



1-1-2019

Evolution Of Object-Oriented Methods From The Reverse Engineering Of Programming Code

Pann Ajjimaporn

Follow this and additional works at: <https://commons.und.edu/theses>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Ajjimaporn, Pann, "Evolution Of Object-Oriented Methods From The Reverse Engineering Of Programming Code" (2019). *Theses and Dissertations*. 2236.

<https://commons.und.edu/theses/2236>

This Thesis is brought to you for free and open access by the Theses, Dissertations, and Senior Projects at UND Scholarly Commons. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of UND Scholarly Commons. For more information, please contact zeinebyousif@library.und.edu.

Evolution of Object-Oriented Methods from the Reverse Engineering of Programming Code

BY

Pann Ajjimaporn
Bachelor of Science, University of North Dakota, 2016
Master of Science, University of North Dakota, 2019

A Thesis

Submitted to the Graduate Faculty
Of the
University of North Dakota
in partial fulfillment of the requirements

for the degree of
Master of Science
in Computer Science in the Graduate College of the
Department of Computer Science, University of North Dakota, Grand Forks, U.S.A.
emanuel.grant@engr.und.edu, pann.ajjimaporn@und.edu


Grand Forks, North Dakota

May 2019

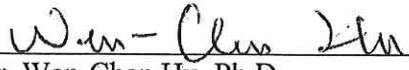
Adviser:

Professor Emanuel S. Grant

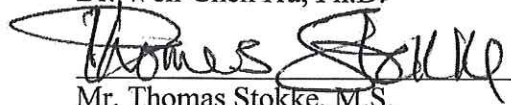
This thesis, submitted by Pann Ajjimaporn in partial fulfillment of the requirements for the Degree of Master of Science from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done and is hereby approved.



Dr. Emanuel S. Grant, Ph.D.



Dr. Wen-Chen Hu, Ph.D.



Mr. Thomas Stokke, M.S.

This thesis is being submitted by the appointed advisory committee as having met all of the requirements of the School of Graduate Studies at the University of North Dakota and is hereby approved.



Dean of the School of Graduate Studies

4/26/19

Date

PERMISSION

Title Evolution of Object-Oriented methods from the Reverse Engineering
 of a Programming code

Department Computer Science

Degree Master of Science

In presenting this thesis in partial fulfillment of the requirements for a graduate degree from the University of North Dakota, I agree that the library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my thesis work or, in her absence, by the Chairperson of the department or the dean of the School of Graduate Studies. It is understood that any copying or publication or other use of this thesis or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of North Dakota in any scholarly use which may be made of any material in my thesis.

Pann Ajjimapun
11/12/2018

ACKNOWLEDGEMENTS

I wish to express my indebtedness for all the work and contributions of my thesis advisor and my supervisor, Professor Emanuel Grant of the school of Computer Science at the University of North Dakota, who made this work possible. The door to Prof. Grant office was always open whenever I needed advice and/or when I got stuck during the time with my thesis.

The thesis has also benefited from comments and suggestions made by Brendon, and my best friend Karl, who has read through the manuscript and helped me at the very last second of this thesis completion.

I take this opportunity to give special thanks to my family, starting with my mom, Amornpan, and my dad, Prasit for their encouragement and love throughout the years. On that note, I'd like to thank my sister, Panan, for her overwhelming support and care. This thesis would not have been possible without them. Thank you from the bottom of my heart.

Lastly, I'd like to thank every faculty member and my friends in the Department of Computer Science, who supported me and provided an excellent learning environment during my time in college.

Author

Pann Ajjimaporn

To the single most important thing in my life,

My family.

ABSTRACT

Many software development projects fail because of their inability to deliver the product in a timely and cost-effective manner, i.e. the software crisis. In a commercial airline company, a safety-critical system for preventing a “single-point of failure” needs to be developed and certified. To meet the project deadline an Agile approach was used in developing the first portion of the system. An appropriate software development methodology, i.e. reverse-engineering was then applied to the software system to develop method that would be used in system maintenance and evaluation.

The goals of this research study were to develop a segment of one of the UML activity diagrams as a purposeful and systematic methodology for conducting reverse-engineering on complete safety critical systems of the airline system. This was done to capture very high and very low level designs of software engineering and to verify and validate the source code.

The UML activity diagram was developed from a source code of an aircraft-gate assignment system. This was done in order to capture very low-level details of the program code, from which the model was reversed engineered. The diagram was made to represent the entire source code, by going through and analyzing the source code line-by-line. Whenever there is a condition in the source code, the diagram branches out and interacts with other activities. To directly see the flow of the program, the directional arrows in the diagram were assigned.

With the data flow of the source code being represented in the visual format of the UML activity diagram, the interaction of each component can be easily understood and identified. The user can see the information that goes into each method, and what each method required. Once the user understands the flow of data within the program, the user can validate and verify that the software was developed with correct methods.

As the UML activity diagram represented the pseudo code of the program in a graphic way, it should be a good candidate to be used as a tool to help in reverse-engineering safety critical systems of an airline system. The UML activity diagram should be able to represent all aspects required in the deconstruction phase of the reverse-engineering methodology. With the source code in the graphic diagram form, it should be much easier to identify the structure, functions and determine how each aspect of the program interact with each other so that the activity diagram can be used in a formal methodology for reverse-engineering.

Keywords: UML Activity Diagram, source code, the reverse-engineering methodology

TABLE OF CONTENTS

1. Introduction	1
1.1 Problem Description	2
1.2 Motivation	3
1.3 Objectives	4
1.4 Scope of Work	4
1.5 Description of Thesis / Report Organization	5
2. Backgrounds	6
2.1 UML Notations	6
2.2 Definition of Reverse Engineering and Methodologies	8
2.3 History of Reverse Engineering and Methodologies	9
2.3.1 Reverse Engineering	9
2.3.2 Methodologies	11
2.4 The Software Methodology Used	14
2.5 Contrast Academia and Industries	18
2.5.1 Reverse Engineering in Academia	19
2.5.2 Reverse Engineering in Industry	20
2.6 UML Activity Diagram	21
2.6 Support Tools	23
3. Application of Methodology	26
3.1 Source Code Background	26
3.2 Modified Methodology	26
3.3 Project Implementation	28
4. Results	36
4.1 Types of Model Developed	36
4.2 Description of Activity Diagram	39
5. Discussion and Conclusions	42
5.1 Lesson Learned	42
5.2 Difficulties Using UML Activity Diagram	42
6. Conclusions	44

6.1 Acceptance of Work	44
6.2 Experience in Conducting Reverse Engineering	44
6.3 Future of Reverse Engineering	46
References	47

