand was centered around a sophomore level course in systems analysis and design methods. This approach aimed to identify the different patterns of software modeling proficiency displayed by learners. Details about the course were explained in Section 4.3. The quantitative strand involved the collection and analysis of exam-responses from the second mid-term of this course. The qualitative strand involved participants who had completed the aforementioned course and were of junior or senior standing. This strand explored how learners utilized abstract thinking and systems thinking while constructing software models. A case study approach using the think-aloud protocol was utilized for the qualitative strand. Each participant in the qualitative strand, following the completion of all activities, was awarded an Amazon gift card valued at $25 as compensation for their time and inconvenience. Further details of the quantitative strand and qualitative strand were elaborated on in Section 4.5. Both studies leveraged the Structures-Behaviors-Functions (SBF) framework detailed below.

4.3 Learning Design

This section details the learning context of the course that the study was centered around. The course was designed to integrate Scrum methodology and cooperative learning. Scrum is an approach to product development where teams achieve goals in an iterative fashion with each iteration incrementally building upon the previous one. Cooperative learning was incorporated in this course with the goal of promoting the skills of teamwork, communication, and problem solving while learning the different techniques and approaches involved in conducting system analysis and design (Magana, Seah, & Thomas, 2017). As students analyze and design the system as part of their term project, they are required to create UML software models that capture functions, structures, and behaviors of the system. Students work in teams making using of the Scrum approach to iteratively develop functional prototypes. The project comprises of four milestones and one final deliverable for which students are to submit documentation. Each of these milestones were completed in increments called sprints and each sprint was delivered in a week. Student teams were provided feedback at the end of each sprint. The goal of the project is to create a functional prototype and a detailed design document as part of their team project that captures the “functional, structural and behavioral views of the system” (Magana, Seah and Thomas, 2017). While defining system requirements and developing the functional prototype, students will have to employ abstract thinking to identify relevant systems requirements (Ghezzi et al., 2002; Kramer, 2007), and use systems thinking to account for how the different components of the information system are connected (Brewer & Dittman, 2018) and how making modifications to one component will impact the others. The learners in this course are also provided instructional scaffolds in the form of in-class active learning activities such as walkthroughs of software modeling which allows them to hone their abstract thinking and systems thinking skills. This is illustrated in the conjecture map shown in Figure 4.2, where students by engaging in the Scrum-based team project and the in-class modeling activities would produce UML software models that capture functions, structures, and behaviors of the system., which in turn should improve abstract thinking, systems thinking, and overall software modeling proficiency. Figure 4.2 shows the conjecture map of the different elements involved in this study. The conjecture map is a high-level graphical representation of how the desired learning outcomes were promoted.

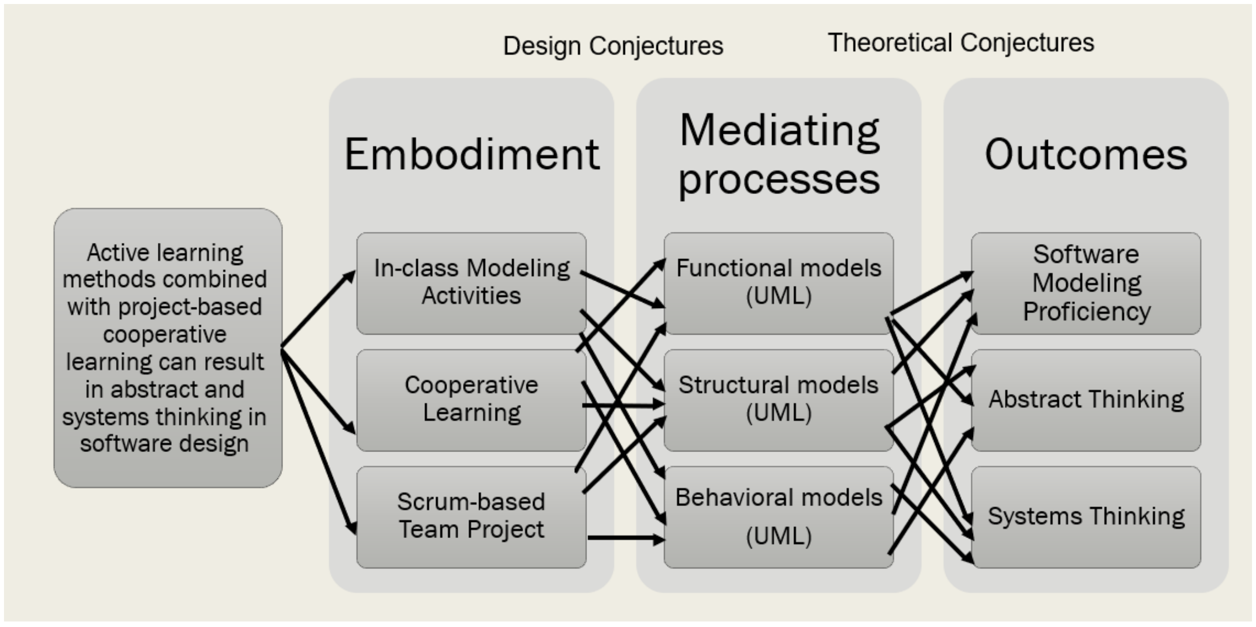


Figure 4.2: Learning design conjecture map

The learning theory that guided the design of the learning environment was social constructivism. Social constructivism posits that knowledge and understanding are developed through coordination with others (Amineh & Asl, 2015). Scholars of social constructivism suggest that individuals learn through collaboration and interaction with others (Kim, 2001). Under social constructivism, learners actively pursue knowledge through the discovery of concepts and facts. Emphasis is placed on learner interactions with those who are considered knowledgeable or experts in specific subjects (Amineh & Asl, 2015). Social constructivism highlights the importance of collaboration among learners as well as instructors taking on the role of facilitators that provide guidelines and foster an environment where learners can arrive at their own solutions.

The embodiment detailed characteristics and features of the learning environment which in this case involves in-class modeling activities where the students are walked through diagramming exercises by the instructor alongside cooperative learning implemented via a Scrum-based team project. The mediating processes referred to those salient performances or products expected to result from the embodied elements. Learners engaged with the various tasks of the course, and employ the epistemic games of functional analysis, structural analysis, and process analysis to produce UML software models as artifacts. These models were the epistemic forms that captured details about the functions, structures, and behaviors about information systems. The outcomes were the result of the mediating processes and these are the elements that are ultimately measured (Sandoval, 2014). The goals of this course were to improve software modeling proficiency exhibited by students alongside their abstract thinking and systems thinking skills.

In summary, the design conjecture was that if learners engage in project-based cooperative learning via in-class modeling activities and the Scrum-based team project, then UML software models – detailing functions, structures, and behaviors; would emerge as a mediating process. The theoretical conjecture was that the development of UML software models would lead to learning outcomes of increased software modeling proficiency, abstract thinking skills and systems thinking skills. The learning environment was social constructivist in nature with students actively involved in the learning process through interaction and collaboration with their peers and the instructor.

4.4 Quantitative Strand

This section discussed the details of the quantitative study. The quantitative strand was centered around a sophomore level systems analysis and design course. The course explored systems development techniques and approaches used by IT professionals such as developers and analysts to model the requirements of an information system, and then construct an acceptable design which is then implemented as a solution (Magana, Seah, & Thomas, 2017).

4.4.1 Research Questions

The quantitative strand aimed to answer the following research questions:

1. To what extent to did students demonstrate proficiency in abstract thinking while analyzing software systems in terms of functions, structures, and behaviors?
2. To what extent to did students demonstrate proficiency in systems thinking while analyzing software systems in functional, structural, and behavioral representations?
3. What were the characteristics or profiles of students in terms of abstract thinking and systems thinking ability as evidenced by their system representations?

4.4.2 Participants

This study included 97 students, majority of whom were second-year computer and information technology students. Students in this course have had prior exposure to at least one introductory systems development course and have knowledge of programming either from coursework or practical experience via internships.

4.4.3 Procedures and Data Collection Methods

This course included several forms of assessment including but not limited to class participation, quizzes, documentation of a semester-long project, final presentation of the term project, and three written examinations. The examinations are meant to evaluate conceptual knowledge alongside modeling ability. The modeling aspects of the second exam of this course served as the data for this study. The second exam (Appendix A. Modeling Exam) was held in the 11th week of the semester. The students had participated in multiple guided in-class modeling activities where they followed the instructor in capturing functions, structures, and behaviors of information systems of example case studies. Also, the students had participated in multiple milestones of the Scrum-based team project that requires teams to develop a functional prototype and detailed system specifications – which include functional, structural, and behavioral representations of the system.

The exam consisted of two distinct parts. The first part involved the learners answering multiple-choice questions that tested their conceptual knowledge and the second part consisting of a case study for which the students were requested to model the system in terms of functions, structures, and behaviors using the appropriate UML models. The case study primarily presented details of an online seat reservation system and that is the functionality for which students were expected to construct the models. The case also provided details of other functionality offered by the system such as reservation cancelation and payment which would additionally test the students' abstraction ability in terms of whether they included these details or not. Due to logistical and time-related constraints, the exam was conducted across two class sessions. In the first class-session, the students completed the multiple-choice questions, use-case narrative, and activity diagram. In the second class-session, the students completed the class diagram and sequence diagram.

4.4.4 Rubric Development

As discussed in the literature review, there were no standard approaches for measuring abstract thinking in the context of software modeling. Therefore, rubrics were developed to evaluate each UML model based on accuracy and conformance to the UML standard. The rubrics were designed with inputs from faculty in the department of computer and information technology with cumulative decades of experience teaching systems development courses. Each rubric was designed to account for 5 levels of student performance. The rubrics were designed to account for five levels of student performance. They performance levels, in general, are detailed below:

- Absent - A rubric element was graded as absent and scored 0 if a student did not attempt to address it at all.
- Deficient - An element was graded as deficient and scored 1 if a student attempted to address the criterion but only captured between 10 and 50% of the expected details.
- Developing - A rubric element was graded as developing and scored 2 if the student has addressed the criterion but only captured 51 and 70% of the expected details.
- Emerging - An element was graded as emerging and scored 3 if a student addressed the criterion but only captured between 71 and 90% of the expected details.
- Proficient - A rubric element was graded as proficient and scored 4 if the student addressed the requirement and captured 91% or more of the expected details.

The rubric for evaluating use case narratives (Appendix C - Use Case Narrative Rubric) consists of three elements. The element "Typical course of events" refers to steps that a system would normally follow and is used to evaluate what percentage of these steps have been captured by the learner. The element “Alternate courses” refers to steps taken by the system if certain specific criteria or conditions are satisfied and is used to evaluate what percentage of these steps have been captured by the learner. The element “Narrative completeness” refers to the degree to which the learner has completed the different components of the use-case narrative template (Appendix B – Use Case Narrative Template) outside of the typical and alternate courses.

The rubric for evaluating activity diagrams (Appendix D – Activity Diagram Rubric) is informed in part by the UML standards discussed in Section 2.5.1. The rubric was used to evaluate functional modeling proficiency in terms of conformity to UML standards alongside proficiency of abstraction and alignment with respect to the use-case narratives. The first four components of the rubric – Start/Stop nodes, Swimlanes, Activities, and Decisions/Merges address UML conformity. The component titled “Abstraction” refers to the percentage of relevant details from the problem statement that were included in the diagram. “Alignment with use case narrative” refers to the percentage of actions and decisions in the activity diagram that can be mapped to steps in the typical course of events and/or alternate courses in the use-case narratives.

The rubric for evaluating class diagrams (Appendix E – Class Diagram Rubric) is informed in part by the UML standards discussed in Section 2.5.2. The rubric was used to evaluate structural modeling proficiency in terms of conformity to UML standards alongside proficiency of abstraction. The first four components of the rubric – Objects, Attributes, Behaviors, and Relationships address UML conformity. “Abstraction” refers to percentage of relevant details included in the class diagram.

The rubric for evaluating sequence diagrams (Appendix F – Sequence Diagram Rubric) is informed in part by the UML standards discussed in Section 2.5.3. The rubric was used to evaluate behavioral modeling proficiency in terms of conformity to UML standards alongside proficiency of abstraction, and alignment with respect to class and sequence diagrams. “Abstraction” refers to the percentage of relevant details included in the sequence diagram. “Alignment with class diagram” refers to the percentage of objects of objects and messages in the sequence diagram that can be mapped to the objects and functions in the class diagram. “Alignment with activity diagram” refers to the percentage of messages and responses in the sequence diagram that can be mapped to actions in the activity diagram. This component only checked for the presence of these messages or responses that can be mapped to the activity diagram and did not evaluate whether the messages and responses originate from or go to the appropriate objects.

4.4.5 Data Scoring and Data Analysis Methods

The analysis of the exam responses allowed for an overall assessment of the learners' software modeling proficiency in terms of capturing the functional, structural, and behavioral aspects of information systems. Software modeling proficiency was categorized by proficiency exhibited by learners in terms of modeling: i) functions of an information system by using use case narratives and activity diagrams; ii) structures of an information system by using class diagrams; and iii) behaviors of an information system by sequence diagrams respectively.

The use case narratives, the activity diagrams, class diagrams, and sequence diagrams were evaluated for completeness and accuracy. In accordance with considerations of the Institutional Review Board (IRB) pertaining to maintaining confidentiality and anonymity, all student names were replaced with a pseudonym. Use-case narratives were evaluated for overall completeness as well as completeness of typical and alternate courses of events. Overall scoring of use-case narratives can be illustrated with an example. For instance, if a use case narrative was scored as proficient (4) for typical courses of events, emerging (3) for alternate course of events, and developing (2) for narrative completeness, it would have a total score of 9 out of a possible 12. The details of scoring each criterion was given in Appendix B – Use Case Narrative rubric. The activity diagrams were assessed for conformity to UML standards as established in the course, how much relevant detail, pertaining to functionality of the system, was incorporated; and how well students mapped to the use case narrative which was listed as the alignment component on the rubric. Overall scoring of activity diagrams can be illustrated with an example. For instance, if an activity diagram was scored as developing (2) for start/stop nodes, proficient (4) for swimlanes, proficient (4) for activities, emerging (3) for decisions/merges, emerging (3) for abstraction, and proficient (4) alignment with use case narrative, it would have a total score of 13 out of a possible 22. The last two criteria were excluded from calculating the total score for activity diagrams to evaluate the relationships between abstract thinking, systems thinking, and functional modeling proficiency. The details of scoring each criterion were given in Appendix D – Activity Diagram rubric.

The class diagram was evaluated for conformity to the UML standards as established in the course and whether it captured the overall structural representation of the system in terms of the objects and their respective attributes and behaviors. Overall scoring of class diagrams can be illustrated with an example. For instance, if a class diagram was scored as proficient (4) for objects, emerging (3) for attributes and behaviors, proficient (4) for relationships, and emerging (3) for abstraction, it would have a total score of 14 out of a possible 16. Abstract thinking was excluded from the total score to evaluate the relationship between abstraction and structural modeling proficiency. The details of scoring each criterion were given in Appendix D – Class Diagram rubric. The sequence diagrams were assessed for conformity to UML standards as established in the course and for how well they mapped to the class diagrams and activity. For example, how many of the classes drawn in the sequence diagram were listed as objects in the class diagram or whether the overall flow of the activity diagram was captured in the sequence diagram. Overall scoring of sequence diagrams can be illustrated with an example. For instance, if a sequence diagram was scored as proficient (4) for objects, lifelines, and processes; emerging (3) for messages, responses, and abstraction; and proficient (4) for alignment with class diagram and alignment with activity diagram, it would have a total score of 18 out of 20. The last three criteria were excluded from the total score calculation evaluate the relationships between abstraction, systems thinking, and behavioral modeling proficiency. The details of scoring each criterion were given in Appendix C – Sequence Diagram rubric. Once the data had been scored, patterns of student performance was illustrated through a descriptive analysis. The descriptive analysis revealed percentage distributions of students for each performance level per rubric element in a specific rubric.

4.4.6 Reliability and Validity Considerations

Face validation (Creswell & Poth, 2016) of the rubrics was performed by a professor in the department of computer and information technology who has several years of experience teaching systems development courses. There is precedent in literature to ensure reliability for larger data sets by having an independent coder or rater score a subset of the available data to then compute interrater reliability (Hammer & Berland, 2014). To ensure reliability of the data scoring process, 20% of the student responses were scored by a second rater who had previously served as teaching assistant for the course and possesses an expertise in systems analysis and design. Interrater reliability was evaluated on the total scores – as scored by the individual raters using the rubrics - for use case narratives, activity diagrams, class diagrams, and sequence diagrams using Cronbach's Alpha. Interrater agreement for each component of the exam as given by Cronbach's alpha is shown below in Table 4.1.

Table 4.1: Interrater Agreement

Exam Component	Cronbach's alpha
Use Case Narratives	.650
Activity Diagrams	.931
Class Diagrams	.900
Sequence Diagrams	.982

4.4.7 Cluster Analysis

While the descriptive analysis illustrated broad patterns of proficiency in terms of the different rubric, they did not reveal any information about underlying groups within the dataset. To investigate the data further, and to determine whether there were any characteristics of students that were not revealed in the descriptive analysis, clustering analysis was employed. Clustering is the process of grouping data and clustering algorithms forms groups based on similarities between the objects under consideration (Ordonez, 2003; Thinsungnoena et al., 2015; Yuan & Yang, 2019). These groups are referred to as clusters and are disjoint subsets of the given dataset (Likas, Vlassis, & Verbeek, 2003). The clusters are characterized by external separation and internal homogeneity which means that patterns in different clusters should be dissimilar while patterns within a cluster should be similar and comparable (Abbas, 2008; Xu & Wunsch, 2005). Clustering is an example of unsupervised classification and does not require any kind of prior training (Chaovalit & Zhou, 2005). It should not be confused with supervised classification which uses a collection of labeled or pre-classified patterns to label new data points (Abbas, 2008). There are several algorithms for data clustering, however, the challenge for most clustering algorithms lies in determining the number of clusters existing in a dataset beforehand. The data for cluster analysis was the scored exam responses as per the rubrics discussed in Section 4.4.5. The data was ordinal in nature with each subject having scores of 22 rubric elements associated with them.

4.4.8 Cluster Identification

To perform clustering analysis, the number of underlying clusters was first identified. The silhouette technique or silhouette coefficient algorithm was used to estimate the optimal number of clusters in a given dataset. It also serves as a tool examine the validity and quality of clustering (Aranganayagi & Thangavel, 2007; Yuan & Yang, 2019; Zhu, Ma, & Zhao, 2010). The silhouette technique groups objects based on tightness or cohesion and separation (Yuan & Yang, 2019). The technique is constructed to select the optimal number of clusters (Thinsungnoena et al., 2015). The silhouette coefficient algorithm involves the calculation of the contour coefficient which is defined based on intra-cluster dissimilarity and inter-cluster dissimilarity and takes on a value between -1 and 1 (Yuan & Yang, 2019). Higher silhouette values for each object indicates a close relationship between the object and the cluster while lower values indicate that it may have been assigned to the wrong cluster (Anitha & Patil, 2019). Higher average values of the contour coefficient or silhouette scores indicate a more effective cluster number. Silhouette coefficients are broadly accepted as a standard measure of cluster validation (Wang et al., 2010). Table 4.2 detailed the silhouette scores for the collected data:

Table 4.2: Silhouette scores

Number of Clusters	Silhouette Scores
2	0.303
3	0.189
4	0.188
5	0.188
6	0.191
7	0.195
8	0.141

4.4.9 Clustering Method

The silhouette scores indicated the cluster number of two was most optimal for the collected data. Therefore, a binary clustering algorithm is most appropriate. Thresholding After Random Projection (n-TARP), a binary clustering machine learning algorithm based in Python, was used to analyze the scored data. n-TARP was chosen due to it being "…computationally inexpensive, scalable to high dimensions, and can be easily modified to handle both very small and large datasets" (Yellamraju & Boutin, 2018, p.4). The n-TARP software projects the dataset *n* number of times into a random single-line plane. Then the algorithm thresholds the projected data at a point and identifies any clustering of data before or after the threshold. Finally, it chooses the two best clusters of the projection (Yellamraju & Boutin, 2018). The dataset was projected 100 times with a threshold of .3 for clustering. To implement this program, each rubric element was treated as a data point for each student while each of the diagrams was considered as an individual test scenario. Following this process, the data was analyzed in graphical form using grouped bar-graphs. The emergent clusters were examined for differences in abstract thinking and systems thinking ability using a t-test (Cohen, 2001).

4.4.10 Correlational Analysis

A correlational analysis is appropriate when a researcher aims to answer questions about a sample without manipulating variables or executing random assignment (Devlin, 2018) and it can be used to "…investigate an area of interest to get some idea of the strength of naturally occurring relationships" (Devlin, 2018, p.20). Therefore, a correlational analysis can be employed to characterize the relationship between student abstract thinking ability, systems thinking ability, and software modeling proficiency in terms of functions, structures, and behaviors. Spearman correlation coefficient was computed to explore the nature of the relationship between the various rubric elements. Spearman correlation coefficient is widely utilized in research where a non-parametric equivalent to Pearson's correlation coefficient is required either due to the non-normal distribution of data or its ordinal nature (Artusi, Verderop, & Marubini, 2002; Bonett & Wright, 2000; de Winter, Gosling, & Potter, 2016).

To characterize the relationship between student systems thinking ability and software modeling proficiency in terms of functions, structures, and behaviors, Spearman correlation coefficient was first computed for the entire dataset to explore: i) the relationship between student proficiency with the use case narratives and student proficiency in activity diagrams; ii) the relationship between student proficiency with class diagrams and student proficiency in sequence diagrams; iii) the nature of the relationship between software modeling proficiency in terms of functions, structures, and behaviors, and systems thinking ability. Correlational analysis was performed using IBM SPSS statistical software. As part of this analysis, a correlation table – containing rubric elements pertaining to abstract thinking, systems thinking, and software modeling proficiency was computed for the entire dataset. Following this, the correlational analysis was performed for each cluster revealed by the clustering analysis to investigate whether there are any differences in relationships between these elements across the clusters.

4.4.11 Regression Analysis and Logistic Regression

While a correlational analysis could be used to understand the strength of relationships, it did not provide an equation that connected the dependent variable with the explanatory variables. A regression analysis could be used to explore functional relationships between variables and these relationships could be expressed in the form of model (Chatterjee & Hadi, 2015). The regression model could be used to predict the degree to which abstract thinking and systems thinking proficiency influenced software modeling proficiency in terms of functions, structures, and behaviors in the context of the identified clusters. Given the ordinal nature of the data and the existence of two clusters, binary logistic regression was most appropriate.

Logistic regression can be utilized when the dependent variables are categorical in nature (Field, 2009). Logistic regression can be used to predict the increase or decrease in the probability of possessing a characteristic based on a unit increase in a specific independent variable while the other independent variables are held constant. A higher predicted value indicates that it is "...more likely that any individual with particular scores on the independent variables will have a characteristic…" (Field, 2009, p. 1). For the purposes of this study, the dependent variable was the cluster to which the student was assigned and can have the values "high performing" or "moderate performing" corresponding to each cluster. The independent variables were the rubric elements corresponding to abstract thinking and systems thinking in terms of functions, structures, and behaviors.

To test the goodness-of-fit of the proposed regression model, the Hosmer and Lemeshow test was used (Archer and Lemeshow, 2006). The Hosmer and Lemeshow test is based on the null hypothesis that the regression model is correctly specified and fits the data well. A statistically significant result for the Hosmer and Lemeshow test would lead to the null hypothesis being rejected and indicates that the proposed model is not a good fit for the given data. A result that is not statistically significant indicates that the proposed model is a good fit for the given data.

A possible regression model is given in (1):

$$\log\left(\frac{prob_{high}}{prob_{mod}}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n \qquad (1)$$

The response variable was given by $\log\left(\frac{prob_{high}}{prob_{mod}}\right)$ and it referred to the odds of a student being high performing versus the odds of a student being moderate performing. $X_1, X_2, \ldots, X_n$ were the predictor variables which in this case refers to individual rubric elements corresponding to abstract thinking and systems thinking in terms of functions, structures, and behaviors (Chatterjee & Hadi, 2015). The initial model contained all relevant predictor variables. A backward step-wise approach was taken to eliminate predictor variables and to arrive at the reduced model containing only the most relevant variables (Wang et al., 2007).

4.5 Qualitative Strand

While the quantitative approach yielded insights into what the patterns of the software modeling proficiency were among students and the relationship between software modeling proficiency and the constructs of abstract thinking and systems thinking, it does not provide any indication as to how abstract thinking or systems thinking was used while constructing software models. The qualitative strand was centered around exploring how students utilize abstract thinking and systems thinking while constructing software models based on a given problem statement or case.

4.5.1 Research Question

The qualitative strand was guided by the research question - How do students use abstract thinking and systems thinking when modeling software systems in terms of functions, structures, and behaviors?

4.5.2 Participants

There were six participants recruited for this study, all of whom have completed the sophomore level systems analysis and design course (Magana, Seah, & Thomas, 2017) detailed in the quantitative study. These participants were part of the computer and information technology department and were either of junior or senior standing. In total, these students had completed a minimum of two systems development courses and two programming courses.

4.5.3 Procedures and Data Collection Methods

For the qualitative phase of this study, a case study approach was used. A case study is an exploration of a case over time involving data collection from multiple sources of context-rich information (Merriam, 1998). A case study can involve the exploration of a single bounded system or multiple bounded systems through qualitative approaches such as observations, interviews etc. (Baxter & Jack, 2008; Creswell et al., 2007). Research designs involving the use of case studies have been widely used in fields such as business, education, medicine, psychology, and many more (Gerring, 2006). A case refers to an entity that includes but is not limited to educational institutions or programs and departments within educational institutions (Stake, 2006). A bounded system or multiple bounded systems defines the extent of what will be explored as part of the study as well as what will not be explored (Baxter & Jack, 2008). For the qualitative strand of this study, a single case study approach was employed. The case was bounded by the requirement that the participants, who are the units of analysis, have completed the sophomore level systems analysis and design course detailed in Chapter 4. Prior to recruitment, prospective participants were briefed about the activities involved as part of the case study and the compensation they would receive for their participation.

Each participant was provided a case centered around staff management (Appendix G – Staff Management Case) and they were to model the "Staffing Request" functionality in terms of functions, structures, and behaviors. The case itself details numerous functionalities associated with a staff management system and it provides steps and conditional actions associated with these functionalities as well as the different actors who are involved with the functioning of the system. This specific case was chosen because of the sufficient complexity it offered that requires participants to employ abstract thinking to include relevant details while excluding those details that are not pertinent to the solution. Participants also had to employ systems thinking to align their different models. The participants were given three hours to construct a use case narrative, activity diagram, class diagram, and sequence diagram. While constructing the models, the participants were asked to think-aloud to justify their design decisions and rationale. The think-aloud protocol is a data-elicitation method where participants are asked to verbalize what is on their mind as they perform a certain task (Jääskeläinen, 2010; Tirkkonen-Condit, 1990). Studies that employ the think-aloud protocol provide "…rich verbal data about reasoning during a problem-solving task." (Fonteyn, Kuipers, & Grobe, 1993, p. 430). Furthermore, analysis of transcripts of the think-aloud protocol can reveal how information was structured or how it was used to solve the problem.

During the study, the researcher may also ask probing questions asking the participants why they performed certain steps, what the design rationale was or to explain their thought process as they construct the models. Researcher notes made during each case study also served as a source of information for this qualitative phase. Audio and video recordings were made of the participants as they complete the modeling tasks to have a record of the modeling process as it happens and their rationale or justifications for their design choices while creating the models. The UML models, researcher notes, audio recordings & transcripts, and video recordings were collected for all six participants thus providing multiple data sources for each unit of analysis.

4.5.4 Data Coding and Analysis

Qualitative data analysis followed the collection of the quantitative data. In accordance with considerations of the Institutional Review Board (IRB) pertaining to maintaining confidentiality and anonymity, all participants names were replaced with a pseudonym. The design artefacts – the functional, structural, and behavioral models - created by the participants were first scored using the rubrics given in Appendix C - Use Case Narrative Rubric, Appendix D – Activity Diagram Rubric, Appendix E – Class Diagram Rubric, and Appendix F – Sequence Diagram Rubric. Following the scoring, the participants were categorized based on their overall proficiency. This provided context for how participants of differing proficiency employed abstract thinking and systems thinking while creating software models. The audio recordings of the think-aloud protocol were transcribed.

Thematic analysis was used identify themes present in the transcriptions and themes were coded using deductive coding scheme. Thematic analysis provided insights into patterns in a data set by focusing on its meaning (Braun & Clark, 2012). Qualitative data analysis is a recursive process that involves noticing concepts of importance and breaking down the data into distinct ideas or themes while describing properties of each code alongside representative examples (Kendall, 1999; Khandkar, 2009). The data was analyzed using NVivo version 12. NVivo allowed for effective organization and coding of qualitative data. The annotations function present in NVivo allowed for the creation of reflective and analytical memos pertaining to each transcript, video, or design artefact. The video recordings were not expressly coded but were reviewed for the purposes of verifying student actions. The videos also provided additional information and context to the corresponding transcripts.

A prominent approach for performing thematic analysis is the six-phase model proposed by Braun and Clark (2006). For the purposes of this study, the six-phase model was adapted to work with a theoretically driven inductive approach towards coding and analysis. This hybrid approach towards analysis involves integrating both data-driven and theory-driven approaches to coding (Fereday & Muir-Cochrane, 2006). Analysis included the broad *a priori* themes (Castleberry & Nolen, 2018), i.e., themes identified in advance that are relevant to the research area, as well emergent themes that arose through the coding process. First, the researcher familiarized himself with the data by performing multiple readings of the transcripts and making notes of initial ideas. Following this, interesting features from the data were captured and organized systematically to produce codes. The researcher worked through the entire data set coding extracts for all relevant themes and patterns pertaining to abstract thinking or systems thinking. For the purposes of this study, the researcher performed coding and analysis based on instances of: i) abstract thinking - episodes where students vocalized whether a detail should be included in the models or not; and ii) systems thinking - episodes where students made connections between different design elements which includes making connections between elements drawn in one diagram to elements drawn in another diagram or students making changes in one diagram based on elements drawn in another diagram. The researcher coded for instances of abstract thinking or systems thinking, as described in literature, while being open to identifying other emergent themes. Furthermore, the coder looked for specific ways in which the participants employed abstraction or systems thinking while constructing their models. The next step involved collating the different codes under the appropriate themes of abstract thinking or systems thinking. Other emergent themes were classified under the theme “miscellaneous”. Following this, the themes were reviewed to ensure that the data within each theme was coherent and identifiably distinct from other themes. The process of reviewing themes was done at two levels – i) at the level of coded extracts to ensure that all instances fit the theme and moving them to other themes if necessary; ii) at the level of the entire data set where the themes were checked for whether it accurately reflects the data set as a whole. This phase also involved some amount of re-coding to ensure accuracy. Re-coding continued until the refinements added nothing substantial to the themes. After this, a detailed analysis was written for each theme to detail its internal consistency and to delineate the lack of overlap compared to the other themes. Finally, a detailed report of results was produced providing the narrative of the data alongside supporting examples. A chronological visualization of the codes and themes will also be provided.

4.5.5 Trustworthiness Considerations

Trustworthiness or credibility in qualitative research is a nuanced topic with different approaches being taken for presenting qualitative data and analyses (Chi, 1997; Cutcliffe & McKenna, 1999; Hammer & Berland, 2014; Miles & Huberman, 1994). Establishing trustworthiness has the effect of increasing reader confidence in the findings (Curtin & Fossey, 2007). One of the approaches towards establishing trustworthiness in qualitative research is for the researcher to be completely transparent when describing the research strategies that were employed (Krefting, 1991). To this end, the specific steps taken as part of the qualitative data collection, coding, and analysis were discussed in detail. A detailed description of the context of the qualitative strand was also provided. Coding errors were minimized by reading through the transcripts several times, which also ensured that codes were correctly organized.

In addition, the primary research received peer feedback to ensure that coding was done consistently throughout. This approach of peer examination involves the research process and results being discussed with impartial peers that provoke the researcher into a deeper level of reflexivity (Krefting, 1991).

4.6 Integration of Qualitative and Quantitative Strands

The quantitative strand allowed the evaluation of student proficiency in modeling information systems in terms of functions, structures, and behaviors. This strand focused on the outcomes, i.e., the UML models that were constructed by the students. This yielded distinct patterns of software modeling proficiency and provided insights into characteristics or profiles of students. In addition, the relationships between software modeling proficiency and the constructs of abstract thinking and systems thinking were also explored. While the quantitative strand was effective in terms of evaluating outcomes, it did not yield any information about the processes or procedures employed by the students while modeling information systems in terms of functions, structures, and behaviors. The next logical step was to explore the differences in processes adopted by the high-performing students and moderate-performing students. These patterns of proficiency were explored further in the qualitative strand with a focus on how participants applied abstract thinking and systems thinking while modeling an information system in terms of functions, structures, and behaviors. The think-aloud protocol was utilized to gain insights into the rationale behind the different design decisions made by participants. The rubrics developed as part of the quantitative strand were used to score the models developed by the participants. These scores served to contextualize the findings from the qualitative strand. The results from both strands – quantitative and qualitative - were both taken into consideration to provide a holistic understanding of overall software modeling proficiency. Furthermore, considering both quantitative and qualitative data sheds light on how students of varying degrees of proficiency in software modeling employed abstract thinking and systems thinking while designing their solutions.

CHAPTER 5. MODELING PERFORMANCE

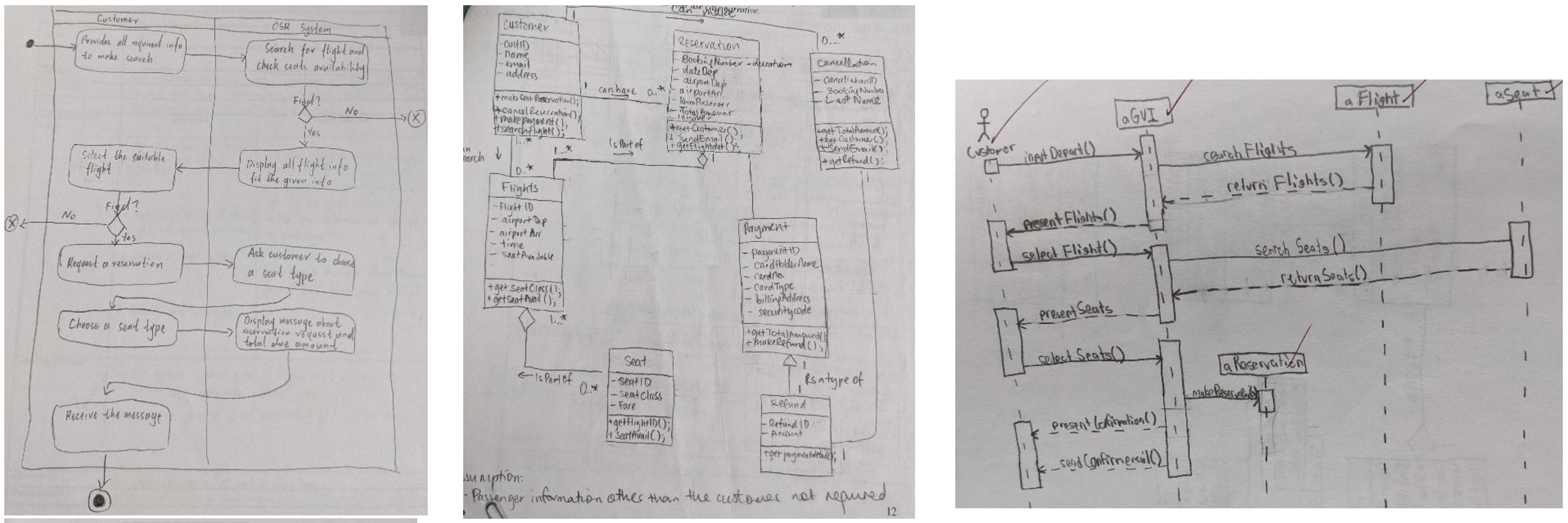
5.1 Overview of Data Analysis Procedures

To facilitate the analysis of the exam responses, rubrics detailed in Appendix B – Use Case Narrative Rubric, Appendix D – Activity Diagram Rubric, Appendix E – Class Diagram Rubric, and Appendix F – Sequence Diagram Rubric were developed and validated. The exam responses of the students were scored using these rubrics and the results were presented using bar charts. The bar charts yielded insights into patterns of proficiency among students as they modeled information systems in terms of functions, structures, and behaviors. The results presented in the bar charts addressed the first two research questions RQ1 - To what extent to did students demonstrate proficiency in abstract thinking while analyzing software systems in terms of functions, structures, and behaviors? and RQ2 - To what extent to did students demonstrate proficiency in systems thinking while analyzing software systems in functional, structural, and behavioral representations? Following this, the silhouette technique was applied to identify the optimal number of clusters in the data set. Clustering analysis was then performed using the n-Tarp binary clustering algorithm. A correlational analysis was performed on the emerging clusters to identify the nature of relationships between the different rubric elements. Logistic regression was then applied to determine how unit increases in rubric elements pertaining to abstract thinking and systems thinking affected the odds of a student being classified as a high-performing student compared to a moderate-performing student. The goodness-of-fit of the proposed regression model was tested using the Hosmer and Lemeshow test. The reduced regression model was then computed using a backward step-wise approach to eliminate predictor variables. These results addressed RQ3 – What were the characteristics or profiles of students in terms of abstract thinking and systems thinking?

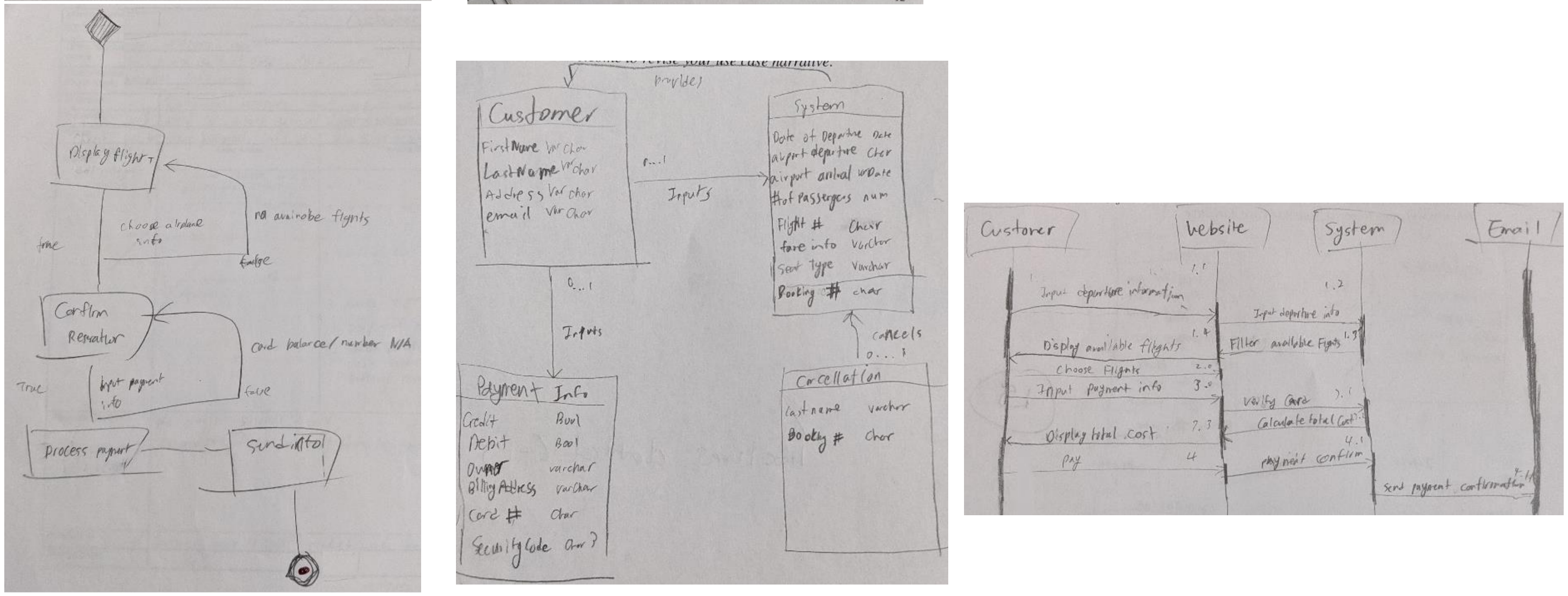
5.2 Data Scoring Approach

Exam responses from the 97 students were scored using detailed in Appendix B – Use Case Narrative Rubric, Appendix D – Activity Diagram Rubric, Appendix E – Class Diagram Rubric, and Appendix F – Sequence Diagram Rubric. Figure 8 provides illustrative examples of functional, structural, and behavioral models constructed by students that were scored as proficient as well as illustrative examples for models that were emerging or deficient due to lacking requisite details.

Figure 5.1a contains examples of proficient functional, structural, and behavioral models. The functional model given by the activity diagram conforms to UML specifications with one exactly one start and stop node drawn. There are swimlanes corresponding to the system and each actor. Only the appropriate details relevant to the reservation functionality has been captured. The structural model given by the class diagram incorporated all relevant classes of the information system. Each class had attributes and behaviors identified. Relevant relationships between classes were identified and labeled appropriately. The behavioral model given by the sequence diagram incorporated relevant classes that aligned with classes specified in the class diagram. All relevant messages and responses were drawn, and they aligned with the overall flow of control specified in the activity diagram. The sequence diagram also conformed to UML standards in terms of visual representations of messages, responses, lifelines, and execution occurrences.



a



b

Figure 5.1: Model examples

Figure 5.1b contains examples of functional, structural, and behavioral models that were not scored as proficient. The functional model given by the activity diagram used incorrect symbols for the start node and final node. The activity diagram did not have any swimlanes corresponding to the system or actor. The model did not contain any decision nodes, omitted relevant details, and did not model the overall flow of control in the system.

The structural model given by the class diagram in Figure 5.1b did not include all relevant objects of the information system. As a result of this, several important relationships were also not modeled. Some of the relationships were incorrectly modeled. While the classes included attributes, they classes did not include any behaviors.

The behavioral model given by the sequence diagram in Figure 5.1b incorrectly identified several objects and the objects did not align with the class diagram. The lifelines and responses did not conform to UML standards – dotted lines were not used. The sequence diagram was missing some key messages and corresponding responses. The overall flow may or may not align with the flow of control specified in the activity diagram.