LIST OF FIGURES

Figure	Page
Figure 1.	Evolution of object-oriented methods and notations 1980s – mid 2000s7
Figure 2.	A Brief History of Software Development Methodologies 13
Figure 3.	DO-168C UML Package-Level Representation. 15
Figure 4.	DO-178C Software Planning Process 4.0 UML Use Case Diagram Representation 16
Figure 5.	UML Activity Diagram Representation of DO-178C Compliant Model-Driven Methodology. 18
Figure 6.	University of North Dakota Undergrad Requirements for Major in Computer Science 2018-2019. 19
Figure 7.	Example of Activity Diagram. 22
Figure 8.	StarUML Windows Interface 23
Figure 9.	bubbl.us Web-Interface 25
Figure 10.	Reverse-Engineering Modified Model-Driven 26
Figure 11.	UML activity diagram of Aircraft-Gate Assignment System. 29
Figure 12.	Code Snippet from UML Activity Diagram. 30
Figure 13.	Example of a condition in UML Activity Diagram 31
Figure 14.	Example of a conditional statement with only one activity generated 32
Figure 15.	Example of a loop in the UML Activity Diagram 33
Figure 16.	Example of a switch...case statement in UML Activity Diagram 34
Figure 17.	expanded diagram of the Evolution of object-oriented methods and notations 1980-mid 2000s in figure 1 up to date as of 2018. 36
Figure 18.	Example of a full-size diagram 38
Figure 19.	Reverse Engineering + UML Diagram 41
Figure 20.	Example code used in the Computer Science II 45

1. INTRODUCTION

Reverse engineering (NPD Solutions, 2016) is a process of purposefully and systematically deconstructing an object that can be used on a software system to reveal its design and purpose. In addition, the process of reverse-engineering also refers to the examination of the system, but not the reconstruction or modification of the original system. The examination of reverse-engineering of a software system has become increasingly important, not only in the industrial field but also the academic field. For the industrial field, the reverse-engineering of a software system was required to both innovate and debug the software (Innovation Enterprise Channels, 2019). This is required as complex programs are made up from many smaller components that do a certain specific task. In an academic and some industrial setting, reverse software engineering is used to study and reuse old programs in the development of new systems with enhanced functionality.

In an academic setting, reverse engineering acts as a stepping stones to approach real-world conditions of the software engineering (Bothe, 2001). Reverse engineering of a software system can also be explored in a more practical way as a method for students to deconstruct an example program for usage in their own program.

Reverse engineering (Engard, 2016) is a very important tool for software developer as it is a technique used whenever someone wants to understand a process and its functionality. Reverse engineering in computer science is used mainly in product and process improvement, cybersecurity field, and intelligence and espionage.

In the last decade, there have been many methodologies and models for software development that are still in use. Some methodologies improved on the ideas of past formal methods and steps to follow in a software engineering process. There are many formal methodologies and notations that work on specific phases of software engineering, each with their own pros and cons, but there are currently no single formally established

methodologies to reverse engineer a software system. Because of this, an opportunity exists for reverse-engineering methodologies to be defined.

Systems modeling is the concept of using models to conceptualize and construct systems in IT environment (sebokwiki, 2018). The fundamental concept in system modeling is abstraction, which draws focus from unimportant details to essential characteristic. System modeling lower the size and complexity of the system and make it easier in order to be computationally and intellectually tractable. There are many conceptual models used in the system modeling, each with their own focuses. One modeling concept that stands out from the rest is the Unified Modeling Language as it provides detailed modeling, well-defined process and is supported by powerful tools (Visual Paradigm, 2018).

A Unified Modeling Language (UML) activity diagram presents a path of execution through the program and displays a representation of the code. Since the activity diagram is a graphical representation of the program pseudo code, it is a good candidate as a tool to be used in reverse-engineering. Each phase of the reverse-engineering modified model-driven methodology can be used to measure the UML activity diagram as an adequate tool for a formal methodology of reverse-engineering.

The UML Activity Diagram should be able to represent all aspects required in the deconstruction phase of the reverse-engineering methodology. With the source code in the graphic diagram form, it should be much easier to identify the structure, functions and determine how each component of the program interact with each other. The Activity diagram then can be used in a formal methodology for reverse-engineering.

1.1 Problem Description

In the year 2018, there are various methodologies and models for software development that are actively being applied in every level of the professional software development. Yet, there currently is no formally established methodology to reverse engineer a software system. There is, however, a formal methodology to reverse engineer

a program code (Gannod, 1994). The problem is that a large amount of the methodology cannot be applied to software development due to the difference in the structure and method to study an object.

In academia, reverse-engineering for software is a known practice, but there is no real formal exploration into the subject. There have been attempts to explore the reverse-engineering as a learning process for engineering (Barone & Mandredi, 2007). The attempt was made with the computer-aided design (CAD) and analysis software in mind, with the emphasis on the general reverse-engineering of the object as the learning subject. Although there were some attempts to use reverse-engineering as a learning experience, none of these were related to software engineering. The majority of computer science learning processes involve learning by example. Reverse engineering computer programs is an assumed part of this process but is not explored explicitly.

In a commercial airline company, a safety-critical system for preventing a “single-point of failure” needs to be developed and certified through compliance with DO-178C. The DO-178C, Software Consideration in Airborne Systems and Equipment Certification, is the document specification to approve software-based aerospace systems by verify and validate the software concepts of high-level requirements and low-level requirements. The DO-178C may uses agile approach in developing the system so a way to produce a documentation of the system is needed, which leads to the need to reverse engineer the code. This then necessitate the identification of a suitable reverse engineering methodology in which will then be implemented on the code.

1.2 Motivation

The use of reverse-engineering in academia to learn computer programming was explored from both a student perspective, as a master’s degree-seeking student of computer science at the University of North Dakota, and from the teaching perspective, as a teaching assistant to class “Computer Science II”. In a computer science class, students are expected to learn how to program by using an example program as a guide.

This fits the description of reverse-engineering: deconstructing to analyze its parts, with the objective of creating something from the parts. The use of reverse-engineering and its effect can be clearly seen in the lecture and lab portion of the class, where the lectures taught the student about the information behind the commands and the lab portion took those concepts and put it to practical use. The example of this will be demonstrated in chapter 5.3 of this thesis. There is an overlap experience between reverse-engineering a software system and the way to teach programming to the student which makes studying the reverse-engineering in the field of software system an exciting task.

There is currently no single formal Reverse Engineering Methodology for software engineering, either for academic use or industrial use. This could mean that the use of reverse-engineering in academia has not yet been fully understood, or at least not enough to where reverse-engineering can be applied consistently to teach students about programming in computer science. The other possibility is that the industrial use of reverse software engineering is something that should not be done. Even though there are methodologies for reverse-engineering in general, there are none that apply specifically to software engineering.

1.3 Objectives

The goals of this research study were to develop a segment from the UML activity diagrams as a purposeful and systematic methodology for conducting reverse-engineering on a safety critical system of an airline system and to develop an updated model of the methodologies. Validation of the objective was the development and acceptance of the models of the methodology as a true representation of the system.

1.4 Scope of Work

In this thesis, the type of UML model will be focused on is the UML activity diagram. The diagram is used in demonstrating the application of reverse-engineering in software development as it is a visual representation of the software system dynamic execution.

The domain of the problem will only focus on the reverse-engineering in a software system, specifically in the computer science field and not in other engineering fields. The goal is to underline the problem regarding the awareness and practicality of the application of reverse-engineering in academia, for both teaching and learning.

The UML Activity diagram was developed with access to the original source code developer so that there would be no misunderstandings of each method in the source code. The UML Activity Diagram itself, however, was not written with the source code developer's explanation of the purpose of the methods, but rather generated directly from the source code itself.

Even though the diagram was not made specifically to fit the DO-178C, it was made as a traditional UML activity diagram and then later verify with the DO-178C requirements. In addition, during the construction phase of the diagram, there was neither verification nor validation process done directly with the diagram. However, both processes were considered and applied during the application process of DO-178C.

1.5 Description of Thesis / Report Organization

Chapter 2 explains most of the core knowledge required to fully understand this work along with the tools used in creating the work for this thesis and any related work. Chapter 3 contains the description of the research done in defining the reverse engineering methodology used in this work. Chapter 4 shows the content of the work done during this study such as application of the methodology and a detailed description of what was performed during this study. Chapter 5 presents a discussion of what was achieved on this project and lessons learned. Chapter 6 is the conclusion of the success of this work and ideas about the future of this topic.

2. BACKGROUNDS

In order to fully understand this work, a base knowledge regarding the definition, history and the tools must first be established. This chapter contains the information on the evolution of the model notations and its link to the reverse engineering, a core methodology used in this work as well as two of the tools used in the creation of models of this work.

2.1 UML Notations

In the last decade, there has been intense research activities in software development methodologies. A software development methodology is a framework to structure, plan, and control the process of developing an information system (Association of Modern Technologies Professionals, 2018). There are many methodologies that are currently used in the software engineering industry such as Agile Software Development (Agile Alliance, 2019), Rapid Application Development (RAD) (Anderson, 2019), Rational Unified Process (RUP) (Powell-Morse, 2016), Waterfall (TutorialsPoint, 2018), and Spiral (TutorialsPoint, 2019).

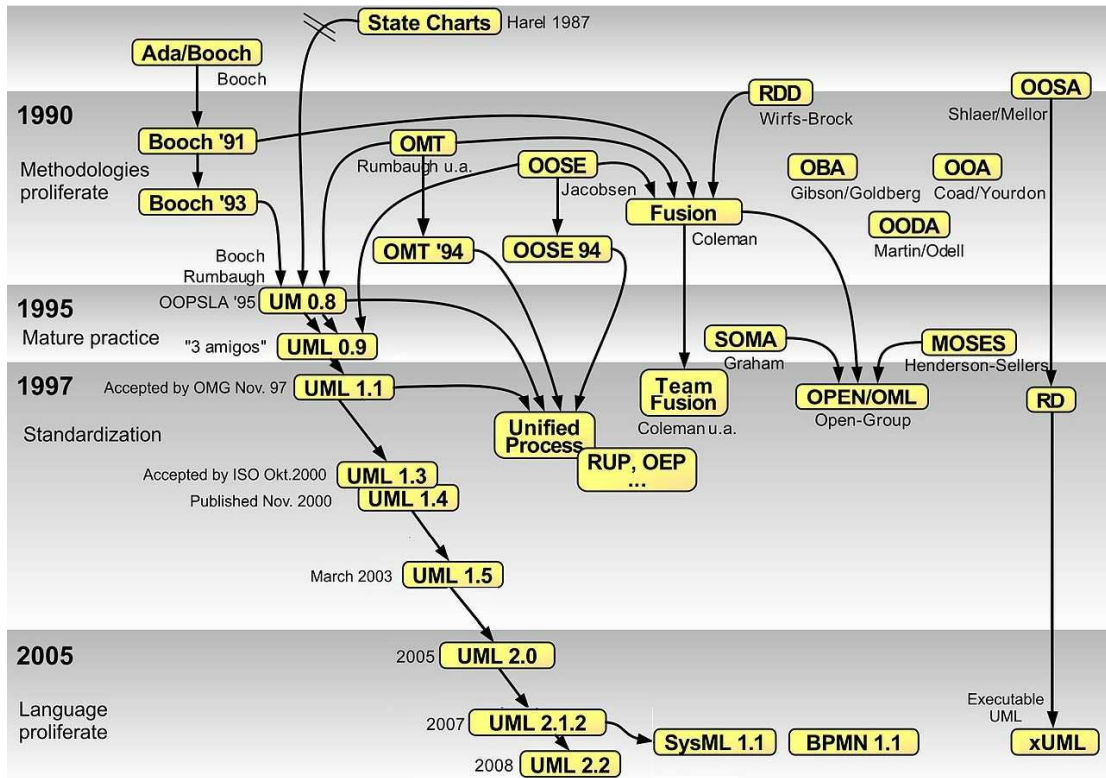


Figure 1. Evolution of object-oriented methods and notations 1980s – mid 2000s¹

Figure 1 shows the evolution of object-oriented methods and notations from 1980 through the mid-2000s, but some information regarding years and whether the notations are still in use are missing. The chart also shows how the Unified Modeling Language has gone through many iterations throughout the years, with the latest version from the chart being UML 2.2 in 2008. The chart also reveals the influence of UML being integrated into another form of models, which transformed RD into Executable UML or xUML (Zockoll, Scheithauer, & Dekker, 2009).

¹ “History of Object Oriented Modeling languages”, Digital Image, <https://en.wikipedia.org/wiki/Unified_Modeling_Language#/media/File:OO_Modeling_languages_history.jpg>

A methodology is defined as the theoretical analysis of methods applied to a field of study (Merriam-Webster, 2018). In Software Engineering, it refers to the process of dividing the software development work into separate phases in order to help improve the design of the product and to reduce the complexity in the management process. For software engineering specifically, it is known as the “software development life cycle”. This means that a single software will have to go through many “cycles” of the process from the beginning to the end of the program’s lifespan. (Centers for Medicare & Medicaid Services, 2008)

The chart, however, cuts short the information regarding the Unified Process, only including RUP and OEP. Even though the Unified Process evolved and grew alongside UML in the beginning, there is no update between 1997 and 2005. The chart also does not specify when and how far the Systems Modeling Language (SysML) and the Business Process Model and Notation (BPMN) evolved since its separation from UML 2.1.2. Part of the work of this thesis is to extend this model to the current time in order to find the most efficient models to represent the system.