The student scores for the functional, structural, and behavioral models given by the activity diagrams, class diagrams, and sequence diagrams were used to perform the quantitative analysis.

5.3 Modeling Functions

Figure 5.2 presented the performance distribution of students while capturing functions using use case narratives.

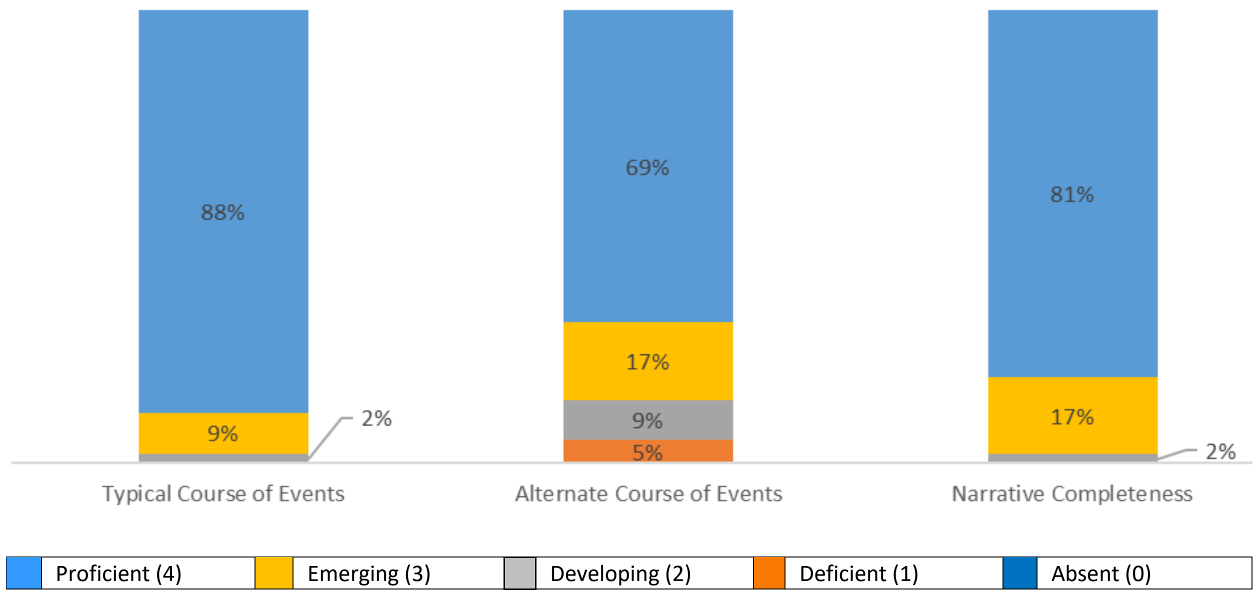


Figure 5.2: Descriptive Statistics - Use Case Narratives

Majority of students performed proficiently in capturing the typical course of events followed by the system in the given case (Appendix A – Modeling Exam). In contrast, only 69% of students captured the alternate courses of events proficiently. The alternate courses of events refer to the steps taken by the system in response to whether certain were satisfied or not. At a minimum, all learners made an attempt at completing the different elements present in the narrative template.

The performance distribution of students while capturing functions using activity diagrams was presented in Figure 5.3**Error! Reference source not found.**.

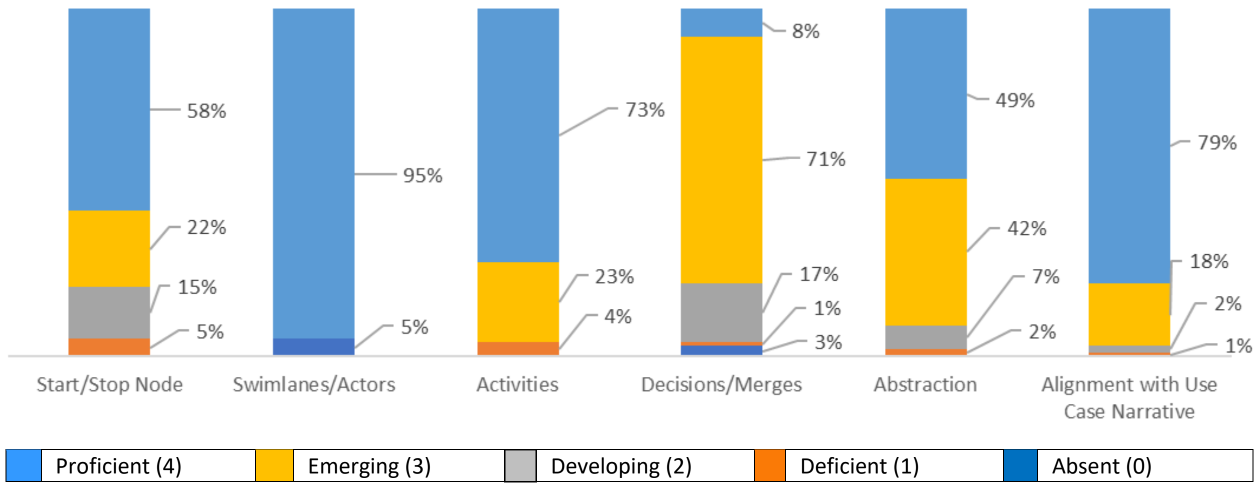


Figure 5.3: Descriptive Statistics - Activity Diagram

Overall, students were proficient in capturing functions of the information system in the given case using activity diagrams with the exception of modeling decision and merge nodes where only 8% of students utilized decision nodes appropriately and connected diverging flows of control using merge nodes accurately. Conformity to UML standards was also evaluated as part of this study. Only 5% of students failed to include swimlanes corresponding to actors in their diagrams and 58% of the students correctly drew exactly one start and stop node outside of the swimlanes.

In terms of systems thinking, 79% of students proficiently aligned their activity diagrams with the previously constructed use case narrative. Overall, majority of students were proficient in the application of systems thinking by mapping their activities and decisions or merges to the typical and alternate courses of events that they detailed in their narratives.

5.4 Modeling Structures

The performance distribution of students while capturing structures using class diagrams was presented in Figure 5.4.

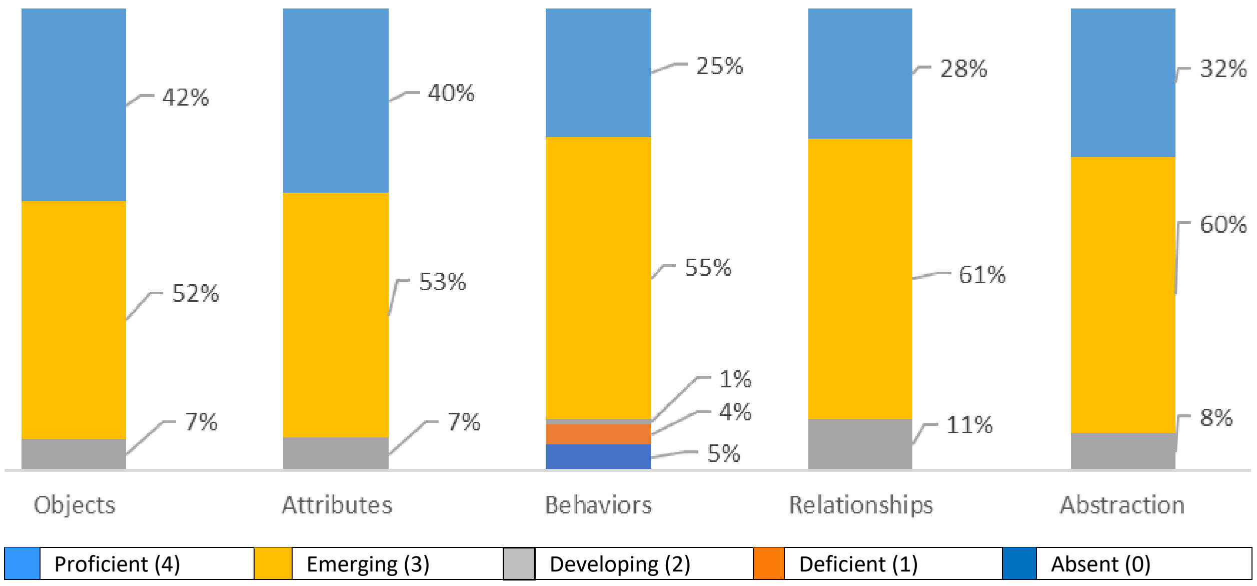


Figure 5.4: Descriptive Statistics – Class Diagrams

Overall, majority of students were at the performance level of emerging in terms of modeling information systems in terms of structures. Furthermore, the results also suggest that learners did not apply abstraction proficiently. This can be, in part, attributed to students incorporating irrelevant objects that were not essential to the prescribed functionality or the system in general. Only 28% of students were able to draw all appropriate relationships between objects. Only a quarter of all students proficiently identified all relevant behaviors for the functionality prescribed in the case while 5% of students identified no behaviors.

5.5 Modeling Behaviors

The performance distribution of students while capturing behaviors using sequence diagrams was presented in Figure 5.5.

.

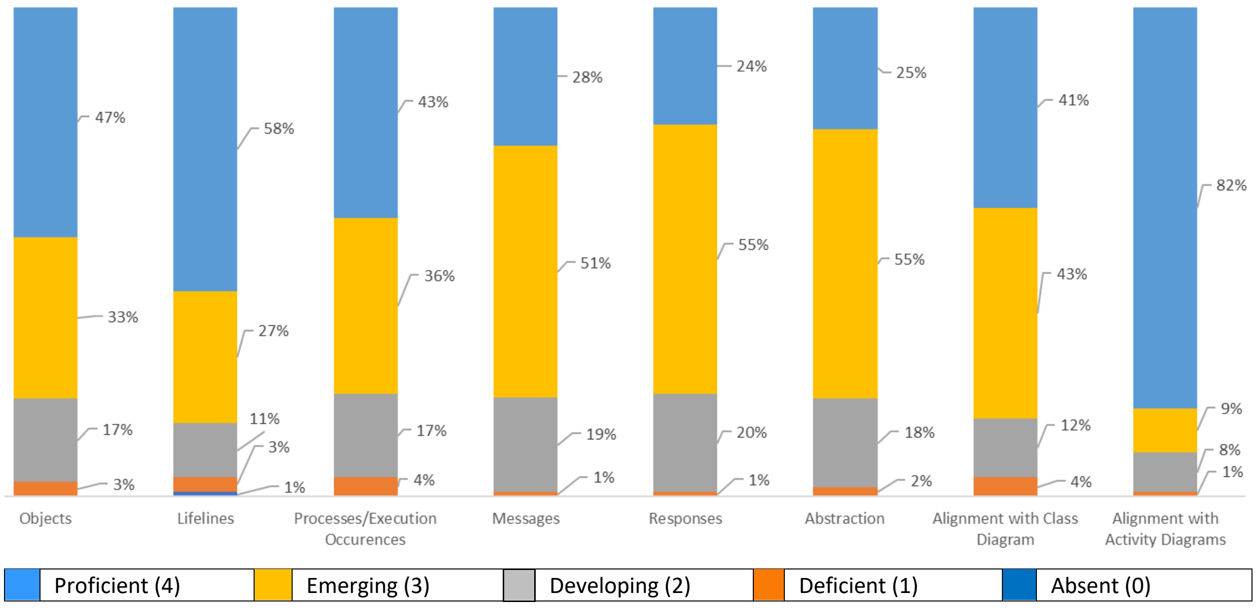
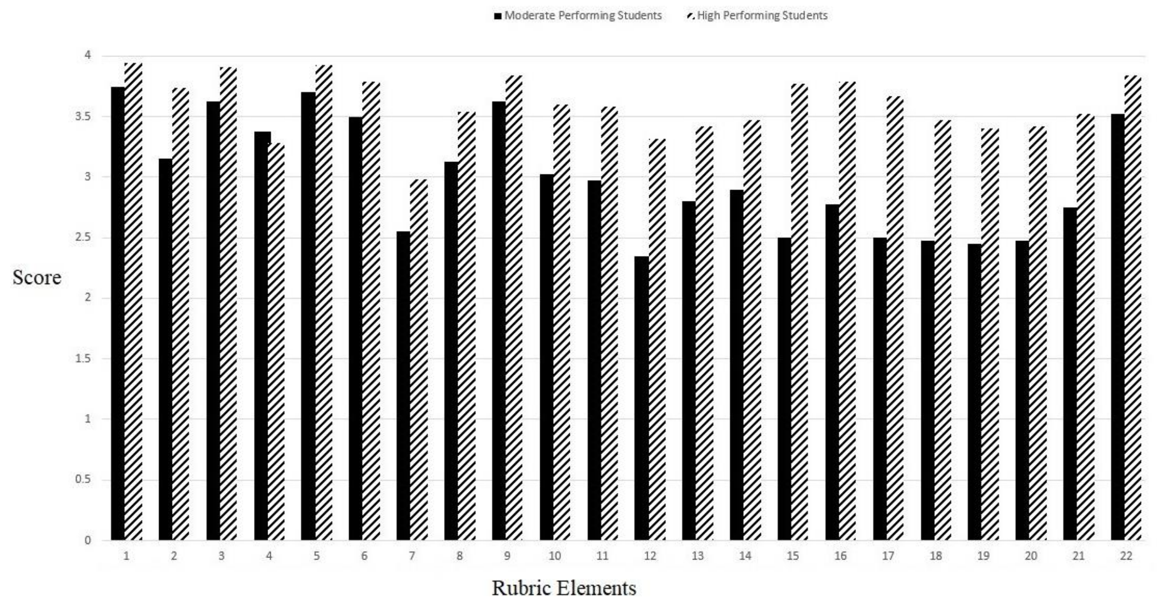


Figure 5.5: Descriptive Statistics – Sequence Diagram

Only 47% of students identified the relevant objects for the functionality prescribed in the case. This had further implications in their diagrams where corresponding lifelines, and messages or responses associated with those objects were not included. In terms of abstract thinking ability, only 25% of all students included all relevant details with minimal irrelevant details. In terms of systems thinking ability, there two distinct aspects to consider in the context of modeling information systems in terms of behaviors. 82% of students were proficient in aligning their sequence diagrams with their activity diagrams – the messages and responses in the sequence diagrams were mapped to actions in the activity diagram. However, only 41% of all students proficiently aligned their sequence diagrams with their class diagrams. Proficient alignment in this case is characterized by most objects and messages in the sequence diagram being mapped to corresponding objects and behaviors in the class diagram. It must be noted that several students featured objects in their sequence diagrams that were not present in their class diagram. Similarly, several students also drew messages in their sequence diagrams that did not have any corresponding behaviors in their class diagrams.

5.6 Cluster analysis results

The silhouette technique detailed in Section 4.4.8 determined that the optimal number of clusters for the given data was two. The clustering algorithm took into consideration the scores for all rubric elements to compute the two clusters The n-Tarp clustering algorithm (Yellamraju & Boutin, 2018) identified two clusters that were referred to as: i) moderate performing students (n=40); and ii) high performing students (n=57). The first cluster was referred to as moderate performing students due to total scores of students ranging from 20 to 44 out of a maximum of 60. The second cluster was referred to as high performing students due to total scores ranging from 39 to 52 out of a maximum of 60. The result of the n-Tarp clustering algorithm is visually represented using a bar chart in Figure 5.6.



1-Typical Course of Events, 2-Alternate Course of Events, 3-Narrative Completeness, 4-Start/Stop Nodes, 5-Swimlanes/Actors, 6-Activities, 7-Decisions/Merges, 8-Activity Diagram Abstraction, 9-Alignment with Use Case Narrative, 10-Objects, 11-Attributes, 12-Behaviors, 13-Relationships, 14-Class Diagram Abstraction, 15-Objects, 16-Lifelines, 17-Process/Execution Ocurrences, 18-Messages, 19-Responses, 20-Sequence Diagram Abstraction, 21-Alignment with Class Diagram, 22-Alignment with Activity Diagram

Figure 5.6: Cluster analysis results

Each bar represents the average score for a specific rubric element in a cluster. Overall, the cluster of high performing students had a higher average score across all rubric elements except that of start/stop nodes in the activity diagrams. Notably, students belonging to the high performing clusters exhibited higher scores in abstract thinking and system thinking in terms of functions, structures, and behaviors. Once the data points in each cluster was identified, the mean and standard deviation were computed for each cluster for the rubric elements corresponding to abstract thinking and systems thinking. Furthermore, to test for statistically significant differences between the two clusters, a t-test was conducted at a 95% confidence interval. The results of the t-test are presented in Table 5.1.

Table 5.1: Clustering t-Test results – Abstract thinking and Systems thinking

	Cluster 1 (n=40)		Cluster 2 (n=57)		
Rubric element	Mean	SD	Mean	SD	p
Activity Diagram Abstraction	3.125	.7574	3.5439	.6288	.0038
Alignment with Use-Case Narrative	3.625	.7048	3.8421	.3679	.051
Class Diagram Abstraction	2.900	.5454	3.4737	.5037	<0.00001
Sequence Diagram Abstraction	2.475	.5986	3.4211	.4981	<0.00001
Alignment with Class Diagram	2.750	.9268	3.5263	.5380	<0.00001
Alignment with Activity Diagram	3.525	.8469	3.8421	.4547	.0192

The t-test revealed statistically significant differences in abstract thinking in terms of functions, structures, and behaviors between the two clusters. The largest disparity in scores between the two clusters was for abstract thinking in terms of behaviors given by sequence diagrams. Statistically significant differences were also found in systems thinking in terms of structures and behaviors between the two clusters, whereas the proficiency of systems thinking in terms of functions between the two clusters were comparable.

5.6.1 Relationships between modeling performance of functions, structures, and behaviors

A correlational table was computed for the entirety of the data collected. The details of data scoring were discussed in Section 4.4.5. Activity diagram total was computed by adding the scores of the individual rubric elements except for abstract thinking and alignment. Similarly, class diagram and sequence diagram totals were computed by adding the scores of the respective rubric elements except for abstract thinking and alignment. The analysis was focused on selected rubric elements pertaining to abstract thinking, systems thinking, and software modeling proficiency. The results of the analysis were presented in terms of spearman coefficient. The results of the correlational analyses for the entire dataset were presented in Table 5.2.

Table 5.2: Spearman coefficient – Entire dataset

	1	2	3	4	5	6	7	8	9	10
1	-									
2	.62**	-								
3	.39**	.44**	-							
4	.45**	.51**	.44**	-						
5	.21*	.32**	.22*	.23*	-					
6	.36**	.27**	.29**	.30**	.79**	-				
7	.25*	.22*	.14	.18	.54**	.55**	-			
8	.08	.12	.13	.18	.32**	.33**	.63**	-		
9	.07	.09	.19	.22*	.21*	.18	.27**	.25*	-	
10	.26**	.17	.21*	.19	.44**	.47**	.84**	.61**	.17	-

Note. *p<0.05. **p<.001
1 – Use-Case Narrative Total, 2 – Activity Diagram Abstraction, 3 – Alignment with Use-Case Narrative, 4 – Activity Diagram Total, 5 – Class Diagram Abstraction, 6 – Class Diagram Total, 7 – Sequence Diagram Abstraction, 8 – Sequence Diagram Alignment with Class Diagram, 9 – Sequence Diagram Alignment with Activity Diagram, 10 – Sequence Diagram Total

Statistically significant correlations were found between use case narrative total, functional abstract thinking ability (given by the rubric element activity diagram abstraction), functional systems thinking ability (given by the rubric element alignment with the use-case narrative), and proficiency of functional modeling (given by the rubric element activity diagram totals). It can be inferred from the results that students exhibiting greater proficiency of abstract thinking and system thinking were also, overall, more proficient with functional modeling. The proficiency of functional modeling was evidenced by constructing activity diagrams that included more relevant details and exhibited better alignment with the use case narratives.

A statistically significant correlation was also found between structural abstract thinking (given by the rubric element class diagram abstraction) and the overall structural modeling proficiency (given by the rubric element class diagram total). It can be inferred from the results that students exhibiting greater proficiency of abstract thinking were also, overall, more proficient with structural modeling (given by the rubric element – class diagram total). The proficiency of structural modeling was evidenced by constructing class diagrams with more relevant details included.

Statistically significant correlations were found between behavioral abstract thinking (given by the rubric element sequence diagram abstraction), alignment with class diagrams, and overall behavioral modeling proficiency (given by sequence diagram total). It can be inferred from these results that students exhibiting greater proficiency of abstract thinking and systems thinking were also, overall, more proficient with behavioral modeling. The proficiency of behavioral modeling was evidenced by constructing sequence diagrams that included more relevant details and were better aligned with the class diagrams.

Given the differing performance patterns observed in the two clusters, separate correlational analyses were then performed for each cluster to investigate whether there were any differences in these relationships in the two clusters. The results of the correlational analyses for the moderate performing students and high-performing students were presented in Table 5.3 and Table 5.4, respectively.