2.2 Definition of Reverse Engineering and Methodologies

Reverse Engineering is a process of purposefully and systematically deconstructing an object to reveal the design and the purpose of the specific parts inside. In the field of software engineering, Institute of Electrical and Electronics Engineers (IEEE) defined reverse-engineering as “the process of analyzing a subject system to identify the system’s components and their interrelationships and to create representations of the system in another form at a higher level of abstraction” (Beckmann, Vogelsang, & Reuter, 2017). The process of reverse-engineering only refers to the examination of the system, and not the reconstruction or modification of the original system. As of 2018, there is no formal methodology of reverse-engineering a software system (Asif, 2003). The definition of Reverse Engineer by Merriam-Webster dictionary is “to disassemble and examine or analyze in detail (a product or device) to discover the concepts involved in manufacture

usually in order to produce something similar” (Merriam-Webster, 2018). Though the definition from the dictionary states that the objective for reverse-engineering a product is to produce something similar, in fields where the product consists of many complex parts working together, discovering the concepts of each specific part can produce another product that had very little in common with the original.

2.3 History of Reverse Engineering and Methodologies

Reverse Engineering has historically been used in the military for strategic advantages such as knowing the weak points of a tank or decrypting the ciphers that were widely used in wartime. (New World Encyclopedia, 2018) The reverse-engineering process is not meant to modify the object in any way, but rather to explore its parts to better understand how the object functions or the procedure of the object’s design and production. In any case, reverse-engineering may be used in the procedure of improving an existing product and may be used more than once in the improvement of the product cycle. The usage of reverse-engineering in software development may simply be to document the code for future changes or improvements. The process of reverse-engineering may help reduce the overall cost of software maintenance by reducing the time required to understand the source code. Understanding the source code may be one of, if not the most time-consuming process of the entire software maintenance cycle, especially if the source code’s original writer is not there to explain.

2.3.1 Reverse Engineering

Reverse Engineering can also be categorized into different motivations. Each reason for reverse-engineering requires different tasks to obtain different items from the same product. The reasons for reverse-engineering according to (New World Encyclopedia, 2018) are lost documentation where the documentation of the product has been lost or didn’t exist to begin with. Many products that are in use in today are a constant upgrade from the products of the past, which means that sometimes the person who originally

designed the parts that are to be modified to fit the new product is no longer available. This means that the only way to fully understand how that specific part works is to disassemble it and write the documentation so that it can be used in the future with a better understanding.

The other reason for reverse-engineering is product analysis. When a product is completed it is difficult to understand how it works since the product usually functions best when it is fully intact the way it was intended to be. This means that it is difficult to see what components it consists of without breaking the product or voiding the warranty. Some of the components of the product may break or stop functioning the moment the product is dissembled, so it is important to know the exact function and materials of each component to estimate costs. A product may look cheap and simple from the outside, as many products have user interface in mind, and to minimize the complexity of the user's to interaction with the product and to maximize the usability of the product, the interactable parts are usually kept simple and bland. This makes it difficult to estimate the cost of the more complex interior of the product.

An important reason to reverse-engineer a product is to identify the potential patent infringement which many products may contain. In a recent example, a software company "Bethesda" sued "Warner Bros" for a copyright infringement of Bethesda's game "Fallout Shelter" by "Westworld" for mobile. "Bethesda sued Westworld game developer Behavior Interactive and their publisher Warner Bros, claiming Behavior stole its design, artwork, and coding, and used them in the mobile app" (Hood, 2018). With the video game industry becoming increasingly large and important in modern society, there are many cases where video games can have many similarities between each other, with very few small differences in the art style and gameplay. It is very difficult to determine if video games infringe on each other's copyrights. In the example provided with Bethesda suing Warner Bros, the two games are extremely similar in gameplay experience, but with a different art style and overall game design, it is difficult to determine if Westworld had infringed on Fallout Shelter or not. But as the original developer of the game,

Bethesda saw that Westworld had bugs that were evident in the early development of Fallout Shelter, which reinforced the claim that Westworld had at least part of Fallout Shelter's code.

2.3.2 Methodologies

The history of software development methodologies is as old as the word “framework” in software development (Oleksandrova, 2018). A brief history of software development methodologies is shown in figure 2. Back in the 1950s, the software lifecycle and structured programming was considered the only methodology framework as software and the computer were mostly tied to each other. The main objective of the software development methodology in 1960 was “to develop large-scale functional business systems in an age of large-scale business conglomerates. Information systems activities revolved around heavy data processing and number crunching routines”.

In the 1960s, the first process model called the waterfall model was introduced. It is referred to as the linear-sequential life cycle model, which is simple to understand and use. It is composed of phases where each phase must be completed before the next one can start. The structure is like flowing down a waterfall (since you cannot go back up). Phases in the waterfall model are requirement analysis, system design, implementation, testing, deployment and maintenance (TutorialsPoint, 2018). The Iterative and Incremental model which was introduced later, in the year 1970, uses a similar model to waterfall but introduces iteration loops to the phases.

Figure 2(2) shows the Prototyping model introduced in the early 1980s. It was a system in which a prototype is built, tested and then reworked. This process is repeated until an acceptable prototype is achieved. After a fully developed prototype is complete, the complete system or product can be generated (Rouse, 2018). The Spiral model is one of the most important software development life cycle models for risk handling. The process of the loop is very similar to how the iterative and incremental model works, but with different phases, and the time spent in each loop is increasingly smaller. The phases

in this model include objective determination and identify alternative solutions, risk identify and resolve, development of the next product version, and review/plan for the next phase (Pal, 2018).