Table 5.3: Spearman coefficient – Moderate performing students

	1	2	3	4	5	6	7	8	9	10
1	-									
2	.67**	-								
3	.55**	.45**	-							
4	.58**	.58**	.51**	-						
5	.18	.35*	.23	.23	-					
6	.35*	.32*	.41**	.46**	.76**	-				
7	.004	.06	.25	.32*	.05	.10	-			
8	-.17	-.12	.11	.09	-.35*	-.22	.62**	-		
9	.13	.12	.29	.32*	-.01	-.09	.41**	.16	-	
10	.02	-.07	.29	.18	-.13	-.03	.80**	.59**	.26	-

Note. *p<0.05. **p<.001
1 – Use-Case Narrative Total, 2 – Activity Diagram Abstraction, 3 – Alignment with Use-Case Narrative, 4 – Activity Diagram Total, 5 – Class Diagram Abstraction, 6 – Class Diagram Total, 7 – Sequence Diagram Abstraction, 8 – Sequence Diagram Alignment with Class Diagram, 9 – Sequence Diagram Alignment with Activity Diagram, 10 – Sequence Diagram Total

Table 5.4: Spearman coefficient – High performing students

	1	2	3	4	5	6	7	8	9	10
1	-									
2	.48**	-								
3	.17	.41**	-							
4	.34**	.45**	.35**	-						
5	-.14	.12	.12	.17	-					
6	.04	.02	.14	.19	.67**	-				
7	-.12	.05	-.12	-.03	.47**	.39**	-			
8	-.14	.06	-.02	.14	.52**	.34**	.43**	-		
9	-.18	-.04	-.003	.07	.25	.21	-.02	0.19	-	
10	-.17	-.08	.003	.14	.33*	.13	.66**	.48**	-.09	-

Note. *p<0.05. **p<.001
1 – Use-Case Narrative Total, 2 – Activity Diagram Abstraction, 3 – Alignment with Use-Case Narrative, 4 – Activity Diagram Total, 5 – Class Diagram Abstraction, 6 – Class Diagram Total, 7 – Sequence Diagram Abstraction, 8 – Sequence Diagram Alignment with Class Diagram, 9 – Sequence Diagram Alignment with Activity Diagram, 10 – Sequence Diagram Total

Across both moderate performing students and high performing students, statistically significant correlations were found between use case narrative total, functional abstract thinking ability (given by the rubric element activity diagram abstraction), functional systems thinking ability (given by the rubric element alignment with the use-case narrative), and proficiency of functional modeling (given by the rubric element activity diagram totals). It can be inferred from the results that students exhibiting greater proficiency of abstract thinking and system thinking were also, overall, more proficient with functional modeling. The proficiency of functional modeling was evidenced by constructing activity diagrams that included more relevant details and exhibited better alignment with the use case narratives.

In both clusters, statistically significant correlations were found between structural abstract thinking (given by the rubric element class diagram abstraction) and the overall structural modeling proficiency (given by the rubric element class diagram total). It can be inferred from the results that students exhibiting greater proficiency of abstract thinking were also, overall, more proficient with structural modeling (given by the rubric element – class diagram total). The proficiency of structural modeling was evidenced by constructing class diagrams with more relevant details included.

Similarly, across both moderate performing students and high performing students, statistically significant correlations were found between behavioral abstract thinking (given by the rubric element sequence diagram abstraction), alignment with class diagrams, and overall behavioral modeling proficiency (given by sequence diagram total). It can be inferred from these results that students exhibiting greater proficiency of abstract thinking and systems thinking were also, overall, more proficient with behavioral modeling. The proficiency of behavioral modeling was evidenced by constructing sequence diagrams that included more relevant details and were better aligned with the class diagrams. Notably, there were no statistically significant correlations in either cluster between overall behavioral modeling proficiency and the alignment of sequence diagrams with activity diagrams. Also, in the cluster of high performing students, there was no statistically significant correlation between behavioral abstract thinking and alignment of sequence diagrams with activity diagrams. There was a statistically significant correlation present between these two elements in the cluster of moderate performing students. Interestingly, in both clusters, statistically significant correlations were found between functional abstract thinking (given by the rubric element activity diagram abstraction) and functional systems thinking (given by the rubric element alignment with use case narrative). It could be inferred from these results that those students who were more proficient with abstract thinking also tend to be more proficient with systems thinking. Overall, the relationships between the various rubric elements remained broadly consistent across the two clusters.

5.7 Logistic regression results

Logistic regression was used to compute the probability of student being classified as high performing or moderate performing based on unit changes in their abstract thinking and systems thinking ability in terms of functions, structures, and behaviors. The Hosmer and Lemeshow test was used to test whether the proposed regression model was a good fit for the given data. The null hypothesis for this test was that the regression model is correctly specified and fits the data well. A statistically significant result would lead to the null hypothesis being rejected and this result would indicate that the proposed model is not a good fit for the given data. The test was run in steps where all independent variables were present in the first step and a variable was removed at each step. The results of the Hosmer and Lemeshow test are presented in Table 5.5.

Table 5.5: Hosmer and Lemeshow Test Results

Step	Chi-square	Df	p
1	4.333	8	0.826
2	4.232	8	0.836
3	3.734	6	0.713
4	5.685	6	0.459
5	3.277	3	0.351
6	5.857	1	0.016

The Hosmer and Lemeshow test did not return significant results until there was only a single independent variable left in the model. This result indicates that a reduced model containing only two independent variables (step five) was a good fit for the given data. This also implies that not every component of abstract thinking and systems thinking played a substantial role in whether a student was classified as high performing or moderate performing.

Given the results of the Hosmer and Lemeshow test, logistic regression was computed for a reduced model containing only two independent variables. SPSS utilized a backward step-wise approach to eliminate predictor variables at each step. The results of the backward step-wise process was presented in Appendix – H. The process began with the following rubric elements included in the model: activity diagram abstraction, alignment with use case narrative, class diagram abstraction, sequence diagram abstraction, sequence diagram alignment with class diagram, and sequence diagram alignment with activity diagram. At each step, an independent variable was removed, and the regression was computed. The results of the logistic regression corresponding to step five of the Hosmer and Lemeshow test were presented in Table 5.6.

Table 5.6: Logistic regression results – Reduced Model

Variables in the Equation	B	S.E.	Wald	Df	Sig.	Exp(B)
Sequence Diagram Abstraction	0.383	0.319	1.443	1	0.23	1.467
Sequence Diagram Alignment with Activity Diagram	0.66	0.36	3.35	1	0.067	1.934
Constant	-3.252	1.505	4.67	1	0.031	0.039

The final reduced regression model was presented below in (2):

$$\log\left(\frac{prob_{high}}{prob_{mod}}\right) = -3.252 + 0.383 * Sequence\ Diagram\ Abstraction + 0.66 * Alignment\ of\ Sequence\ Diagram\ with\ Activity\ Diagram \quad (2)$$

The following conclusions can be inferred from the logistic regression model:

i) A unit increase in alignment of sequence diagram with activity diagrams while sequence diagram abstraction is held constant, increases the odd of high performance by a factor of 1.934.

ii) A unit increase in sequence diagram abstraction while alignment of sequence diagram with activity diagrams is held constant, increases the odds of high performance by a factor of 1.467.

5.8 Summary – Modeling Performance

The first research question investigated as part of this analysis was - To what extent to did students demonstrate proficiency in abstract thinking while analyzing software systems in terms of functions, structures, and behaviors? Overall, students were proficient in modeling the given information system in terms of functions using use case narratives and UML activity diagrams. Students were able to proficiently employ abstract thinking to incorporate relevant details in their solution models and capture the interactions between the user and information system. However, it must be noted that students were in general less proficient in identifying the alternate course of events using use case narratives. In contrast, the overall performance of students dropped to the level of emerging when modeling the given information system in terms of structures and behaviors using UML class diagrams and UML sequence diagrams respectively. This was accompanied by a drop in abstract thinking scores as well. Students often failed to include important details in both diagrams. Class diagrams were missing key objects and their associated attributes, behaviors, and relationships. Sequence diagrams were missing key objects and their associated messages and responses.

The second research question investigated was - To what extent to did students demonstrate proficiency in systems thinking while analyzing software systems in terms of functions, structures, and behaviors? Overall, students were proficient in applying systems thinking while modeling information systems in terms of functions. Most students were able to accurately map actions in the activity diagrams to individual steps in the use case narratives. Similarly, most students were able to map the alternate course of events from their use case narratives to decisions and merges in their activity diagrams. While constructing sequence diagrams, students were proficient with aligning their sequence diagrams with their activity diagrams. Students were able align the overall flow of control within the system based on their activity diagrams even if some students had made errors in drawing messages being sent from and to incorrect objects. In comparison, students were substantially less proficient in aligning their sequence diagrams with their class diagrams. Only 41% of learners had included objects in their sequence diagrams that were present in their class diagram and mapped messages in their sequence diagrams to behaviors in their class diagrams. Many learners featured objects in their sequence diagram that did not exist in the class diagram and messages that did not map to any specific behavior.

The third research question that was investigated was - What were the characteristics or profiles of students in terms of abstract thinking and systems thinking as evidenced by their system representations? The silhouette technique was applied to the dataset and it indicated that the optimal number of clusters was two. The n-Tarp binary clustering algorithm revealed the existence of two distinct clusters – moderate-performing students (n=40) and high-performing students (n=57). High performing students demonstrated significantly higher abstract thinking ability in terms of functions, structures, and behaviors compared to the students compared to the moderate performing students. High performing students were also significantly more proficient in systems thinking in terms of structures and behaviors. However, the two clusters exhibited comparable systems thinking ability in terms of functions. It must be noted that the differences between these clusters in terms of systems thinking given by alignment of the sequence diagram with the class diagram and alignment of the sequence diagram with the activity diagram were also statistically significant. The results of the correlational analysis indicated the following relationships:

i) Proficiency of abstract thinking and system thinking were positively associated with overall functional modeling proficiency.
ii) Proficiency of abstract thinking was positively associated with overall structural modeling proficiency.

iii) Proficiency of abstract thinking and systems thinking were positively associated with overall behavioral modeling proficiency.

Finally, logistic regression was performed to identify the effect of unit-increases of rubric elements on the odds of a student being high-performing. In addition, the regression model determined the most important factors by adopting a backwards step-wise approach towards eliminating predictor variables. The Hosmer and Lemeshow test confirmed that the reduced model was a good fit for the data set. It could be confirmed from the regression model that the two most important elements that positively affect the odds of a student being high performing were the abstract thinking ability exhibited while modeling the information system in terms of behaviors, and systems thinking ability given by the alignment of the sequence diagrams with the activity diagrams.

CHAPTER 6. MODELING APPROACHES

6.1 Summary of Data Analysis Procedures

A single case study approach was employed in the qualitative strand of this study. A thematic analysis approach was employed to analyze the transcripts of the audio recordings of each participant. The researcher familiarized himself with the data by reading through each transcript multiple times while making notes of initial ideas or patterns. Following this the researcher captured and organized interesting features from the data to produce codes. The researcher was specifically looking for ways the participants employed abstract thinking and systems thinking while modeling the information system, given in the case, in terms of functions, structures, and behaviors. Following this, the codes were collated under the broad themes of abstract thinking and systems thinking. The operational definition for abstract thinking referred to episodes where students vocalized whether a detail should be included in the models. The operational definition for systems thinking referred to episodes where students made connections between different design elements which included making connections between elements drawn in one diagram to elements drawn in another diagram or students making changes in one diagram based on elements drawn in another diagram. Codes that did not fit under either theme were collated under the theme "miscellaneous". The only example of this code was that of "UML conformity" where participants explicitly vocalized content pertaining to the use of specific symbols or design choices as prescribed by the UML standards.

After this step, the themes and codes were reviewed to ensure that the data within each theme and code were coherent and identifiably distinct from other themes and codes, respectively. The two-level review process was conducted. The first was at the level of coded extracts to ensure that all instances fit the theme and moving them to other themes if necessary. This proved to be especially challenging in the context of codes pertaining to systems thinking because participants adopted proactive and reactive approaches to systems thinking. The code "mapping between models" refers to the proactive approach where participants actively referenced previously constructed models and "alignment" refers to the reactive approach where changes were made retroactively. The video corresponding to these segments were used to verify the approach taken by the participant and the code was attributed accordingly. The second level of review was done at the level of entire data set where the themes were checked for whether they accurately reflect the data set as a whole. This review process led to a degree of re-coding. Re-coding continued until no further refinements could be or there no substantial additions to the codes. The extracts did not have to fit exclusively into any single code. There were several instances where the extracts could be attributed to two or more codes. This is an expected outcome as participants would, for instance, be employing abstract thinking and systems thinking simultaneously while modeling information systems in terms of functions, structures, and behaviors.

The themes and codes, based on the thematic analysis of six transcripts, were detailed in Table 6.1. Operational definitions and sample quotes were provided for each code.

Table 6.1: Themes and Codes

Theme	Code	Definition	Sample Quote
Abstract Thinking	Aggregation	Instances of grouping together different details from the problem statement	*“I'm going to combine some of these methods for this…I'm probably going to combine the like- send information to client and then like send request bill and memo to the manager, like send information to the manager”*
	Decomposition	Instances of breaking down details in the problem statement into smaller components	*“…like here where I wrote, like type experience and qualifications all into one … I’m going to split this up again because it makes sense to have them as different data members for the actual- because they're actually checking against the database”*
	Action and response identification	Instances of identifying interactions between the system and the user to achieve a specific functionality	*“I'll just do a top decision. it terminates but then..so if it's no.. so technically this should be part of... if there's an a Staffing request this month... So it has to do with expiration and the request for you falls... Staffing request is not valid. It sends back the letter. Okay, so technically it's invalid. It goes go back to the Client, I'm using now the original. Document. so receive letter of rejection. So technically that ends the process here and just connect back to the end node.*

Table 6.1 continued

Actor identification	Instances of identifying the different users that interact with the system	*"yeah so I add a two other actors inside a system because in the placement department there are some employee checking the qualification of the the the worker the employee there's something for you inside the department checking it and they mark it as reserved in the staff database, so uhmm, there that also actors inside of system and what arrangement department they also mark… and they do some action inside a system and send it back to the contract manager so they should also be a actor instead of system itself processing"*
Object identification	Instances of identifying objects integral to the structure of the information systems	*"Now I'm trying to visualize the class diagram because I guess- I mean since I'm already here I might as well draw it from ... Just trying to figure out how to draw it which I mean obviously will be PSSM and then the Department's client would be separate, employee would be separate. Contract would go under client, and I think that that's an entity ... probably also create the request and then attach it with Bill in them."*
Relationship identification	Instances of identifying the relationships between the objects of the information system	*"So I'm sticking with general relationships except for like the requests in the contracts … or aggregation and composition is the name of that one. Okay, because these can't exist without these existing; where these all exist independently of each other, you need a- you need a client to have a contract. I have a request"*
Message and response identification	Instances of identifying interactions between the objects of the information system	*"I will have to return to staffing request database, but I can represent that towards the lower end of the line when it's needed again. again, mostly this is just not looking ahead towards where I had those two actors that kind of threw this off a little bit, but I should have like realized from activity diagram, but for some reason. I just find it much easier to follow the actual use case than the diagrams that I have … which Again part of it is also because I don't really feel confident about everything that happened after the arrangements Department … all of the different activities that happened there. anyway moving on …"*

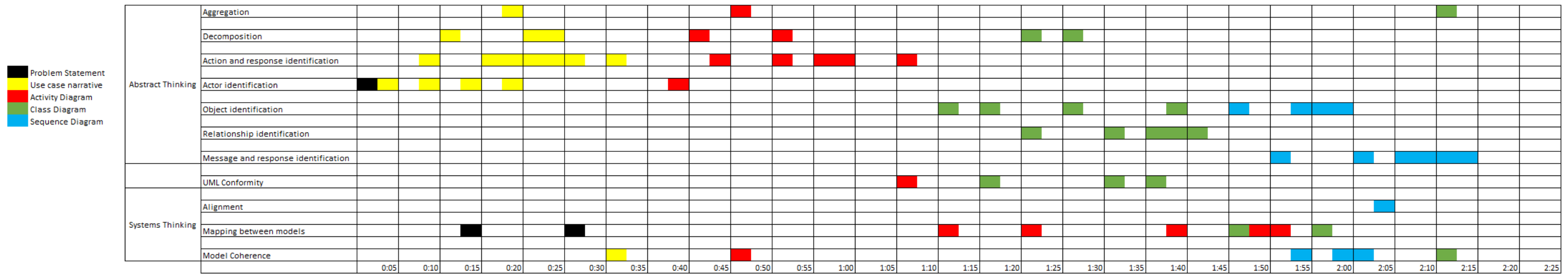
Table 6.1 continued

Miscellaneous	UML conformity	Instances where the participant identified UML standards for different the diagrams or made changes to their diagrams to conform to UML standards	*"I always forget what the symbol you put before the data member of a entity in a class diagram is …. I think its a plus or minus .. hmmm…"*
Systems Thinking	Alignment	Instances where the participant made changes to a diagram based on details observed in another diagram	*"So these should have been checking against what I erased here. So this should actually be over here."*
	Mapping between models	Instances where the participant actively referenced an already constructed diagram while constructing another diagram	*"first just to get like all the different- I think I'm probably going to go with the different entities being the same as the swim lanes. So just to get those all down- make sure I have them large enough"*
	Model Coherence	Instances where the participant made changes to a diagram in response to new details that were incorporated	*"Most of these I can translate into this fairly easily but… I'm going to need … contract database … request database … staff database … so I am going to need …. I'm going ot squeeze another one in here actually …"*

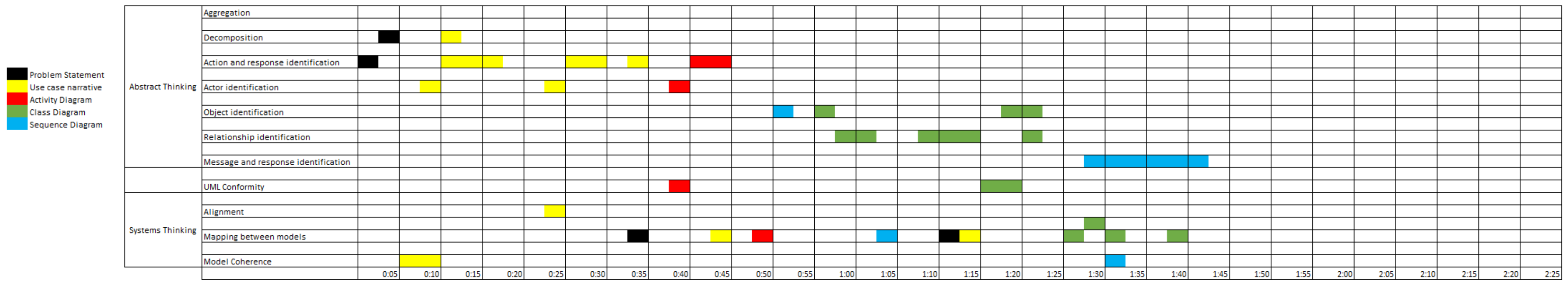
6.2 Chronological Visualization

Through the process of thematic analysis, the researcher observed that participants chose to go about modeling systems in terms of functions, structures, and behaviors in different orders. All participants modeled the system given in the case in terms of functions first using the use case narrative and activity diagram. However, following this, some participants modeled the systems in terms of behaviors first before moving onto structures while others chose to model the system in terms of structures first before moving onto behaviors. Gantt charts were used to visualize how participants employed abstract thinking and systems thinking while modeling an information system in terms of functions, structures, and behaviors. Figure 6.1 presents the codes - identified as part of thematic analysis - on a Gantt chart for participants who first modeled functions, then structures, and finally behaviors of the information system. Figure 6.2 presents the codes - identified as part of thematic analysis - on a Gantt chart for participants who first modeled functions, then behaviors, and finally structures of the information system. The figures present the codes on a timeline alongside the information of what artefact was being used for each instance of the code.

a



b



c

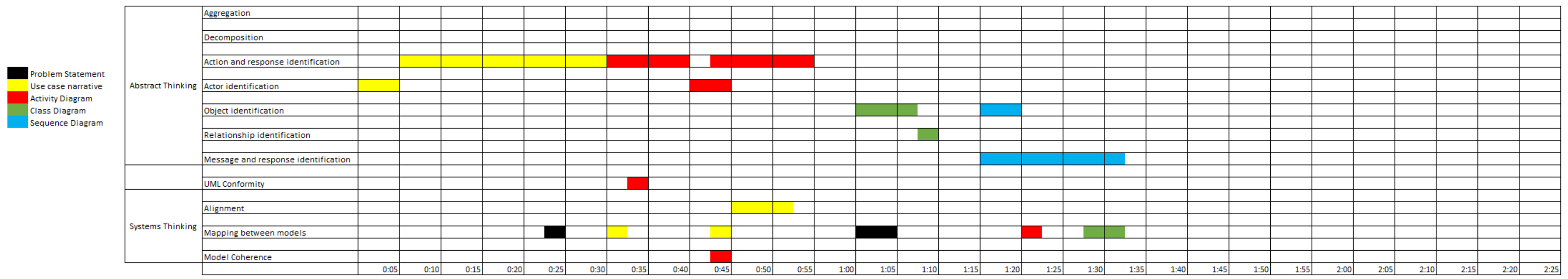


Figure 6.1: Gantt chart visualization – structures before behaviors

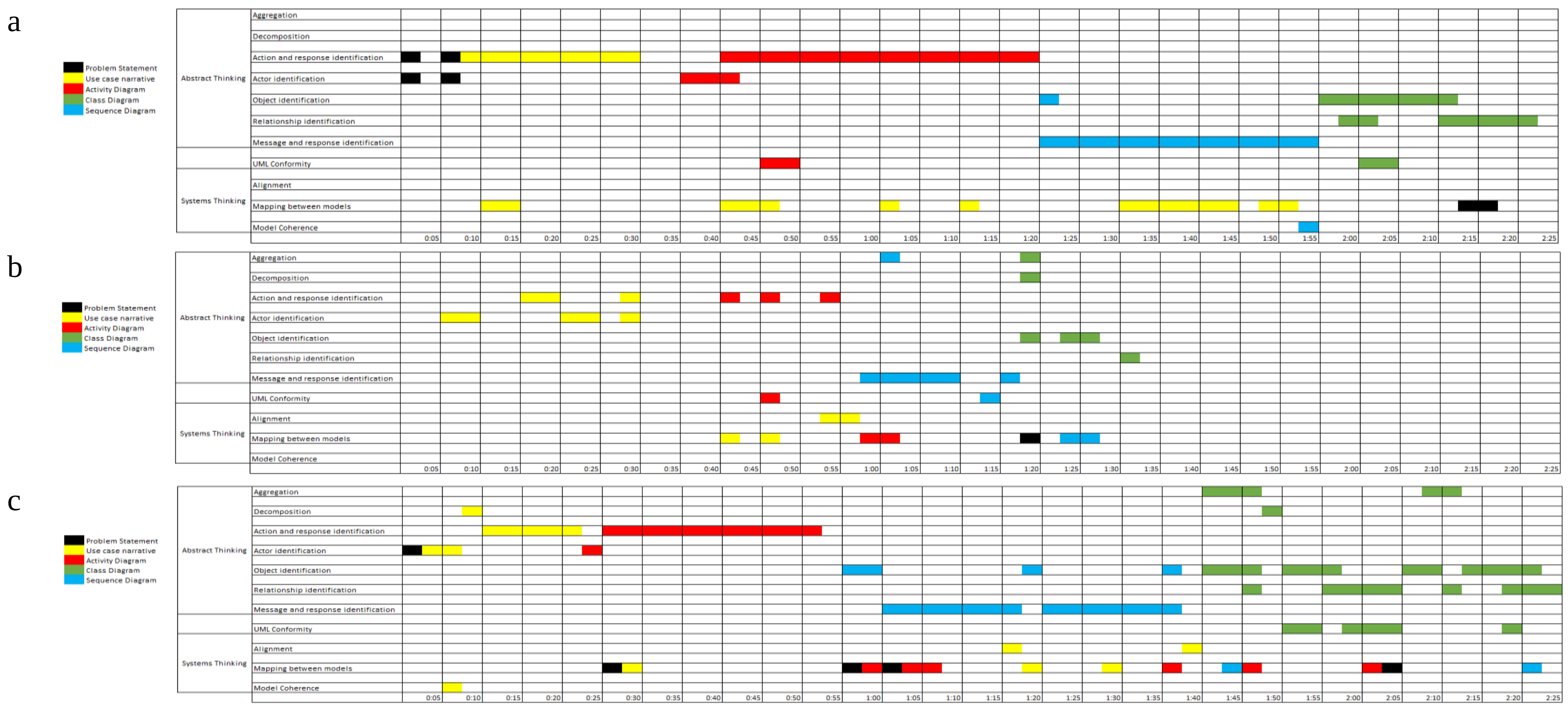


Figure 6.2: Gantt chart visualization – behaviors before structures

Figure 6.1 presents the Gantt chart of codes pertaining to the participants who constructed the structural models before the behavioral models – they were referred to as FSB1, FSB2, and FSB3. Figure 6.2 presents the Gantt chart of codes pertaining to the other three participants who constructed the behavioral models before the structural models – they were referred to as FBS1, FBS2, and FBS3. Participants took between one hour and thirty minutes to two hours and twenty-five minutes to model the information system in terms of functions, structures, and behaviors.

In both approaches, participants modeled the system in terms of functions using the use case narratives and constructing activity diagrams. Abstract thinking this phase was broadly characterized by the codes - actor identification and action and response identification. Participants would identify the different users that interact with the system and capture the flow of control between users and the system to achieve a specific functionality. Participant FSB1 took the additional step of combining certain attributes given in the case to ease the process of functional modeling. This corresponds to the code of aggregation. The participants exhibited systems thinking by proactively referencing the problem statement while completing the use case narratives and then proceeded to reference the narrative while constructing the activity diagram. It must be noted that participant FBS3 referenced the problem statement in addition to the use case narrative while constructing the activity diagram. Participants FBS2 and FSB3 also made changes to their use case narratives based on details included in their activity diagrams. This corresponds to the codes of "alignment" in Figure 6.1a and Figure 6.2b, respectively. This also represents a more reactive approach to systems thinking. At around the 45-minute mark, participant FSB1 reorganized the activity diagram to ensure that the model was still coherent after the new details that were incorporated. Table 6.2 presents the scores of the participants in terms of modeling the given the case in terms of functions for selected rubric elements. The artefacts produced by the participants i.e., the functional models – given by the use case narratives and activity diagrams were scored using Appendix C – Use Case Narrative Rubric and Appendix D – Activity Diagram Rubric, respectively. There were no substantial differences in functional modeling proficiency between the participants. The differences in overall scores for the activity diagrams could be attributed to lack of UML conformity – namely the number of start and stop nodes, the absence of swimlanes, and the absence of merge nodes to combine control flows following decisions. All participants were proficient in abstract thinking and systems thinking in terms of functions. Participants included relevant details in their functional models and aligned their activity diagrams proficiently with their use case narratives.

Table 6.2: Participant scores – Functional Modeling Proficiency

Participant	Use Case Narrative Total (12)	Activity Diagram Abstraction (4)	Alignment with Use-Case Narrative (4)	Activity Diagram Total (16)
FSB1	10	4	4	14
FSB2	12	3	4	13
FSB3	12	4	4	15
FBS1	12	4	4	14
FBS2	12	4	4	16
FBS3	11	4	4	12

Following the construction of the functional models, the participants moved on to modeling the given system in terms of structures or behaviors. Participants FSB1, FSB2, and FSB3 modeled the system in terms of structures first. Abstract thinking in this phase was broadly characterized by the codes - object identification and relationship identification. All three participants identified objects and the relationships between them. All three participants took different approaches to systems thinking. Participant FSB1 primarily referenced the activity diagram while performing object and relationship identification. It must be noted that participant FSB1 also decomposed the attributes - that was aggregated prior - into its component parts. Participant FSB1 also vocalized explicitly the correct symbols to be used to model relationships in class diagrams corresponding to code of UML conformity. Participant FSB2 identified the objects in the sequence diagram first before deciding to complete the class diagram. As such, the partially constructed sequence diagram was also referenced in addition to the problem statement and the use case narrative. Participant FSB3 solely referenced the problem statement while modeling the given system in terms of structures.

Once the structural models were completed, participants FSB1, FSB2, and FSB3 modeled the system in terms of behaviors using UML sequence diagrams. Abstract thinking in this phase was broadly characterized by the codes – object identification, and message and response identification. Participants FSB1 and FSB3 identified the relevant objects alongside the pertinent messages and responses. Participant FSB2 had already identified the objects earlier. Participant FSB1 employed aggregation to consolidate some attributes as part of messages. In terms of systems thinking, the participants all actively referenced other models while modeling the system in terms of behaviors corresponding to the code mapping between models. Participants FSB1 and FSB3 referenced the activity diagram and class diagram while constructing the sequence diagram. Participant FSB2 exclusively referenced the class diagrams. Participants FSB1 and FSB2 reworked their diagrams to ensure that the models were coherent in response to the new details incorporated.

Participants FBS1, FBS2, and FBS3 modeled the system in terms of behaviors first. Abstract thinking in this phase was broadly characterized by the codes – object identification, and message and response identification. All three participants identified the relevant objects alongside the pertinent messages and responses. However, it must be noted that participant FBS2 did not vocalize object identification. Participant FBS2 consolidated the different employee objects into a single object called staff corresponding to the code - aggregation. In terms of systems thinking, the participants all actively referenced other models while modeling the system in terms of behaviors corresponding to the code mapping between models. Participant FBS1 exclusively referenced the use case narrative while participant FBS2 exclusively referenced the activity diagram. In contrast, participant FBS3 referenced the problem statement, the use case narrative, and activity diagram while constructing the sequence diagram.

Once the behavioral models were completed, participants FBS1, FBS2, and FBS3 modeled the system in terms of structures using UML class diagrams. Abstract thinking in this phase was broadly characterized by the codes - object identification and relationship identification. All three participants identified objects and the relationships between them. Participant FBS2 vocalized the earlier aggregation of employee objects and decomposed them into the component objects – corresponding to the code decomposition. Participant FBS3 vocalized aggregation in terms of introducing super-classes to house common attributes. Participants FBS1 and FBS3 focused on conforming to UML standards in terms of attribute visibility in addition to relationship symbols and multiplicity corresponding to the code – UML conformity. In terms of systems thinking, the participants all actively referenced other models while modeling the system in terms of structures corresponding to the code mapping between models. Participant FBS1 exclusively referenced the problem statement. Participant FBS2 referenced the previously constructed sequence diagram in addition to the problem statement whereas participant FBS3 referenced the activity diagram, problem statement, and sequence diagram.

Table 6.3 presents the scores of the participants in terms of modeling the given the case in terms of structures and behaviors for selected rubric elements. The artefacts produced by the participants i.e., the structural models given by the class diagrams and the behavioral models given by the sequence diagrams were scored using Appendix E – Class Diagram Rubric, and Appendix F – Sequence Diagram Rubric, respectively.

Table 6.3: Participant scores – Structural and Behavioral Modeling Proficiency

Participant	Class Diagram Abstraction (4)	Class Diagram Total (16)	Sequence Diagram Abstraction (4)	SD - Alignment with Class Diagram (4)	SD - Alignment with Activity Diagram (4)	Sequence Diagram Total (20)
FSB1	2	9	3	2	4	14
FSB2	4	15	4	4	4	20
FSB3	3	11	3	3	4	17
FBS1	3	13	3	4	4	18
FBS2	4	16	3	3	4	15
FBS3	3	13	3	2	3	15

In general, the scores indicate a marked reduction in proficiency of abstract thinking in terms of structures and behaviors compared to the proficiency of abstract thinking in terms of functions. With the exception of participant FSB1, neither approach – modeling behaviors before structures, nor modeling structures before behaviors – yielded dramatically different results in terms of overall software modeling proficiency. Participants across both approaches displayed varying degrees of systems thinking ability in terms of aligning their sequence diagrams with class diagrams, however, participants in both approaches were generally proficient in aligning their sequence diagrams with their activity diagrams in terms of overall flow. It is also worth noting that participants FSB3 and FBS2 referenced other models the least among all participants but their proficiency of systems thinking was not negatively impacted.

Table 6.4 presents the proportion of time spent by each participant on each model. The results are presented in terms of percentage of total time spent on the problem statement, use case narratives, activity diagrams, class diagrams, and sequence diagrams. Idle time refers to periods of time where the participant did not think aloud pertaining to design decisions or the rationale for including specific details. Idle time also included breaks taken by the participants.

Table 6.4: Time spent on models by participants

Participant	Total Time (minutes)	% - Idle time	% on Problem statement	% of time on Functions UCN	AD	% of time on Structures	% of time on behaviors
FSB1	135	27	2	20	16	15	20
FSB2	103	23	5	19	10	23	19
FSB3	93	10	0	31	28	10	20
FBS1	143	5	4	16	31	24	19
FBS2	93	21	0	21	16	16	26
FBS3	145	2	2	15	20	31	29

Note. UCN – Use Case Narratives. AD – Activity Diagrams.

Overall, the participants in either approach spent comparable amounts of time on constructing each model. There also does not seem to be any direct relationship between time spent on modeling and the proficiency exhibited by the participants. Functional modeling, on the whole, takes longer because of participants constructing both use case narratives and activity diagrams to model functions. It must be noted that while participants FSB3 and FBS2 took the least amount of time to construct the functional, structural, and behavioral models of the given system there were no substantial differences in modeling proficiency between them nor were there any substantial differences compared to participants who took longer. These participants also spent no time on the problem statement exclusively and proceeded to functional modeling almost immediately.

6.3 Summary – Modeling Approaches

The research question investigated through this analysis was - How did students use abstract thinking and systems thinking when modeling software systems in terms of functions, structures, and behaviors? The theme of abstract thinking was characterized in terms of functions, structures, and behaviors. Abstract thinking in terms of functions were broadly characterized by the codes - actor identification, and action and response identification. Participants identified the users that interact with the system and modeled the nature of these interactions. Abstract thinking in terms of structures were broadly characterized by the codes – object identification and relationship identification. Participants identified the objects relevant to the solution and the relationships between them. Abstract thinking in terms of behaviors were broadly characterized by the codes – object identification and message and response mapping. Participants identified the objects relevant to the solution and modeled the messages and responses that are passed between them to achieve the functionality of the system. Systems thinking was broadly characterized in terms of the codes – alignment, mapping between models, and model coherence. While modeling the system in terms of functions, the participants exhibited systems thinking, corresponding to the code mapping between models, by actively referencing their use case narratives or problem statement while constructing the activity diagrams. Participants also occasionally returned to the previously constructed use case narratives to make updates based on details that they included in their activity diagrams. This corresponds to the systems thinking code of alignment. There were also instances of participants reworking the diagrams that were currently constructing based on newly observed details and this corresponds to the systems thinking code of model coherence. The participants were overall proficient in terms of applying abstract thinking and systems thinking while modeling the system in terms of functions.

To explore systems thinking further, it is important to recognize that participants took two different approaches towards modeling the information system in terms of structures and behaviors. Half of the participants modeled the information system in terms of structures first while the rest of the participants modeled the information system in terms of behaviors. As such, the participants who modeled the information system in terms of structures first exhibited systems thinking by referencing the problem statement, use case narratives, and activity diagrams while constructing the class diagrams. These participants would then reference the class diagrams while constructing the sequence diagrams. The participants who modeled the information system in terms of behaviors first exhibited systems thinking by referencing problem statement, use case narratives, and activity diagrams while constructing the class diagrams. These participants would then reference the sequence diagrams while constructing the class diagrams. It must be noted that with the exception of one participant, there were no substantial differences in terms of overall software modeling proficiency between the two approaches.

CHAPTER 7. DISCUSSION AND IMPLICATIONS

This section summarized the results and findings from the quantitative and qualitative strands of the study in the context of literature. Following this, the theoretical and methodological implications of this research were presented as well as how these results and findings could be utilized to inform instructional practices in higher education settings.

7.1 Characterizing Abstract Thinking

The results of the quantitative strand point to students being proficient in terms of applying abstract thinking while modeling the information system in terms of functions. However, there was an overall reduction in proficiency of abstract thinking while modeling the system in terms of structures and behaviors. This aligns with the notion that abstract thinking can be applied at multiple levels (Hadar & Hadar, 2006; Kramer, 2007; Zehetmeier et al., 2019). The differences in proficiency of modeling information systems in terms of functions, structures, and behaviors also runs contrary to studies conducted in other fields such as physics, engineering, biology, and medicine (Chi, De Leeuw, Chiu, & Lavancher, 1994; Hmelo-Silver & Pfeffer, 2004; Lammi, 2011; Vattam et al., 2011). An explanation for this discrepancy in information systems compared to other fields could be how physical systems readily present their structural and behavioral elements. This contrasts with how information systems are primarily perceived in terms of the functionality offered by them with users of these systems not necessarily pausing to consider the structural and behavioral elements that implement these functions.

This study specifically deals with problem, object, and program levels of abstraction as portrayed by Perrenet (2010). Modeling the information system in terms of functions aligns with the problem level of abstraction. The problem level of abstraction was defined as the highest level of abstract thinking and the students were generally proficient at it. Modeling the information system in terms of structures and behaviors aligns with the object and program levels of abstract thinking which are considered to be lower levels of abstract thinking and interestingly the students were less proficient at these levels. The students experienced some difficulty in accurately implementing data abstraction – defining data structures and relationships; and procedural abstraction – defining functional calls (Liskov 1988; Morgan, 1988). While the patterns of proficiency remained similar across both strands of this study, it must be noted that the participants in the qualitative strand generally received better scores for their models. This could be, at least in part, attributed to how participants of the qualitative strand of this study were of junior or senior standing and had completed programming courses centered around different programming languages such as C++ and Java. This provided them with more opportunities to practice the cognitive work involved in moving between various levels of abstraction. In comparison, students involved in the quantitative strand who had only completed one introductory course centered around object-oriented programming.

The clustering analysis revealed the existence of two distinct groups of students – high performing students who were more proficient in applying abstract thinking in terms of functions, structures, and behaviors compared to the moderate performing students. The correlational analysis of these clusters revealed statistically significant positive correlations between abstract thinking ability and software modeling proficiency. This further substantiates claims made in literature that abstract thinking is essential in the process of constructing models (Devlin, 2003; Kramer, 2007; Nguyen & Wong, 2001). The logistic regression model pointed to abstract thinking ability in terms of behaviors as being one of two key components to overall high performance in terms of software modeling proficiency.

The qualitative strand of this study provided an insight into how the participants employed abstract thinking while modeling systems in terms of functions, structures, and behaviors. The codes identified as part of the thematic analysis process adds to the literature in terms of how abstract thinking can be employed in the context of systems analysis and design. Across the different models, participants employed aggregation to consolidate related elements and decomposition to break down elements with multiple components. While constructing the use case narratives and activity diagrams, the participants focused on identifying actors and the flow of control between the users and the system through actions and responses. These codes correspond with some of the functional analysis games such as critical event analysis, cause-and-effect analysis, and problem-centered analysis (Collins & Ferguson, 1993; Sherry & Trigg, 1996). The participants were overall proficient in modeling the system in terms of functions although some participants did fail to conform to UML standards in terms of modeling swimlanes, use of decision and merge nodes, and the number and location of start/stop nodes.

When constructing class diagrams, the participants vocalized identifying the various relevant objects and the relationships between them. However, it must be noted that the participants were less successful in identifying all relevant objects and relationships. This could be attributed to the participants not always correctly identifying objects relevant to the solution and missing associated relationships as a direct result of this. These codes correspond to structure analysis games such as primitive-elements analysis and spatial decomposition which the participants employed to identify the relevant objects and relationships between them (Collins & Ferguson, 1993; Sherry & Trigg, 1996).

Overall proficiency in terms of modeling structures using class diagrams was negatively affected by failing to list relevant attributes and behaviors associated with every object and failing to appropriately label relationships. When constructing sequence diagrams, the participants vocalized identifying the relevant objects as well as identifying the flow of messages and responses between them. These codes correspond to the process analysis games of identifying systems-dynamics models and situation-action models to capture the objects and the interactions between them (Collins & Ferguson, 1993; Sherry & Trigg, 1996). It must be noted that the participants often did not identify all relevant objects or failed to identify correct messages and responses that go between these objects which had a negative impact on overall proficiency.

Through identifying actors, objects, relationships, messages and responses etc. abstract thinking was used across the functional, structural, and behavioral models to create models that map to real-world constructs (Devlin, 2003; Kramer, 2007) and capture details that are essential to the solution (Hadar & Hadar, 2006; Kramer, 2007; Zehetmeier et al., 2019).

7.2 Characterizing Systems Thinking

Systems thinking was broadly defined in literature as the ability of an individual to capture different views of a system and think of a system as involving inter-related components instead of as independent parts (Godfrey, Deakin Crick, & Huang, 2014). This aligns with the concept of epistemic fluency, which was defined as the ability to organize knowledge into different patterns while making sense of a problem in different ways (Sherry & Trigg, 1996). For the purposes of this study, students and participants employed the epistemic games of functional analysis, structural analysis, and process analysis to model the system in terms of functions, structures, and behaviors. They accomplished this by using the epistemic forms of use case narratives and activity diagrams, class diagrams, and sequence diagrams.

The results of the quantitative strand point to students being proficient in terms of applying systems thinking while modeling the information system in terms of functions where 79% of the students successfully aligned their actions and decisions in their activity diagrams with the typical and alternate courses of events in the use case narrative. In comparison, there was a substantial drop in overall proficiency of aligning the sequence diagrams with the class diagrams with only 41% of being successful at doing so. This could be attributed to including objects in the sequence diagram that were not identified in the class diagram or labeling messages with function calls that were not identified in the class diagram. However, 82% of the students successfully aligned their sequence diagrams with their activity diagrams in terms of overall flow. Even if there were objects missing or messages and responses were mapped to incorrect objects, the overall flow of control of the system was still captured. These results corroborate the findings from a prior study where students encountered difficulty in making connections between different UML diagrams (Burgueño, Vallecillo, & Gogolla, 2018).

The clustering analysis revealed the existence of two distinct groups of students – high performing students who were more proficient in applying systems thinking in terms of functions, structures, and behaviors compared to the moderate performing students. The correlational analyses of these clusters revealed statistically significant positive correlations systems thinking ability - given by alignment of activity diagrams with use case narratives and alignment of sequence diagrams with class diagrams – overall software modeling proficiency. Interestingly, there were no statistically significant correlations found between sequence diagram totals and alignment of sequence diagrams with activity diagrams which could be because vast majority of students were proficient at it. However, the logistic regression model pointed to systems thinking ability in terms of aligning sequence diagrams with activity diagrams as being one of two key components to overall high performance in terms of software modeling proficiency.