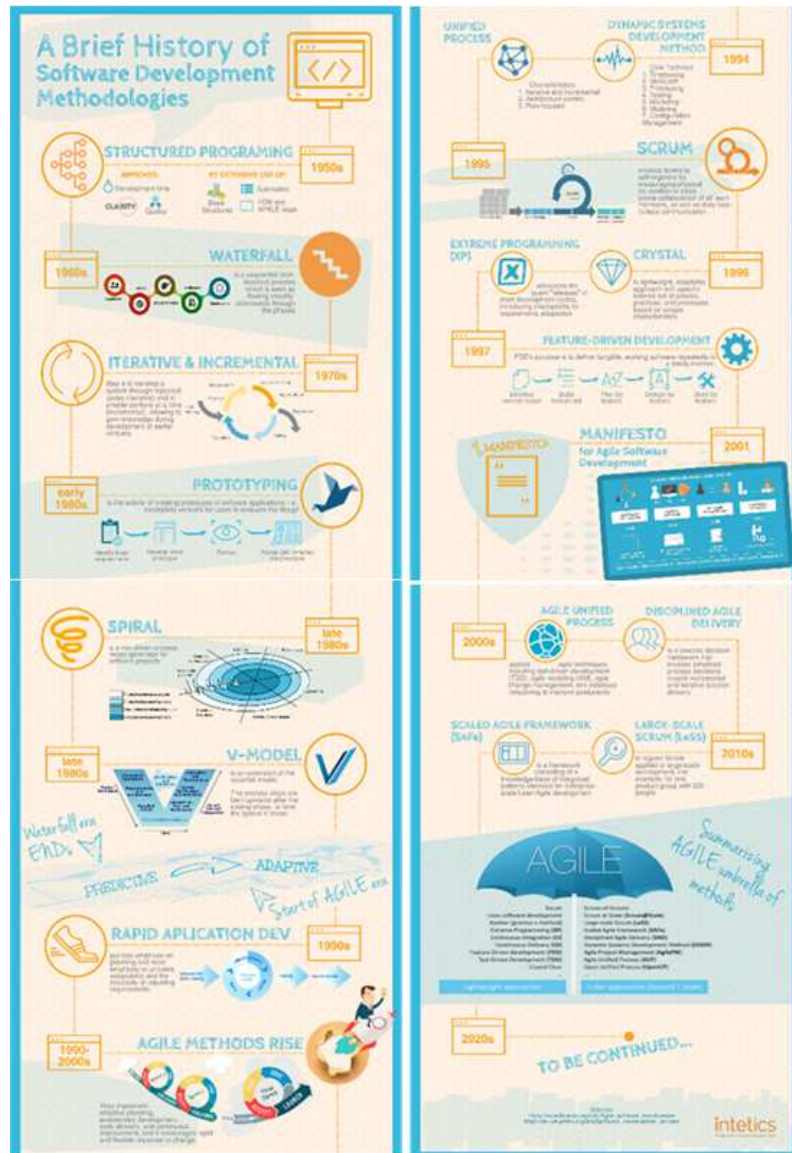


Figure 2. A Brief History of Software Development Methodologies; 2(1) from 1950s-early 1980s, 2(2) late 1980s-2000s, 2(3) from 1994-2001, 2(4) 2000s- 2010s, and summarizing the AGILE umbrella of methods.²

² “A Brief History of Software Development Methodologies”, digital image, viewed 11th November 2018, <<https://intetics.com/blog/a-brief-history-of-software-development-methodologies>>

Figure 2(3) shows V-Model development methodologies, which is very similar to the waterfall method that follows strict, step-by-step stages. The methodology has 2 main phases: project definition, and project test and integration. The “V” in the model represent the two phases and how they interact with each other. This model is the first to adopt verification and validation methods into its model (Powell-Morse, 2016). At this point at the end of 1980, the waterfall era ends, and agile era begins as the software development model changes from predictive to adaptive. During 1990, the rapid application development (RAD) model was introduced. This model is based on prototyping and iterative development with no specific planning involved (TutorialPoint, 2018).

There were many development methods generated from 1990 to the 2000s as shown in figure 2(4). Unified Process, an agile-based UML was born during this era along with SCRUM, Crystal and Extreme Programming. SCRUM in particular, was a popular software development technique using agile framework with an emphasis on software development (and not just for product development).

During the next 10 years, as shown in figure 2(5), the agile development technique had further branched into Rational Unified Process (RUP) and Object Engineering Process (OEP). RUP had later become Agile Unified Process (AUP). SCRUM in the other hand, had become a prominent figure in the software development industries as it expands to include Large-Scale SCRUM (LeSS) which differ itself from regular SCRUM by applied to a much larger scale development (over hundreds of people for LeSS compared to SCRUM that meant for 3 to 9 members) (Schwaber, 1993).

2.4 The Software Methodology Used

The software development methodology applied in this project came out of the academic program taught in the university and research work on the safety-critical system in general, and more specifically for avionics software systems (Grant & Ajjimaporn,

2018). The genesis of the methodology was on an unmanned aerial system (UAS) for monitoring the flight operations of unmanned aerial vehicles (UAVs) in unrestricted airspace. In order to conduct software development in UAS domain, the RTCA DO-178C specification was used as the definitive guideline (RTCA, 2011). The work with DO-178C was two-fold: firstly, the document was transformed from its textual representation to a graphical representation, in the UML notation. Figures 3, and 4 illustrate two of the models developed to represent the DO-178C specification. Figure 3 represents the DO-178C specification, software development methodology components as a UML package-level model. Each package of Figure 3 is decomposed into a set of UML use case diagrams, class diagrams, and activity diagrams. Figure 4 represents the DO-178C Software Planning Process (Section 4 of the DO178C specification) as a UML Use Case Diagram, wherein the user is the project development team. Figure 4 is one of the models contained in the Software Planning Process 4.0 of Figure 3 package-level model (Grant & Ajjimaporn, 2018).

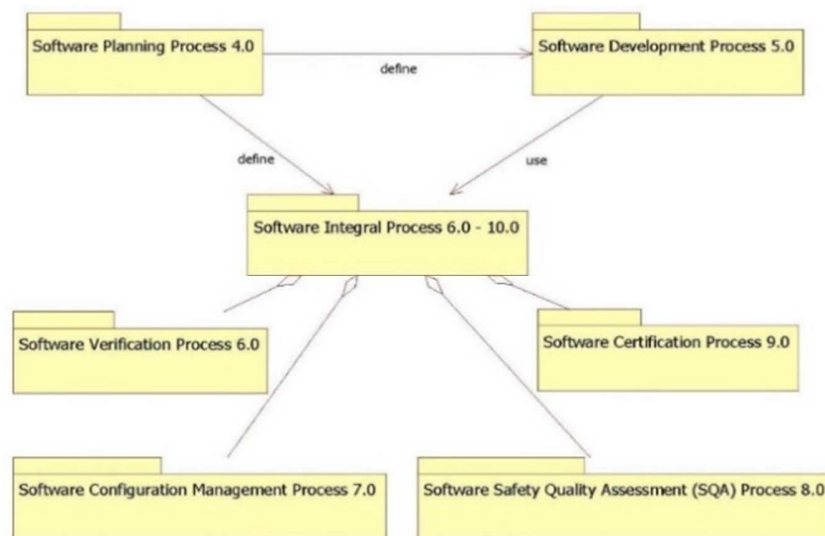


Figure 3. DO-168C UML Package-Level Representation.

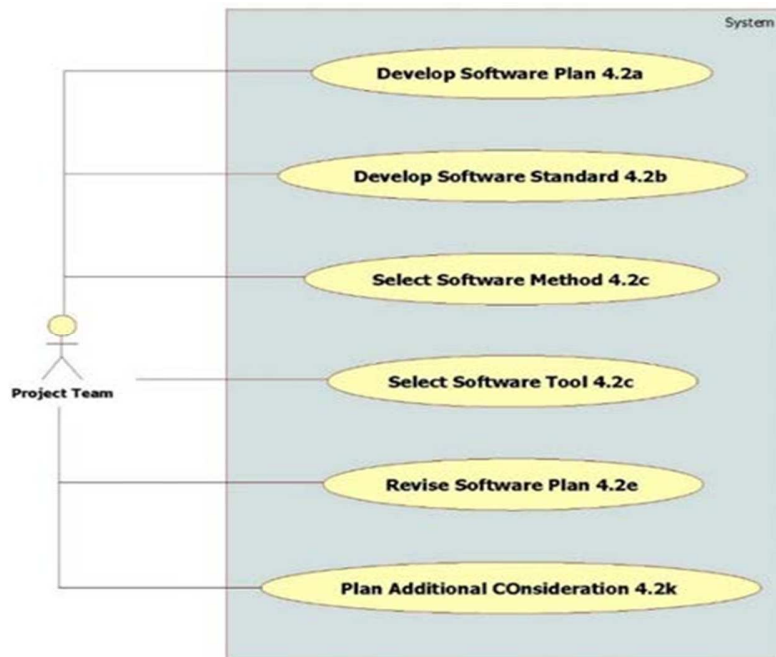


Figure 4. DO-178C Software Planning Process 4.0 UML Use Case Diagram Representation

The second area of work with the DO-178C specification is the definition of a model-driven software development methodology that incorporates and is compliant with the DO-178C requirements. This methodology is illustrated in Figure. 5 in the form of a UML activity diagram. Figure 5 is a UML activity diagram representation of the requirement-level activities contained in the Software Development Process 5.0 package of Figure 3. The activities of Figure 5 are mapped to the respective sections of the DO-178C document, by way of the DO-178C section number being listed in the activities of the model. Figure 5 captures the activities as specified in the DO-178C for the software requirements analysis and design phases; the software implementation (coding), testing, and deployment phases are represented in separate UML activity diagrams. The work reported on in this manuscript is limited to the scope of Figure 5. The UML models specified in Figure. 5 are specific to this instantiation of the methodology; in other

instantiations, other models may be used to satisfy the requirements of either the problem domain or the expertise of the development team.

A first task requirement of DO-178C Software Planning Process (4.0) is the determination of the software level of development. DO-178C specifies five (5) levels of criticality, designated Level-A through Level-E, with Level-A being the highest and Level E the lowest. Once the software level has been determined then DO-178C Software Development Process (5.0) and Software Integral process (6.0 – 10.0) specify the required set of activities and data elements necessary for certification of the system that is to be developed. The outputs of these activities are the Software Plan (4.2a), Software Standard (4.2b), Software Method (4.2c), and Software Tool (4.2c), as listed in Figure 4. There may be Additional Considerations, for the particular application domain (Grant & Ajjimaporn, 2018). This work focus on the creation of the UML Activity Diagram on the “Conduct High-Level Design” phase of the methodology which also covers the “Verify High-Level Design” phase as well.

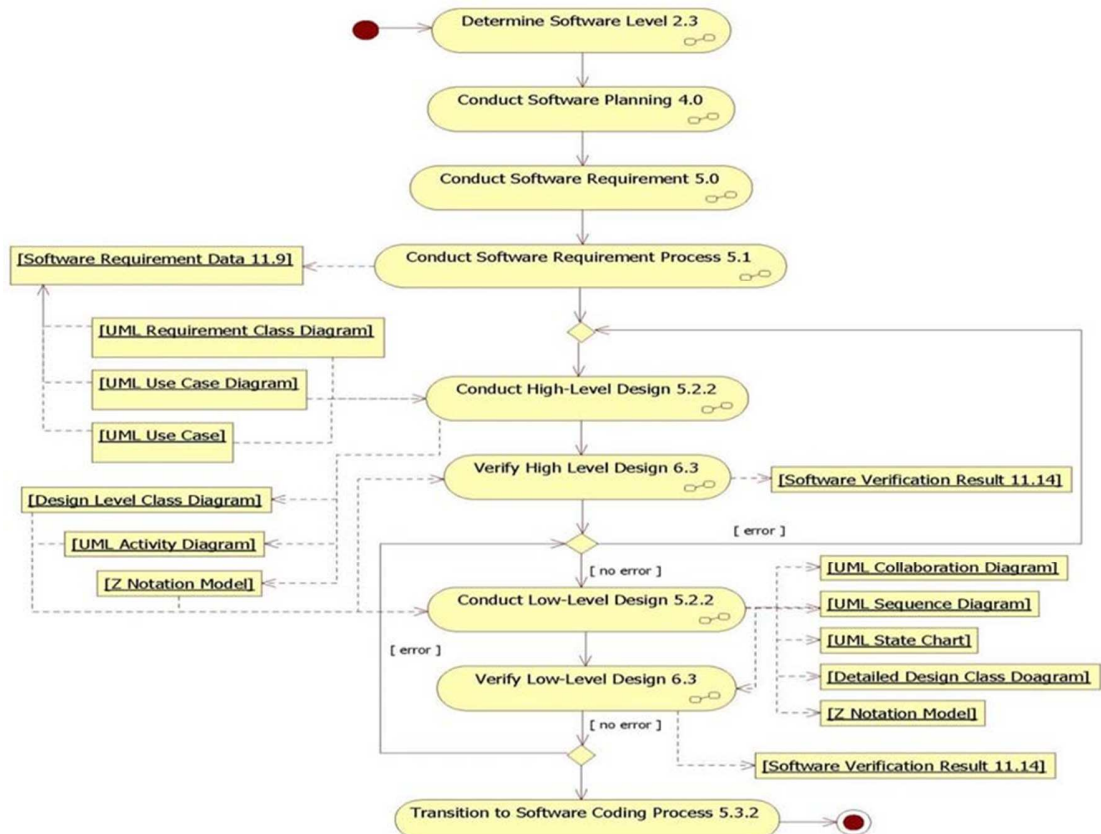


Figure 5. UML Activity Diagram of DO-178C Model-Driven Methodology.