The qualitative strand of this study provided an insight into how the participants employed systems thinking while modeling systems in terms of functions, structures, and behaviors. Given the uniform practice of systems thinking that has been developed (Sevaldson, 2011), the codes identified as part of the thematic analysis process adds to the literature in terms of how systems thinking can be employed in the context of systems analysis and design. The thematic analysis process revealed that participants employed systems thinking by actively referencing previously constructed models – corresponding to the code, mapping between models – or by retroactively make changes to previously constructed models based on new details that were observed – corresponding to the code, alignment. In addition to this, participants also actively modified their models in response to new details that they incorporated – corresponding to the code, model coherence. These characterizations of systems thinking align with literature definitions that portray it as an approach for seeing the relationships between different parts of the system (Godfrey, Deakin Crick, & Huang, 2014) and recognizing that changes made in part of the system can affect other parts (Wolstenholme, 2003).

The qualitative strand also revealed that participants took two distinct approaches towards modeling the given system in terms of functions, structures, and behaviors. All participants modeled the systems in terms of functions first. Participants referenced the problem statement while constructing the use case narratives. They would then reference the use case narrative, and at times the problem statement also, while constructing the activity diagrams. Following this, half of the participants modeled the system in terms of behaviors using sequence diagrams then structures using class diagrams, and the others modeled the system in terms of structures using class diagrams then behaviors using sequence diagrams. In the first approach, the participants constructed the sequence diagrams while primarily referencing the use case narrative and activity diagram. Some participants also referenced the problem statement. The class diagram was then constructed primarily referencing the sequence diagram although some participants did reference the problem statement, use case narrative, and activity diagram as well. In the latter approach, participants constructed the class diagrams first. They identified objects and the relationships between them by referencing the problem statement and activity diagram. Following this they constructed the sequence diagram by referencing the class diagram. This group could have been conditioned to model structures before behaviors based on It must be noted that in the case of either approach, there were no substantial differences in terms of exhibited systems thinking ability. Participants were generally proficient in aligning their activity diagrams with their use case narratives and aligning their sequence diagrams with their activity diagrams. However, participants were generally less proficient in aligning their class diagrams and sequence diagrams. It must be noted that the participants tended to apply systems thinking proactively by referencing previously constructed models. Participants also retroactively made changes to already constructed models based on new details, albeit less frequently.

7.3 Implications

7.3.1 Theoretical and methodological implications

The theoretical implications of this study were two-fold. The review of literature conducted as part of this study revealed that abstract thinking has been defined and operationalized in different ways depending on the context (Zehetmeier et al., 2019). Systems thinking also has been defined in numerous context-dependent ways (Brewer & Dittman, 2018; Godfrey, Deakin Crick, & Huang, 2014; Senge, 1990; Stearman, 2000). However, in the context of modeling information systems, there has been limited research conducted regarding abstract thinking or systems thinking. Therefore, there were no pre-existing operational definitions that could be directly utilized. The first theoretical contribution of this study includes the operationalization of abstract thinking and systems thinking in the context of information systems modeling. Abstract thinking was operationalized in terms of percentage of relevant details included in the functional, structural, and behavioral models, and systems thinking was operationalized in terms of alignment between the different models. The second theoretical implication was related to epistemic forms, epistemic games, and the SBF framework. Literature generally points to structural analysis games being the simplest and process analysis games being the most complex with functional analysis games being right between (Collins & Ferguson, 1993; Sherry & Trigg, 1996). In numerous studies across different domains that utilized the SBF framework, subjects were typically most proficient at identifying structural elements of given system while encountering some difficulty in identifying functions and behaviors (Chi, De Leeuw, Chiu, & Lavancher, 1994; Hmelo-Silver & Pfeffer, 2004; Lammi, 2011; Vattam et al., 2011). Most of these studies supported the notion that experts rather than novices tended to organize systems by functions and behaviors. However, this study revealed that in the context of information systems, students were most proficient at the epistemic game of functional analysis, and least proficient at behavioral analysis. These results also corresponded to the abstract thinking ability displayed by students while modeling the given information systems in terms of functions, structures, and behaviors. Figure 7.1 illustrates how abstract thinking and systems thinking are essential to the process of translating a problem statement into software models. The figure also demonstrates how the concepts of epistemic forms and games are utilized in the context of information systems modeling.

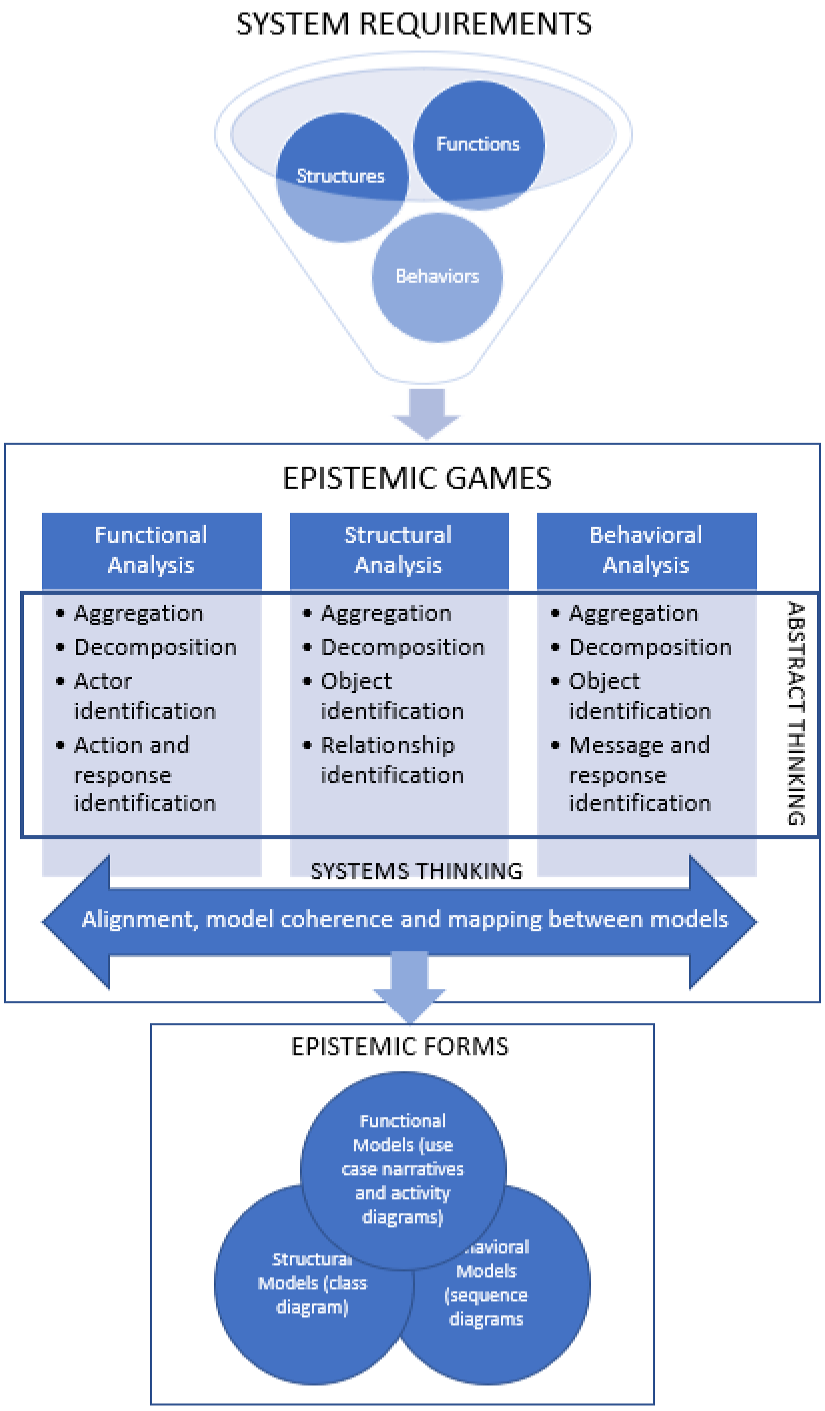


Figure 7.1: Software modeling process

Epistemic games are employed to model an information system in terms of functions, structures, and behaviors. The epistemic games of functional analysis, structural analysis, and behavioral analysis all utilize abstract thinking to extract relevant details from the problem statement to construct the models. Systems thinking is utilized either proactively or reactively to bring about alignment and coherence between elements of the different models. Ultimately, the outcomes of this are the functional, structural, and behavioral models i.e. the epistemic forms.

The methodological implications of this study pertain to the rubrics that were developed to evaluate the functional, structural, and behavioral models in terms of abstract thinking and systems thinking as well as conformity to UML standards. The validity and reliability of these rubrics were established through a combination of face validity and interrater reliability. These rubrics can be utilized in systems development courses to score or grade UML activity diagrams, class diagrams, and sequence diagrams while also evaluating the abstract thinking and systems thinking ability of students.

7.3.2 Implications for teaching and learning

The quantitative results and qualitative findings from the study indicate that students were generally proficient in applying abstract thinking and systems thinking while modeling information systems in terms of functions through use case narratives and UML activity diagrams. However, they were generally less proficient at applying abstract thinking and systems thinking while modeling information systems in terms of structures and behaviors. The results of the clustering analysis revealed the existence of a cluster of high-performing students (n=57) who were significantly more proficient at applying abstract thinking and systems thinking. It could be inferred that the design conjectures and theoretical conjectures of the sophomore-level systems analysis and design course – detailed in Chapter 4 - were successful in equipping more than half of the students with the abstract thinking and systems thinking skills required to proficiently model information systems in terms of functions, structures, and behaviors. The learners enrolled in this course engaged in project-based cooperative learning through the in-class modeling activities and the Scrum-based team project – the design conjectures - to produce UML models that described the system in terms of functions, structures, and behaviors. The theoretical conjecture that the development of these UML software models would lead to increased software modeling proficiency, abstract thinking skills and systems thinking skills was substantiated by the cluster of high-performing students. However, the relative lack of proficiency of abstract thinking and systems thinking while modeling systems in terms of structures and behaviors of the moderate performing students (n=40) need to be addressed.

The discrepancy in modeling proficiency could be addressed by structured exercises walking students through the process of constructing structural and behavioral models. These exercises can cover the mechanical aspects of constructing structural and behavioral models such as adhering to UML standards in terms of symbols and notations used. However, the focus of these exercises should explicitly be on how to apply abstract thinking and systems thinking while modeling a system in terms of structures and behaviors. This approach would involve integrating the tenets of cognitive apprenticeship.

Cognitive apprenticeship aims to assist learners in understanding when a skill can or cannot be used, and when to generalize a skill so that it can be applied in novel situations (Collins, Brown, & Holum, 1991). Cognitive apprenticeship methods include modeling, coaching, scaffolding, articulation, reflection, and exploration (García-Cabrero et al., 2018). Modeling involves the instructor performing a task while the students observe. The instructor can guide learners through the process of developing software models when given a problem statement. Through this guided exercise, learners can gain insights into how can abstract thinking be applied to extract the details relevant to the solution. The instructor should elaborate on the rationale behind each design decision, such as why some elements were included in the model – in terms of objects, relationships, messages, and responses – while others were not. The instructors should also illustrate systems thinking by actively referencing previously constructed models and explaining the connections between the different models of the system. Examples of this can include how objects in the sequence diagram must exist in the class diagram and how the messages in the sequence diagram should be labeled with function calls corresponding to behaviors listed in the class diagram. Systems thinking can also be illustrated by adopting both approaches taken by participants in the qualitative strand of this study in terms of modeling structures before behaviors and modeling behaviors before structures. Following the modeling exercises, coaching can be employed where learners are given cases to construct models for while the instructor provides hints or feedback. Scaffolding can be implemented by providing students with partially complete models for example cases with the expectation that they complete the models. The partially completed portions should ideally illustrate UML conformity and highlight the correct symbols to be used when modeling different systems. Articulation and reflection could be implemented together in the form of small group activities where learners can verbalize their different approaches towards constructing the UML models and they can receive feedback from their peers. The exploration aspect of cognitive apprenticeship can be facilitated by providing students with a complex case with expectation that they are to identify different functionalities offered by the system. Following this, the students can be asked to model the system in terms of functions, structures, and behaviors for each functionality. It must be noted that the sophomore-level systems development course detailed in Chapter 4 already implements elements of cognitive apprenticeship, specifically that of modeling, articulation, reflection, and exploration - through the in-class modeling activities as well as the term project (Magana, Seah, & Thomas, 2018). However, incorporating coaching and scaffolding with an explicit focus on structural and behavioral modeling could help improve student proficiency in modeling structural and behavioral aspects of information systems. The tenets of cognitive apprenticeship align strongly with the principles of social constructivism by promoting interaction and collaboration between learners as well as between the instructor and learners.

Another approach that could address the relative lack of structural and behavioral modeling proficiency among students is an exercise that provides students with complete functional code and the problem statement that they must use to construct UML models. The code would provide learners with insights into how the problem statement was implemented in terms of structures and behaviors – corresponding to classes and their respective attributes and behaviors. This exercise could also serve to improve the abstract thinking and systems thinking ability of students by being forced to determine what details included in the problem statement are relevant to the solution and how the different elements are connected.

CHAPTER 8. CONCLUSION

8.1 Limitations

The results and findings from this study were subject to the following limitations. The data for the quantitative strand was collected from students who were enrolled in a sophomore-level systems analysis and design course at a large midwestern university while the data for the qualitative strand was collected from students of junior senior standing who were part of the department of computer and information technology and had already completed the sophomore-level systems analysis and design course at the same university. Therefore, there were two distinct samples. Both samples were not necessarily representative of the student population of the university. As such, the results may not be generalizable to other universities. The data of the quantitative strand was limited by the preparation done by the students for their second mid-term exam. Due to logistical constraints, the exam was conducted in two parts so there was potential for interaction between the students prior to the second part. The qualitative strand was limited to the number of volunteer participants who had completed the sophomore-level course. For each participant in the qualitative strand, the data was collected in a single three-hour session. As a result, participants did experience fatigue. This was mitigated in part by allowing the participants to take breaks as required. The qualitative strand was also subject to limitations associated with the think-aloud process where codes were only identified based on instances where participants vocalized design decisions or explained their rationale upon being prompted by the researcher. Neither strand explored student motivation or its relationship with proficiency of abstract thinking or systems thinking. The study also did not explore the effect of prior knowledge of programming on abstract thinking or systems thinking ability exhibited while modeling information systems.

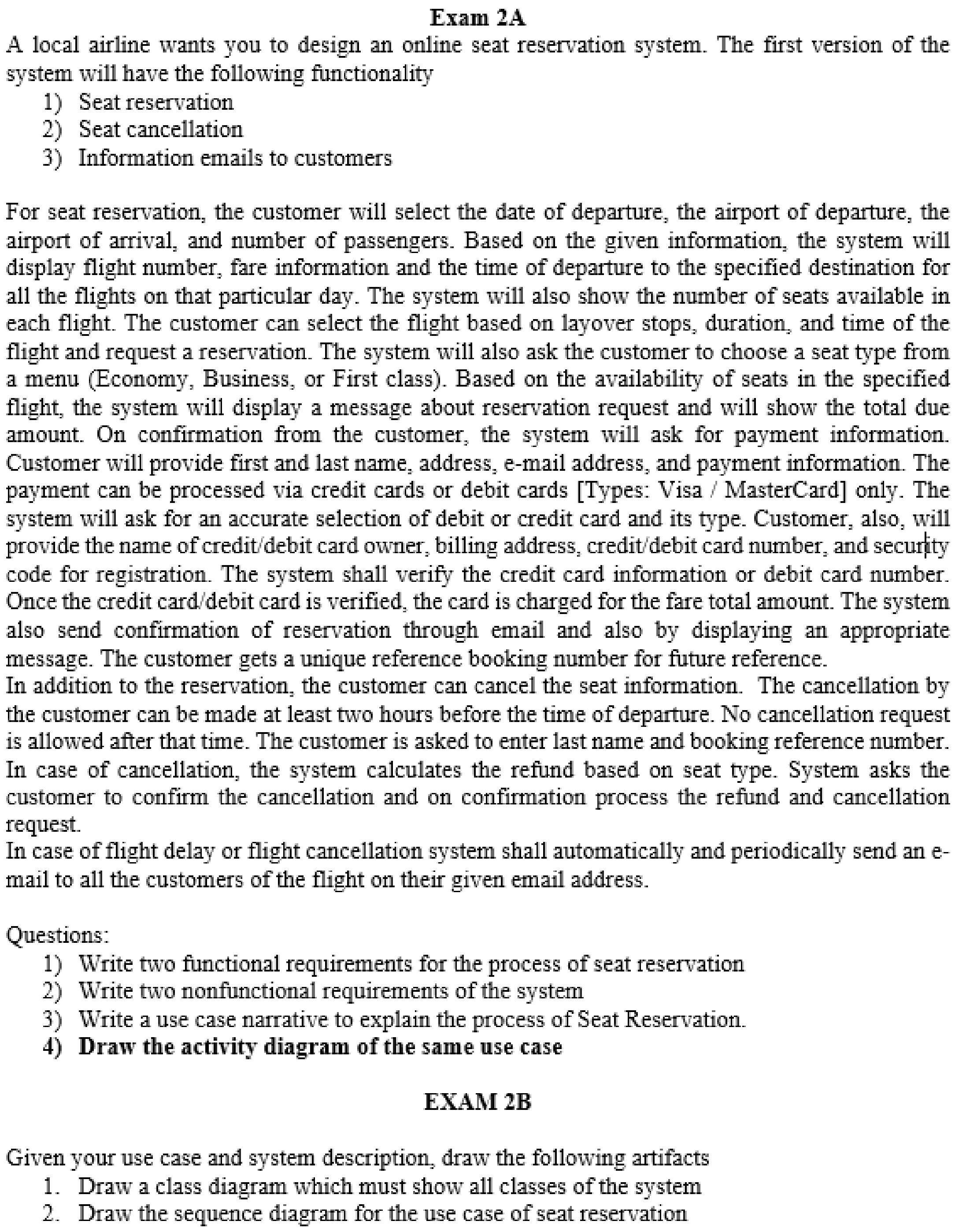
8.2 Conclusion and Future Work

The goal of this study was to characterize abstract thinking and systems thinking proficiency exhibited by students while modeling information systems in terms of functions, structures, and behaviors using UML models. This was accomplished by employing a multi-methods approach towards measuring and characterizing abstract thinking and systems thinking in the context of information systems modeling. The study was successful in employing the SBF framework in the context of information systems modeling and the patterns of proficiency exhibited by the participants was found to be dramatically different from that of other studies conducted in other domains - specifically that the students exhibited greater proficiency of modeling systems in terms of functions compared to that of modeling structures and behaviors. The study provided valuable additions to the literature pertaining to abstract thinking and systems thinking in the area of systems analysis and design. The study also tied together how epistemic forms and games can be applied in information systems modeling. The results of quantitative strand revealed patterns of proficiency among students. Abstract thinking and systems thinking generally correlated positively with software modeling proficiency. A logistic regression model was computed that identified the key elements that predicted software modeling proficiency. The rubrics developed as part of this strand can be utilized in systems analysis and design courses to evaluate software models developed by students as well as their level of abstract thinking and systems thinking. The findings from the qualitative strand revealed the different approaches taken by participants in applying abstract thinking and systems thinking while constructing the functional, structural, and behavioral models. These findings extend existing literature definitions and operationalizations of abstract thinking and systems thinking but specifically characterize these constructs in context of information systems modeling. This study makes a case for the development of learning interventions and evaluation mechanisms that can aid students in being more proficient at applying abstract thinking and systems thinking while modeling information systems in terms of structures and behaviors. This in turn would lead to more complete and accurate models being produced. Implementing cognitive apprenticeship in the classroom through guided exercises that utilize instructional scaffolds could help students better apply abstract thinking and systems thinking which in turn would improve overall software modeling proficiency.

Given the state of literature and the limitations of this study, further research pertaining to abstract thinking and systems thinking in context of modeling information systems would be useful in expanding the collective understanding of these constructs. Future work could explore the effect of each of the suggested instructional interventions on software modeling proficiency exhibited by students. A true mixed-methods approach would also help ground the findings in a specific sample. It would also be interesting to assess the effectiveness of these suggested interventions in an online instructional setting. These proposed studies could yield vital information in understanding the nature of instructional support required to bolster abstract thinking and systems thinking ability among students in the context of information systems modeling. In addition, an inquiry could be conducted into the nature of the relationship between academic achievement and constructs of abstract thinking and systems thinking. Some research questions that could guide directions for future research are as follows:

- How can cognitive apprenticeship be implemented in the classroom to improve student abstract thinking and systems thinking while modeling information systems?
- How do differences in students' prior knowledge of programming affect their software modeling proficiency?
- What is the relationship between students' abstract thinking ability and their academic achievement in a systems analysis and design course?
- How does student motivation affect their proficiency of abstract thinking and system thinking?
- What are the differences in abstract thinking and systems thinking ability exhibited by students in a fully online instructional setting compared to an in-person setting?