2.5 Contrast Academia and Industries

The main purpose for the reverse-engineering of a software is to expand the information about the software development process, to fully understand the software functionality. With that being the main product of the reverse-engineering, there can be many other benefits stemming from understanding the function of said parts. The benefits of reverse-engineering can be categorized into two aspects: the academic aspect of the reverse software engineering and the industrial benefit from the reverse-engineering.

2.5.1 Reverse Engineering in Academia

For academia, reverse software engineering reuses the older programs to help develop new systems with enhanced functionality (Rashid, Salam, ShahSani, & Alam, 2013). The main concept for the usage of reverse software engineering in academia is “innovation”, specifically to build more complex and better programs by understanding the old existing programs. This concept is shown in many of the computer science classes at the University of North Dakota (UND). For example, an assignment for a computer science class “Data Engineering and Management” was to construct an Android application from Android Studio and integrate Android Google Maps to create a native mobile app. The students were then shown the examples of a few small programs including source codes, such as a basic app that uses GPS to obtain longitude and latitude data then display them to the user. The students were then shown a few more applications that implemented the Google Maps Android API using Android Studios. The students could then reverse engineer the example programs for specific parts and functionalities that they needed to complete the assignment. This example shows that the reverse software engineering in academia plays a very important role. It teaches the students about the topics of the class so that they can use the concepts that they learn to write programs.

CSCI 160	Computer Science I *	4
CSCI 161	Computer Science II *	4
CSCI 230	Systems Programming *	3
CSCI 242	Algorithms and Data Structures *	3
CSCI 289	Social Implications of Computer Technology	3
CSCI 363	User Interface Design	3
CSCI 365	Organization of Programming Languages *	3
CSCI 370	Computer Architecture *	4
CSCI 435	Formal Languages and Automata	3
CSCI 451	Operating Systems I	3
CSCI 492	Senior Project I	2
CSCI 493	Senior Project II	2
CSCI 494	Special Projects in Computer Science (Co-Req CSCI 493)	1
CSCI Electives **		12
Total Credits		50

Figure 6. UND Requirements for Undergraduates Majoring in Computer Science.

“In many USA universities, reverse-engineering is normally taught as an “add-on” to software development methodologies. The resulting situation is that graduates leave these software engineering programs with minimal knowledge about reverse-engineering then find themselves in a work environment where reverse-engineering is of paramount importance. Figure 6 shows the requirements for getting a bachelor of science degree in Computer Science from the University of North Dakota during the year 2018-2019. As shown from Figure 6, there was no emphasis on the reverse-engineering aspect of the computer science program, even though the idea of reverse-engineering were applied during the Computer Science I and Computer Science II classes (University of North Dakota, 2018). The experience of the faculty researchers on the project documented in this report, and from a prior project on the development of a UAS airworthiness system for monitoring UAVs operation in a restricted airspace, is that there needs to be a change in the pedagogical approach to teaching reverse-engineering.

The student researchers on this project and a prior project, in which reverse-engineering was also applied, expressed specific and strong opinions on the need to be taught formal approaches to reverse-engineering. Some of these student researchers had participated in internship programs at a variety of industrial organizations and had been exposed to reverse-engineering tasks. There was a unanimous conclusion that reverse-engineering is important to the software development activities in the real-world project and consequently, they think the process should be offered in courses on the same level as forwarding engineering topics. There was also a consensus among the student researchers that working with code at the start of the project was challenging and this challenge may be alleviated if they had a grounding in techniques to reconstruct the code.” (Grant & Ajjimaporn, 2018)

2.5.2 Reverse Engineering in Industry

The other aspect of reverse software engineering is to be used in the industry setting. For the industry, everything must be as efficient as possible to maximize profit gains and

minimalize deficit. Reverse engineering can help by offering a more efficient way to perform software debug and maintenance. By reverse-engineering software, one can add comments and notes to the source code to clarify and specify the intention of the section of the code. A program usually has more functionality than it intended to, and by understanding each and every line of the code, the debugging phase of the software can be reduced and the misunderstanding during the software development phase will be lessened. During the reverse engineering process, the source code is not altered, but more information about it is generated (Muller, Wong, & Tilley, 1994). Another major usage of the reverse-engineering in the industrial area is the fact that many of the original programmers who wrote the code for a program do not keep working on that program, but rather pass it onto the next phase or for future uses. Which means that the person who worked on the program next will have to first study the source code to find out how the program works, which will take time and money to do so. But with reverse-engineering, one can lessen the burden on the next programmer by explaining the functions and code in the comment sections.

2.6 UML Activity Diagram

The Unified Modeling Language or UML is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system. (Booch, Rumbaugh, & Jacobson, 1998) The UML Activity Diagram is an important diagram in UML to describe the dynamic aspects of the system. The diagram performs similarly to a flowchart that represented the data flow from one activity to another with the purpose to capture the behavior of the system. (TutorialsPoint, 2018). The purpose of the activity diagram works wonderfully in fulfilling the requirement of reverse-engineering. Since the UML Activity Diagram shows the flow of data, the sequence of the control flow, and how each decision branched out, the purpose of the methods in the source code can easily be interpreted by the user of the diagram.

UML Activity Diagram was mainly used as a flowchart to show the activities performed by the system. Before drawing the diagram, there are multiple elements to be identified, such as the activities, the association of the activities, the condition, and the constraints. Once the elements are identified, the diagram flow design can start. The UML Activity diagram has a template of how a general diagram should look, and are mostly consist of 5 elements.