APPENDIX A - MODELING EXAM



Exam 2A

A local airline wants you to design an online seat reservation system. The first version of the system will have the following functionality

1) Seat reservation
2) Seat cancellation
3) Information emails to customers

For seat reservation, the customer will select the date of departure, the airport of departure, the airport of arrival, and number of passengers. Based on the given information, the system will display flight number, fare information and the time of departure to the specified destination for all the flights on that particular day. The system will also show the number of seats available in each flight. The customer can select the flight based on layover stops, duration, and time of the flight and request a reservation. The system will also ask the customer to choose a seat type from a menu (Economy, Business, or First class). Based on the availability of seats in the specified flight, the system will display a message about reservation request and will show the total due amount. On confirmation from the customer, the system will ask for payment information. Customer will provide first and last name, address, e-mail address, and payment information. The payment can be processed via credit cards or debit cards [Types: Visa / MasterCard] only. The system will ask for an accurate selection of debit or credit card and its type. Customer, also, will provide the name of credit/debit card owner, billing address, credit/debit card number, and security code for registration. The system shall verify the credit card information or debit card number. Once the credit card/debit card is verified, the card is charged for the fare total amount. The system also send confirmation of reservation through email and also by displaying an appropriate message. The customer gets a unique reference booking number for future reference.
In addition to the reservation, the customer can cancel the seat information. The cancellation by the customer can be made at least two hours before the time of departure. No cancellation request is allowed after that time. The customer is asked to enter last name and booking reference number. In case of cancellation, the system calculates the refund based on seat type. System asks the customer to confirm the cancellation and on confirmation process the refund and cancellation request.
In case of flight delay or flight cancellation system shall automatically and periodically send an e-mail to all the customers of the flight on their given email address.

Questions:

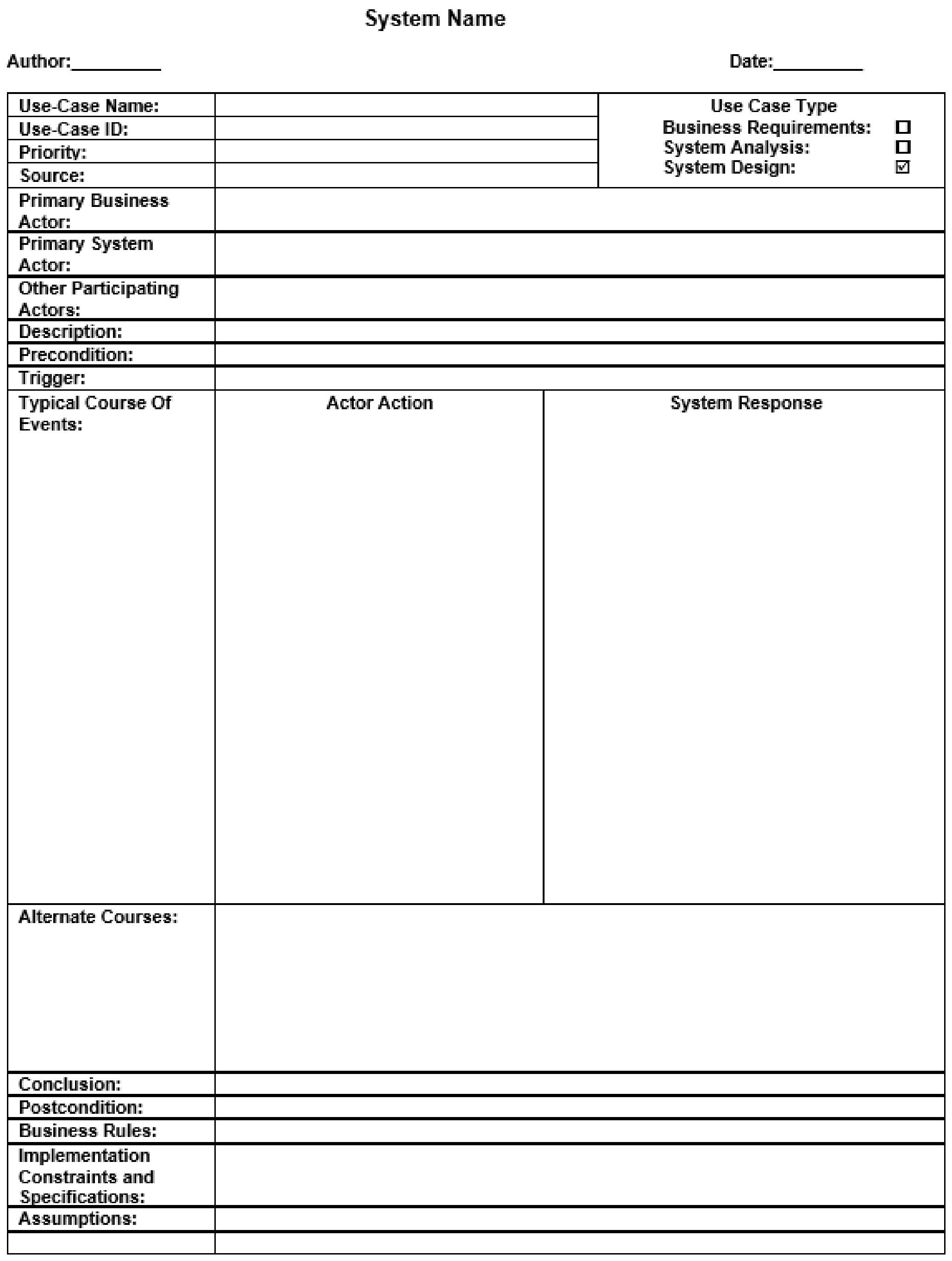
1) Write two functional requirements for the process of seat reservation
2) Write two nonfunctional requirements of the system
3) Write a use case narrative to explain the process of Seat Reservation.
4) **Draw the activity diagram of the same use case**

EXAM 2B

Given your use case and system description, draw the following artifacts

1. Draw a class diagram which must show all classes of the system
2. Draw the sequence diagram for the use case of seat reservation

APPENDIX B - USE CASE NARRATIVE TEMPLATE



System Name

Author:_________ Date:_________

Use-Case Name:		Use Case Type Business Requirements: ☐ System Analysis: ☐ System Design: ☑
Use-Case ID:		
Priority:		
Source:		
Primary Business Actor:		
Primary System Actor:		
Other Participating Actors:		
Description:		
Precondition:		
Trigger:		
Typical Course Of Events:	Actor Action	System Response
Alternate Courses:		
Conclusion:		
Postcondition:		
Business Rules:		
Implementation Constraints and Specifications:		
Assumptions:		

APPENDIX C - USE CASE NARRATIVE RUBRIC

	Absent (0)	Deficient (1)	Developing (2)	Emerging (3)	Proficient (4)
Typical Course of Events	Typical course of events absent	10-50% of typical course of events	51-70% of typical course of events	71-90% of typical course of events	91-100% of typical course of events
Alternate Courses	Alternate courses absent	10-50% of alternate courses	51-70% of alternate courses	71-90% of alternate courses	91-100% of alternate courses
Narrative Completeness	Narrative details absent	10-50% of narrative details complete	51-70% of narrative details complete	71-90% of narrative details complete	91-100% of narrative details complete

APPENDIX D - ACTIVITY DIAGRAM RUBRIC

	Absent (0)	Deficient (1)	Developing (2)	Emerging (3)	Proficient (4)
Start/Stop Nodes	Start and Stop nodes absent	Missing either start or stop node or used incorrect symbols	Multiple start/stop nodes or missing either node or used incorrect symbols	Start and/or stop nodes drawn within swimlanes	Exactly one start and one stop node shown outside the swimlanes
Swimlanes/Actors	No swimlanes shown	Showed 10-50% swimlanes and/or did not label swimlanes	Showed 51-70% swimlanes and/or did not label swimlanes	Showed and labeled 71-90% swimlanes	Showed and labeled 91-100% swimlanes
Activities	No activities shown	10-50% appropriate activities shown, some with verb, some include object, MANY compound activities	51-70% appropriate activities shown, some with verb, some include object, MANY compound activities	71-90% appropriate activities shown, begin with verb, include object, FEW compound activities	91-100% of appropriate activities shown, begin with verb, include object, NO compound activities
Decisions/Merges	No decisions or merges shown	10-50% decisions and branches labeled. Did not use merge nodes to connect multiple control flows entering the same node	51-70% decisions and branches labeled. Did not use merge nodes to connect multiple control flows entering the same node	71-90% decisions and branches labeled. Used merge nodes sometimes to connect multiple control flows entering the same node	91-100% decisions and branches labeled. Used merge nodes ALWAYS to connect multiple control flows entering the same node
Abstraction	No relevant details from problem statement included in diagram	10-50% of relevant details from problem statement included in diagram	51-70% of relevant details from problem statement included in diagram	71-90% of relevant details from problem statement included in diagram	91-100% of relevant details from problem statement included in diagram
Alignment with Use Case Narrative	None of the actions in the activity diagram mapped to steps in the use case narrative	10-50% of the actions in the activity diagram mapped to steps in the use case narrative	51-70% of the actions in the activity diagram mapped to steps in the use case narrative	71-90% of the actions in the activity diagram mapped to steps in the use case narrative	91-100% of the actions in the activity diagram mapped to steps in the use case narrative

APPENDIX E - CLASS DIAGRAM RUBRIC

	Absent (0)	Deficient (1)	Developing (2)	Emerging (3)	Proficient (4)
Objects	No objects shown	10-50% objects correctly identified and named	51-70% objects correctly identified and named	71-90% objects correctly identified and named	91-100% objects correctly identified and named
Attributes	No attributes shown	10-50% attributes listed	51-70% attributes listed	71-90% attributes listed	91-100% attributes listed
Behaviors	No behaviors shown	10-50% of behaviors listed	51-70% of behaviors listed	71-90% of behaviors listed	91-100% of behaviors listed
Relationships	No relationships shown	10-50% of relationships named bi-directionally with correct multiplicity	51-70% of relationships named bi-directionally with correct multiplicity	71-90% of relationships named bi-directionally with correct multiplicity	91-100% of relationships named bi-directionally with correct multiplicity
Abstraction	No relevant details from problem statement included in diagram	10-50% of relevant details from problem statement included in diagram	51-70% of relevant details from problem statement included in diagram	71-90% of relevant details from problem statement included in diagram	91-100% of relevant details from problem statement included in diagram

APPENDIX F - SEQUENCE DIAGRAM RUBRIC

	Absent (0)	Deficient (1)	Developing (2)	Emerging (3)	Proficient (4)
Objects	No objects shown	10-50% of objects correctly identified	51-70% of objects correctly identified	71-90% of objects correctly identified	91-100% of objects correctly identified
Lifelines	No lifelines shown	10-50% of lifelines shown	51-70% of lifelines shown	71-90% of lifelines shown	91-100% lifelines shown
Processes / Execution Occurrences	No processes shown	10-50% of processes shown	51-70% of processes shown	71-90% of processes shown	91-100% processes shown
Messages	No messages shown	10-50% of messages shown and/or correctly labeled with appropriate parameters	51-70% of messages shown and/or correctly labeled with appropriate parameters	71-90% of messages shown and/or correctly labeled with appropriate parameters	91-100% of messages shown and correctly labeled with appropriate parameters
Responses	No responses shown	10-50% of responses shown and/or correctly labeled	51-70% of responses shown and/or correctly labeled	71-90% of responses shown and correctly labeled	91-100% of responses shown and correctly labeled
Abstraction	No relevant details from problem statement included in diagram	10-50% of relevant details from problem statement included in diagram	51-70% of relevant details from problem statement included in diagram	71-90% of relevant details from problem statement included in diagram	91-100% of relevant details from problem statement included in diagram
Alignment with Class Diagram	None of the objects and messages in the sequence diagram map to classes and functions in the class diagram	10-50% of the objects and messages in the sequence diagram map to classes and functions in the class diagram	51-70% of the objects and messages in the sequence diagram map to classes and functions in the class diagram	71-90% of the objects and messages in the sequence diagram map to classes and functions in the class diagram	91-100% of the objects and messages in the sequence diagram map to classes and functions in the class diagram
Alignment with Activity Diagram	None of the messages and responses map to actions in the activity diagram	10-50% of the messages and responses map to actions in the activity diagram	51-70% of the messages and responses map to actions in the activity diagram	71-90% of the messages and responses map to actions in the activity diagram	91-100% of the messages and responses map to actions in the activity diagram

APPENDIX G – STAFF MANAGEMENT CASE

Professional and Scientific Staff Management (PSSM) is a unique type of temporary staffing agency. When a PSSM client company determines that it will need a temporary professional or scientific employee, it issues a staffing request against the contract it had previously negotiated with PSSM. When PSSM's contract manager receives a staffing request, the contract number referenced on the staffing request is entered in the contract database. Using information from the database, the contract manager reviews the terms and conditions of the contract and determines whether the staffing request is valid. The staffing request is valid if the contract has not expired, the type of professional or scientific employee requested is listed on the original contract, and the requested fee falls within the negotiated fee range. If the staffing request is not valid, the contract manager sends the staffing request back to the client with a letter stating why the staffing request cannot be filled, and a copy of the letter filed. If the staffing request is valid, the contract manager enters the staffing request database as an outstanding staffing request. The staffing request is then sent to the PSSM placement department.

In the placement department, the type of staff member, experience, and qualifications requested on the staffing request are checked against the database of available professional and scientific staff. If a qualified individual is found, the individual is marked "reserved" in the staff database. If a qualified individual cannot be found in the database or is not immediately available, the placement department creates a memo that explains the inability to meet the staffing request and attaches it to the staffing request. All staffing requests are then sent to the arrangements department.

In the arrangements department, the prospective temporary employee is contacted and asked to agree to the placement. After the placement details have been worked out and agreed to, the staff member is marked "placed" in the staff database. A copy of the staffing request and a bill for the placement fee is sent to the client. Finally, the staffing request, the "unable-to-fill" memo (if any), and a copy of the placement fee bill are sent to the contract manager. If the staffing request was filled, the contract manager closes the open staffing request in the staffing request database. If the staffing request could not be filled, the client is notified.

Q. Model the functions, structures, and behaviors of the staff request use case. Invoke other functionalities and use cases as required.

APPENDIX H – STEP-WISE LOGISTIC REGRESSION RESULTS

	Variables in the Equation	B	S.E.	Wald	df	Sig.	Exp(B)
Step 1a	Proficiency of Abstraction - Activity Diagram	-0.311	0.382	0.664	1	0.415	0.733
	Alignment with Use Case Narratives	0.006	0.504	0	1	0.991	1.006
	Proficiency of Abstraction - Class Diagram	-0.139	0.445	0.097	1	0.755	0.87
	Proficiency of Abstraction - Sequence Diagram	0.44	0.467	0.886	1	0.346	1.552
	Alignment with Class Diagrams - SD	0.086	0.356	0.059	1	0.808	1.09
	Alignment with Activity Diagram - SD	0.721	0.379	3.619	1	0.057	2.057
	Constant	-2.451	1.945	1.589	1	0.208	0.086
Step 2a	Proficiency of Abstraction - Activity Diagram	-0.309	0.335	0.852	1	0.356	0.734
	Proficiency of Abstraction - Class Diagram	-0.138	0.444	0.097	1	0.755	0.871
	Proficiency of Abstraction - Sequence Diagram	0.44	0.467	0.888	1	0.346	1.552
	Alignment with Class Diagrams - SD	0.087	0.356	0.059	1	0.808	1.091
	Alignment with Activity Diagram - SD	0.722	0.371	3.787	1	0.052	2.059
	Constant	-2.442	1.775	1.893	1	0.169	0.087
Step 3a	Proficiency of Abstraction - Activity Diagram	-0.312	0.335	0.868	1	0.351	0.732
	Proficiency of Abstraction - Class Diagram	-0.152	0.44	0.12	1	0.729	0.859
	Proficiency of Abstraction - Sequence Diagram	0.51	0.367	1.936	1	0.164	1.666
	Alignment with Activity Diagram - SD	0.729	0.37	3.885	1	0.049	2.073
	Constant	-2.347	1.732	1.836	1	0.175	0.096
Step 4a	Proficiency of Abstraction - Activity Diagram	-0.342	0.325	1.105	1	0.293	0.711
	Proficiency of Abstraction - Sequence Diagram	0.455	0.329	1.907	1	0.167	1.576
	Alignment with Activity Diagram - SD	0.719	0.367	3.836	1	0.05	2.052
	Constant	-2.533	1.643	2.377	1	0.123	0.079
Step 5a	Proficiency of Abstraction - Sequence Diagram	0.383	0.319	1.443	1	0.23	1.467
	Alignment with Activity Diagram - SD	0.66	0.36	3.35	1	0.067	1.934
	Constant	-3.252	1.505	4.67	1	0.031	0.039
Step 6a	Alignment with Activity Diagram - SD	0.762	0.347	4.817	1	0.028	2.143
	Constant	-2.474	1.315	3.541	1	0.06	0.084

REFERENCES

[ABET]. (2016). Criteria for Accrediting Engineering Programs Effective for Reviews during the 2017-2018 Accreditation Cycle. Retrieved November 15, 2019 from http://www.abet.org/wp-content/uploads/2016/12/E001-17-18-EAC-Criteria-10-29-16-1.pdf

Abbas, O. A. (2008). Comparisons between data clustering algorithms. *International Arab Journal of Information Technology (IAJIT), 5*(3).

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile software development methods: Review and analysis.

Adey, P., & Shayer, M. (2006). *Really raising standards: Cognitive intervention and academic achievement*. Routledge.

Aljumaily, H., Cuadra, D., & Laefer, D. F. (2019). An empirical study to evaluate students' conceptual modeling skills using UML. *Computer Science Education, 29*(4), 407-427.

Alter, S. (2004). Desperately seeking systems thinking in the information systems discipline. *ICIS 2004 proceedings*, 61.

Amineh, R. J., & Asl, H. D. (2015). Review of constructivism and social constructivism. *Journal of Social Sciences, Literature and Languages*, *1*(1), 9-16.

Anguera, M. T., Blanco-Villaseñor, A., Losada, J. L., Sánchez-Algarra, P., & Onwuegbuzie, A. J. (2018). Revisiting the difference between mixed methods and multimethods: Is it all in the name?. *Quality & Quantity, 52*(6), 2757-2770.

Anitha, P., & Patil, M. M. (2019). RFM model for customer purchase behavior using K-Means algorithm. *Journal of King Saud University-Computer and Information Sciences*.

Aranganayagi, S., & Thangavel, K. (2007, December). Clustering categorical data using silhouette coefficient as a relocating measure. In *International conference on computational intelligence and multimedia applications (ICCIMA 2007)* (Vol. 2, pp. 13-17). IEEE.

Archer, K. J., & Lemeshow, S. (2006). Goodness-of-fit test for a logistic regression model fitted using survey sample data. The Stata Journal, 6(1), 97-105.

Artusi, R., Verderio, P., & Marubini, E. (2002). Bravais-Pearson and Spearman correlation coefficients: meaning, test of hypothesis and confidence interval. *The International journal of biological markers, 17*(2), 148-151.

Balaban, M., & Maraee, A. (2013). Finite satisfiability of UML class diagrams with constrained class hierarchy. *ACM Transactions on Software Engineering and Methodology (TOSEM), 22*(3), 24.

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice*. Addison-Wesley Professional.

Baxter, P., & Jack, S. (2008). Qualitative case study methodology: Study design and implementation for novice researchers. *The qualitative report, 13*(4), 544-559.

Bennedssen, J., & Caspersen, M. E. (2008). Abstraction ability as an indicator of success for learning computing science?. In *Proceedings of the Fourth international Workshop on Computing Education Research* (pp. 15-26). ACM.

Berry, K. J., & Mielke Jr, P. W. (1988). A generalization of Cohen's kappa agreement measure to interval measurement and multiple raters. *Educational and Psychological Measurement*, *48*(4), 921-933.

Boersma, K., Waarlo, A. J., & Klaassen, K. (2011). The feasibility of systems thinking in biology education. *Journal of Biological Education*, *45*(4), 190-197.

Bonett, D. G., & Wright, T. A. (2000). Sample size requirements for estimating Pearson, Kendall and Spearman correlations. *Psychometrika*, *65*(1), 23-28.

Booch, G. (1999). UML in action. *Communications of the ACM*, *42*(10), 26-28.

Böttcher, A., Schlierkamp, K., Thurner, V., & Zehetmeier, D. (2016, October). Teaching abstraction. In *2nd. International conference on higher education advances (HEAD'16)* (pp. 357-364). Editorial Universitat Politècnica de València.

Boustedt, J. (2012). Students' different understandings of class diagrams. *Computer Science Education*, *22*(1), 29-62.

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology*, *3*(2), 77-101.

Braun, V., & Clarke, V. (2012). Thematic analysis.

Brewer, J. L., & Dittman, K. C. (2018). *Methods of IT project management*. Purdue University Press.

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

Burgueño, L., Vallecillo, A., & Gogolla, M. (2018). Teaching UML and OCL models and their validation to software engineering students: an experience report. *Computer Science Education*, *28*(1), 23-41.

Castleberry, A., & Nolen, A. (2018). Thematic analysis of qualitative research data: Is it as easy as it sounds?. *Currents in Pharmacy Teaching and Learning*, *10*(6), 807-815.

Cernosek, G., & Naiburg, E. (2004). The value of modeling. *IBM developerWorks*.

Chatterjee, S., & Hadi, A. S. (2015). *Regression analysis by example*. John Wiley & Sons.

Chaovalit, P., & Zhou, L. (2005, January). Movie review mining: A comparison between supervised and unsupervised classification approaches. In *Proceedings of the 38th annual Hawaii international conference on system sciences* (pp. 112c-112c). IEEE.

Checkland, P. B. (1988). Information systems and systems thinking: time to unite?. *International Journal of Information Management*, *8*(4), 239-248.

Chi, M. T. (1997). Quantifying qualitative analyses of verbal data: A practical guide. *The journal of the learning sciences*, *6*(3), 271-315.

Chi, M. T., De Leeuw, N., Chiu, M. H., & LaVancher, C. (1994). Eliciting self-explanations improves understanding. *Cognitive science*, *18*(3), 439-477.

Chi, M. T., Feltovich, P. J., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive science*, *5*(2), 121-152.

Ciccozzi, F., Malavolta, I., & Selic, B. (2019). Execution of UML models: a systematic review of research and practice. *Software & Systems Modeling*, *18*(3), 2313-2360.

Clarke, E. M., Grumberg, O., & Long, D. E. (1994). Model checking and abstraction. *ACM transactions on Programming Languages and Systems (TOPLAS)*, *16*(5), 1512-1542.

Cohen, M. E. (2001). Consive review: analysis of ordinal dental data: evaluation of conflicting recommendations. *Journal of dental research*, *80*(1), 309-313.

Collins, A., & Ferguson, W. (1993). Epistemic forms and epistemic games: Structures and strategies to guide inquiry. *Educational psychologist*, *28*(1), 25-42.

Collins, A., Brown, J. S., & Holum, A. (1991). Cognitive apprenticeship: Making thinking visible. *American educator*, *15*(3), 6-11.

Coupal, C. & Boechler, K. (2005). Introducing Agile into a Software Development Capstone Project. *Proceedings from the Agile Conference, 2005*, Denver, CO.

Creswell, J. W., Hanson, W. E., Clark Plano, V. L., & Morales, A. (2007). Qualitative research designs: Selection and implementation. *The counseling psychologist*, *35*(2), 236-264.

Creswell, J. W. (2014). *A concise introduction to mixed methods research*. SAGE publications.

Creswell, J., & Plano Clark, V. (2018). Designing and conducting mixed methods research (2nd ed.). Los Angeles: SAGE Publications.

Creswell, J. W., & Poth, C. N. (2016). *Qualitative inquiry and research design: Choosing among five approaches*. Sage publications.

Curtin, M., & Fossey, E. (2007). Appraising the trustworthiness of qualitative studies: Guidelines for occupational therapists. *Australian occupational therapy journal*, *54*(2), 88-94.

Cutcliffe, J. R., & McKenna, H. P. (1999). Establishing the credibility of qualitative research findings: the plot thickens. *Journal of advanced nursing*, *30*(2), 374-380.

de Champeaux, D., Constantine, L., Jacobson, I., Mellor, S., Ward, P., & Yourdon, E. (1990). Structured analysis and object oriented analysis. *ACM SIGPLAN Notices*, *25*(10), 135-139.

Dennis, A., Wixom, B. H., & Tegarden, D. (2020). *Systems analysis and design: An object-oriented approach with UML*. John Wiley & sons.

de Winter, J. C., Gosling, S. D., & Potter, J. (2016). Comparing the Pearson and Spearman correlation coefficients across distributions and sample sizes: A tutorial using simulations and empirical data. *Psychological methods*, *21*(3), 273.

Devlin, K. (2003). Why universities require computer science students to take math. *Communications of the ACM*, *46*(9), 37.

Devlin, A. S. (2018). *Environmental Psychology and Human Well-being: Effects of Built and Natural Settings*. Academic Press.

Dobing, B., & Parsons, J. (2006). How UML is used. *Communications of the ACM*, *49*(5), 109-113.

Engels, G., & Groenewegen, L. (2000). Object-oriented modeling: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 103-116). ACM.

Eriksson, M., Börstler, J., & Borg, K. (2005). The PLUSS approach–domain modeling with features, use cases and use case realizations. In *International Conference on Software Product Lines* (pp. 33-44). Springer, Berlin, Heidelberg.

Eshuis, R. (2006). Symbolic model checking of UML activity diagrams. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *15*(1), 1-38.

Feltovich, P. J., Coulson, R. L., Spiro, R. J., & Dawson-Saunders, B. K. (1992). Knowledge application and transfer for complex tasks in ill-structured domains: Implications for instruction and testing in biomedicine. In *Advanced models of cognition for medical training and practice* (pp. 213-244). Springer, Berlin, Heidelberg.

Ferrari, M., & Chi, M. T. (1998). The nature of naive explanations of natural selection. *International journal of science education*, *20*(10), 1231-1256.

Fernández-Sáez, A. M., Chaudron, M. R., & Genero, M. (2018). An industrial case study on the use of UML in software maintenance and its perceived benefits and hurdles. *Empirical Software Engineering*, *23*(6), 3281-3345.

Field, A. (2009). Logistic regression. Discovering statistics using SPSS, 264, 315.

Fonteyn, M. E., Kuipers, B., & Grobe, S. J. (1993). A description of think aloud method and protocol analysis. *Qualitative health research*, *3*(4), 430-441.

Fowler, M. (2004). *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional.

García-Cabrero, B., Hoover, M. L., Lajoie, S. P., Andrade-Santoyo, N. L., Quevedo-Rodríguez, L. M., & Wong, J. (2018). Design of a learning-centered online environment: a cognitive apprenticeship approach. *Educational Technology Research and Development*, *66*(3), 813-835.

Gero, J. S. (1990). Design prototypes: a knowledge representation schema for design. *AI magazine*, *11*(4), 26-26.

Gero, J. S., & Kannengiesser, U. (2004). The situated function–behaviour–structure framework. *Design studies*, *25*(4), 373-391.

Gero, J. S., & McNeill, T. (1998). An approach to the analysis of design protocols. *Design studies*, *19*(1), 21-61.

Gerring, J. (2006). *Case study research: Principles and practices*. Cambridge university press.

Ghezzi, C., Jazayeri, M., & Mandrioli, D. (2002). *Fundamentals of software engineering*. Prentice Hall PTR.

Godfrey, P., Crick, R. D., & Huang, S. (2014). Systems thinking, systems design and learning power in engineering education. *International Journal of Engineering Education*.

Grohs, J. R., Kirk, G. R., Soledad, M. M., & Knight, D. B. (2018). Assessing systems thinking: A tool to measure complex reasoning through ill-structured problems. *Thinking Skills and Creativity*, *28*, 110-130.

Hadar, I., & Hadar, E. (2006). Iterative cycle for teaching object oriented concepts: from abstract thinking to specific language implementation. In *Tenth Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts, ECOOP*.

Hammer, D., & Berland, L. K. (2014). Confusing claims for data: A critique of common practices for presenting qualitative research on learning. *Journal of the Learning Sciences*, *23*(1), 37-46.

Hausmann, J. H., Heckel, R., & Taentzer, G. (2002). Detection of conflicting functional requirements in a use case-driven approach. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002* (pp. 105-115). IEEE.

Hendrix, T. D., Cross, J. H., Maghsoodloo, S., & McKinney, M. L. (2000). Do visualizations improve program comprehensibility? Experiments with control structure diagrams for Java. In *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education* (pp. 382-386).

Hill, J. H., Houle, B. J., Merritt, S. M., & Stix, A. (2008). Applying abstraction to master complexity. In *Proceedings of the 2nd international workshop on the role of abstraction in software engineering* (pp. 15-21). ACM.

Hmelo, C. E., Holton, D. L., & Kolodner, J. L. (2000). Designing to learn about complex systems. *The journal of the learning sciences*, *9*(3), 247-298.

Hmelo-Silver, C. E., & Pfeffer, M. G. (2004). Comparing expert and novice understanding of a complex system from the perspective of structures, behaviors, and functions. *Cognitive science*, *28*(1), 127-138.

Ho-Quang, T., Hebig, R., Robles, G., Chaudron, M. R., & Fernandez, M. A. (2017, May). Practices and perceptions of UML use in open source projects. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)* (pp. 203-212). IEEE.

Ivankova, N. V., Creswell, J. W., & Stick, S. L. (2006). Using mixed-methods sequential explanatory design: From theory to practice. *Field methods*, *18*(1), 3-20.

Ivankova, N. V., & Creswell, J. W. (2009). Mixed methods. *Qualitative research in applied linguistics: A practical introduction*, *23*, 135-161.

Jääskeläinen, R. (2010). Think-aloud protocol. *Handbook of translation studies*, *1*, 371-374.

Jones, S.R., Torres, V. & Arminio, J. (2014). *Negotiating the complexities of qualitative research in higher education* (2nd ed.). New York, NY: Routledge.

Jacobson, M. J. (2001). Problem solving, cognition, and complex systems: Differences between experts and novices. *Complexity*, *6*(3), 41-49.

Kamthan, P. (2016). On the Nature of Collaborations in Agile Software Engineering Course Projects. *International Journal of Quality Assurance in Engineering and Technology Education (IJQAETE)*, 5(2), 42-59.

Kazman, R., Kruchten, P., Nord, R. L., & Tomayko, J. E. (2004). *Integrating software-architecture-centric methods into the Rational Unified Process*. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

Kendall, J. (1999). Axial coding and the grounded theory controversy. *Western journal of nursing research*, *21*(6), 743-757.

Khandkar, S. H. (2009). Open coding. *University of Calgary*, *23*, 2009.

Kim, B. (2001). Social constructivism. *Emerging perspectives on learning, teaching, and technology*, *1*(1), 16.

Koppelman, H., & van Dijk, B. (2010). Teaching abstraction in introductory courses. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education* (pp. 174-178). ACM.

Kramer, J. (2007). Is abstraction the key to computing?. *Communications of the ACM*, *50*(4), 36-42.

Krefting, L. (1991). Rigor in qualitative research: The assessment of trustworthiness. *American journal of occupational therapy*, *45*(3), 214-222.

Kruchten, P. (2005). Casting software design in the function-behavior-structure framework. *IEEE software*, *22*(2), 52-58.

Lammi, M. D. (2011). Characterizing high school students' systems thinking in engineering design through the function-behavior-structure (FBS) framework.

Larsen, G. (1999). Designing component-based frameworks using patterns in the UML. *Communications of the ACM*, *42*(10), 38-45.

Likas, A., Vlassis, N., & Verbeek, J. J. (2003). The global k-means clustering algorithm. *Pattern recognition*, *36*(2), 451-461.

Liskov, B. (1988). Data abstraction and hierarchy. *SIGPLAN notices*, *23*(5), 17-34.

Magana, A. J., Seah, Y. Y., & Thomas, P. (2018). Fostering cooperative learning with Scrum in a semi-capstone systems analysis and design course. *Journal of Information System Education*.

Mahnic, V. (2012). A Capstone Course on Agile Software Development Using Scrum. *IEEE Transactions on Education*, 55(1), 99-106.

Marton, F. (1986). Phenomenography—a research approach to investigating different understandings of reality. *Journal of thought*, 28-49.

Marques, J. F., & McCall, C. (2005). The application of interrater reliability as a solidification instrument in a phenomenological study. *The Qualitative Report*, *10*(3), 439-462.

Mattsson, A., Fitzgerald, B., Lundell, B., & Lings, B. (2012). An approach for modeling architectural design rules in UML and its application to embedded software. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *21*(2), 10.

Merriam, S. B. (1998). *Qualitative Research and Case Study Applications in Education. Revised and Expanded from" Case Study Research in Education."*. Jossey-Bass Publishers, 350 Sansome St, San Francisco, CA 94104.

Mielicki, M. K., Kacinik, N. A., & Wiley, J. (2017). Bilingualism and symbolic abstraction: Implications for algebra learning. *Learning and Instruction*, *49*, 242-250.

Miles, M. B., & Huberman, A. M. (1994). *Qualitative data analysis: An expanded sourcebook*. sage.

Moody, D., & van Hillegersberg, J. (2008, September). Evaluating the visual syntax of UML: An analysis of the cognitive effectiveness of the UML family of diagrams. In *International Conference on Software Language Engineering* (pp. 16-34). Springer, Berlin, Heidelberg.

Morgan, C. (1988). Procedures, parameters, and abstraction: Separate concerns. In *On the Refinement Calculus* (pp. 47-58). Springer, London.

Morrison, D., & Collins, A. (1995). Epistemic fluency and constructivist learning environments. *Educational Technology, 35*(5), 39-45.

Morse, J. M. (1994). Designing funded qualitative research.

Narayanan, N. H., & Hegarty, M. (1998). On designing comprehensible interactive hypermedia manuals. *International journal of human-computer studies, 48*(2), 267-301.

Nguyen, D., & Wong, S. (2001). OOP in introductory CS: Better students through abstraction. In *Proceedings of the Fifth Workshop on Pedagogies and Tools for Assimilating Object-Oriented Concepts*.

Or-Bach, R., & Lavy, I. (2004). Cognitive activities of abstraction in object orientation: an empirical study. *ACM SIGCSE Bulletin, 36*(2), 82-86.

Ordonez, C. (2003, June). Clustering binary data streams with k-means. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery* (pp. 12-19).

Peneva, J., Ivanov, S., & Tuparov, G. (2006). Utilization of UML in Bulgarian SME–possible training strategies. In *Int. Conf. on Computer Systems and Technologies–CompSysTech*

Penner, D. E. (2000). Explaining systems: Investigating middle school students' understanding of emergent phenomena. *Journal of Research in Science Teaching: The Official Journal of the National Association for Research in Science Teaching, 37*(8), 784-806.

Perkins, D. N., & Grotzer, T. A. (2000). Models and Moves: Focusing on Dimensions of Causal Complexity To Achieve Deeper Scientific Understanding.

Perrenet, J. C. (2010). Levels of thinking in computer science: Development in bachelor students' conceptualization of algorithm. *Education and Information technologies, 15*(2), 87-107.

Queralt, A., & Teniente, E. (2012). Verification and validation of UML conceptual schemas with OCL constraints. *ACM Transactions on Software Engineering and Methodology (TOSEM), 21*(2), 13.

Reigeluth, C., & Stein, R. (1983). Elaboration theory. *Instructional-design theories and models: An overview of their current status (1983)*, 335-381.

Reigeluth, C. M. (Ed.). (2018). *Instructional theories in action: Lessons illustrating selected theories and models*. Routledge.

Regnell, B., Kimbler, K., & Wesslen, A. (1995, March). Improving the use case driven approach to requirements engineering. In *Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95)* (pp. 40-47). IEEE.

Rico, D. F. & Sayani, H. H. (2009). Use of Agile Methods in Software Engineering Education. *Agile Conference, 2009. AGILE'09*, 174-179.

Riegelman, R. K. (2016). The STIRS framework and integrative liberal education. *Peer Review*, *18*(4), 8.

Rijke, W. J., Bollen, L., Eysink, T. H., & Tolboom, J. L. (2018). Computational Thinking in Primary School: An Examination of Abstraction and Decomposition in Different Age Groups. *Informatics in education*, *17*(1), 77.

Rising, L., & Janoff, N. S. (2000). The Scrum software development process for small teams. *IEEE software*, *17*(4), 26-32.

Robal, T., Viies, V., & Kruus, M. (2002). The Rational Unified Process with the" 4+ 1" View Model of Software Architecture-a Way for Modeling Web Applications. In *BalticDB&IS* (pp. 119-132).

Rothschild, A. S., Dietrich, L., Ball, M. J., Wurtz, H., Farish-Hunt, H., & Cortes-Comerer, N. (2005). Leveraging systems thinking to design patient-centered clinical documentation systems. *International Journal of Medical Informatics*, *74*(5), 395-398.

Sandoval, W. (2014). Conjecture mapping: An approach to systematic educational design research. *Journal of the learning sciences*, *23*(1), 18-36.

Sangwan, R., Neill, C., Bass, M., & El Houda, Z. (2008). Integrating a software architecture-centric method into object-oriented analysis and design. *Journal of Systems and Software*, *81*(5), 727-746.

Santos, N., Fernandes, J. M., Carvalho, M. S., Silva, P. V., Fernandes, F. A., Rebelo, M. P., Barbosa, D., Maia, P., Couto, M., & Machado, R. J. (2016, July). Using Scrum together with UML models: a collaborative university-industry R&D software project. In *International Conference on Computational Science and its Applications* (pp. 480-495). Springer, Cham.

Schwaber, K. (1997). Scrum development process. In *Business object design and implementation* (pp. 117-134). Springer, London.

Senge, P. M. (1990). The art and practice of the learning organization.

Sevaldson, B. (2011). GIGA-Mapping: Visualisation for complexity and systems thinking in design. *Nordes*, (4).

Sherry, L., & Trigg, M. (1996). Epistemic forms and epistemic games. *Educational technology*, 38-44.

Shimoda, T. A., & Borge, M. (2016). The Web of Inquiry: Computer support for playing epistemic games. *International Journal of Information and Education Technology*, *6*(8), 607.

Shukla, A. & Williams, L. (2002). Adapting Extreme Programming for a Core Software Engineering Course. *Proceedings from the 15th Conference on Software Engineering Education and Training (CSEE&T 2002)*, Covington, KY.

Siau, K., & Rossi, M. (2011). Evaluation techniques for systems analysis and design modelling methods – a review and comparative analysis. *Information Systems Journal, 3*(21), 249-268.

Sprague, P., & Schahczenski, C. (2002). Abstraction the key to CS1. *Journal of Computing Sciences in Colleges*, *17*(3), 211-218.

Stake, R.E. (2006). *Multiple case study analysis.* The Guilford Press, NY: Guilford Publications Inc.

Stange, K. C., Crabtree, B. F., & Miller, W. L. (2006). Publishing multimethod research.

Stearman, J. (2000). Systems thinking and modeling for a complex world. *Irwin McGraw-Hill, Boston.*

Sutherland, J., & Schwaber, K. (2007). The Scrum Papers. *Nuts, Bolts and Origins of an Agile Process*.

Tall, D., & Thomas, M. O. J. (Eds.). (2002). *Intelligence, learning and understanding in mathematics: A tribute to Richard Skemp*. Post Pressed.

Tamai, T. (2005). How to teach software modeling. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* (pp. 609-610). IEEE.

Terrell, S. R. (2012). Mixed-methods research methodologies. *The qualitative report*, *17*(1), 254-280.

Thinsungnoena, T., Kaoungkub, N., Durongdumronchaib, P., Kerdprasopb, K., & Kerdprasopb, N. (2015). The clustering validity with silhouette and sum of squared errors. *learning*, *3*(7).

Tirkkonen-Condit, S. (1990). A Think-Aloud Protocol Study. In *Learning, Keeping, and Using Language: Selected Papers from the 8th World Congress of Applied Linguistics, Sydney, 16-21 August 1987* (Vol. 1, p. 381). John Benjamins Publishing Company.

Ulrich, W. (1994). Can we secure future-responsive management through systems thinking and design?. *Interfaces*, *24*(4), 26-37.

Umphress, D. A., Hendrix, T. D., & Cross, J. H. (2002). Software Process in the Classroom: The Capstone Project Experience. *IEEE Software*, 19(5), 78-81.

Unkelos-Shpigel, N., Sheidin, J., & Kupfer, M. (2019). Climb Your Way to the Model: Teaching UML to Software Engineering Students. In *International Conference on Advanced Information Systems Engineering* (pp. 40-46). Springer, Cham.

Vattam, S. S., Goel, A. K., Rugaber, S., Hmelo-Silver, C. E., Jordan, R., Gray, S., & Sinha, S. (2011). Understanding complex natural systems by articulating structure-behavior-function models. *Journal of Educational Technology & Society*, *14*(1), 66-81.

Vidich, A. J., & Lyman, S. M. (2000). Qualitative methods: Their history in sociology and anthropology. *Handbook of qualitative research*, *2*, 37-84.

Von Bertalanffy, L. (1972). The history and status of general systems theory. *Academy of management journal*, *15*(4), 407-426.

Walker, H. M. (1996). *Abstract data types: specifications, implementations, and applications*. Jones & Bartlett Learning.

Wand, Y., & Weber, R. (1993). On the ontological expressiveness of information systems analysis and design grammars. *Information Systems Journal*, *3*(4), 217-237.

Wang, Q., Koval, J. J., Mills, C. A., & Lee, K. I. D. (2007). Determination of the selection statistics and best significance level in backward stepwise logistic regression. *Communications in Statistics-Simulation and Computation*, *37*(1), 62-72.

Wang, H., Huo, D., Huang, J., Xu, Y., Yan, L., Sun, W., & Li, X. (2010, July). An approach for improving K-means algorithm on market segmentation. In *2010 International Conference on System Science and Engineering* (pp. 368-372). IEEE.

Whitten, Jeffrey L & Bentley, Lonnie D (2007). *Systems analysis and design methods (7th ed)*. McGraw-Hill/Irwin, Boston

Wilensky, U., & Resnick, M. (1999). Thinking in levels: A dynamic systems approach to making sense of the world. *Journal of Science Education and technology*, *8*(1), 3-19.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717-3725.

Xu, R., & Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on neural networks*, *16*(3), 645-678.

Yellamraju, T., & Boutin, M. (2018). Pattern dependence detection using n-TARP clustering. *arXiv preprint arXiv:1806.05297*.

Yeo, K. T. (1993). Systems thinking and project management—time to reunite. *International journal of project management*, *11*(2), 111-117.

Yuan, C., & Yang, H. (2019). Research on K-value selection method of K-means clustering algorithm. *J—Multidisciplinary Scientific Journal*, *2*(2), 226-235.

Zehetmeier, D., Böttcher, A., Brüggemann-Klein, A., & Thurner, V. (2019). Defining the Competence of Abstract Thinking and Evaluating CS-Students' Level of Abstraction. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*.

Zhu, L., Ma, B., & Zhao, X. (2010). Clustering validity analysis based on silhouette coefficient [J]. *Journal of Computer Applications*, *30*(2), 139-141.