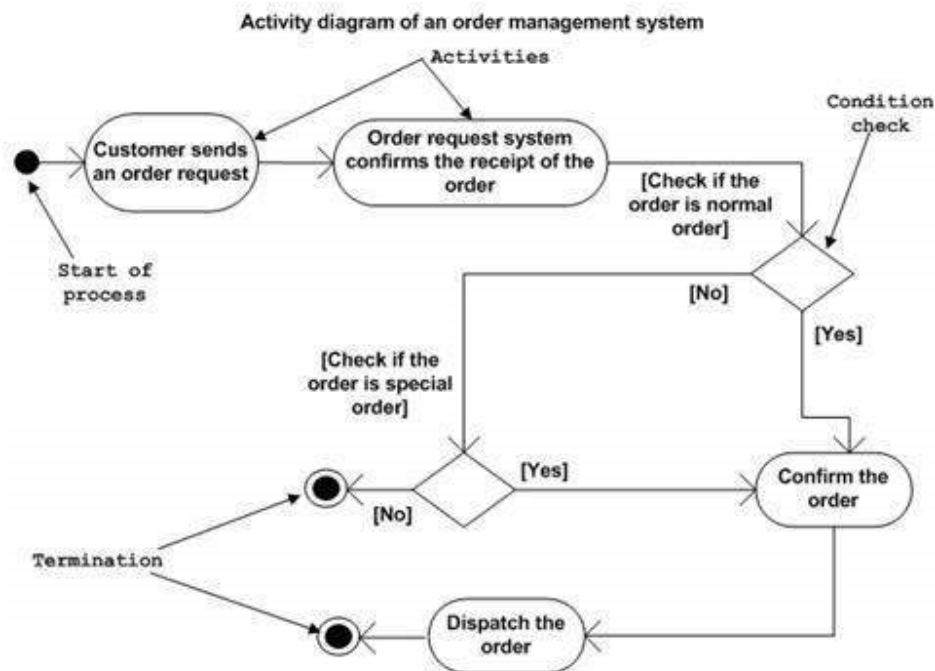


Figure 7. Example of Activity Diagram.³

The UML Activity Diagram as shown as an example in Figure 7 starts the flow with a solid black circle, representing the start of the process. Each of the activity will be represented by the rectangle with round edges, with a phrase representing the activities inside the shape. Connecting all the activity will be a directional line with a point toward

³ "UML Activity Diagram", digital image, viewed 13th November 2018, <https://www.tutorialspoint.com/uml/uml_activity_diagram.htm>

the next activity, representing the “directional flow” from one activity to another. The flow can only be in one direction. If the activity is a condition, the shape will instead, becomes a rectangular-diamond shape, with the condition phrase itself above the shape, and the resulting phrase being under the shape around each branch’s directional line. A certain version of the UML Activity Diagram may have the conditional phrase in a normal activity shape prior to the conditional diamond for a more organized look. At the end of each and every branch of the UML Activity Diagram, there must be a termination point, represented by a similarly shaped black circle to the starting point, but with an additional circle around the shape (TutorialsPoint, 2018).

2.6 Support Tools

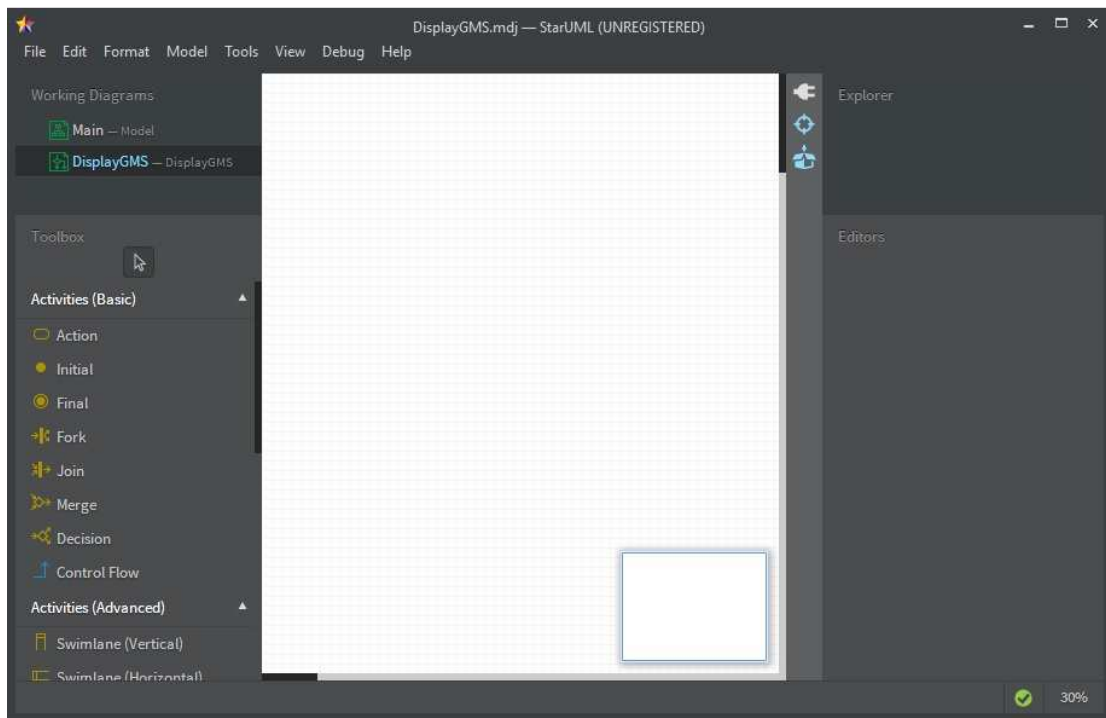


Figure 8. StarUML Windows Interface

The UML Activity Diagram created during this thesis were created in StarUML software, an open-source software to generate many types of diagrams using an easy to

use graphical user interface meant for professional persons and educational institutes (StarUML, 2018). The reason StarUML is chosen for the creation of the UML Activity Diagram was due to its accessibility and its open-source nature. StarUML is a multi-platform software which means that the software can be installed and use on all 3 major operating system, Windows, Mac OS, and Linux. The accessibility nature of the software means that the generated diagram can be accessed by many of the users as possible, as long as the user owned the software and is installed in their machine. But that part is also covered by the open-source nature of the software, which means that the software is free to download and use. So, any user should have access to the software to be able to view the diagram. Figure 8 shows StarUML application user-interface that was used in the creation of the diagrams made for this thesis as StarUML version 2.1.4 on machines with Windows 7, 8.5, and 10 OS. The “Activity Diagram” option was selected to create each of the diagrams. The modeling concepts used for generating the diagram as shown on the bottom left of Figure 8 was:

- Action: a bubble to create an activity. Can click and drag to create the activity to be a specific size or click to place a default size bubble to just put information in and let the auto-resize feature adjust the bubble size to perfectly fit the information in the activities.
- Initial: the initial node is crucial in all activity diagram as it marks the starting of the flow in the diagram and locating the first activity to run in this program. There can only be one initial node per diagram.
- Final: the final node or terminal node is a node that marked the end of the path flow in the diagram. There can be more than one final node in a single diagram due to the fact that a single program can have multiple termination points.
- Decision: the decision diamond split a flow into at least two directions according to the conditional statement in the immediate activity prior to the decision node.
- Control Flow: an arrow used to represent the flow of the activity from one to the next, with the arrowhead pointing toward the next activity. The control flow line

can have multiple styles, either rectilinear or oblique, which is mostly up to the developer of the diagram to decide.

The usage of these options will be explained in more detail in conjunction with the UML activity diagram made for this thesis in chapter 3.

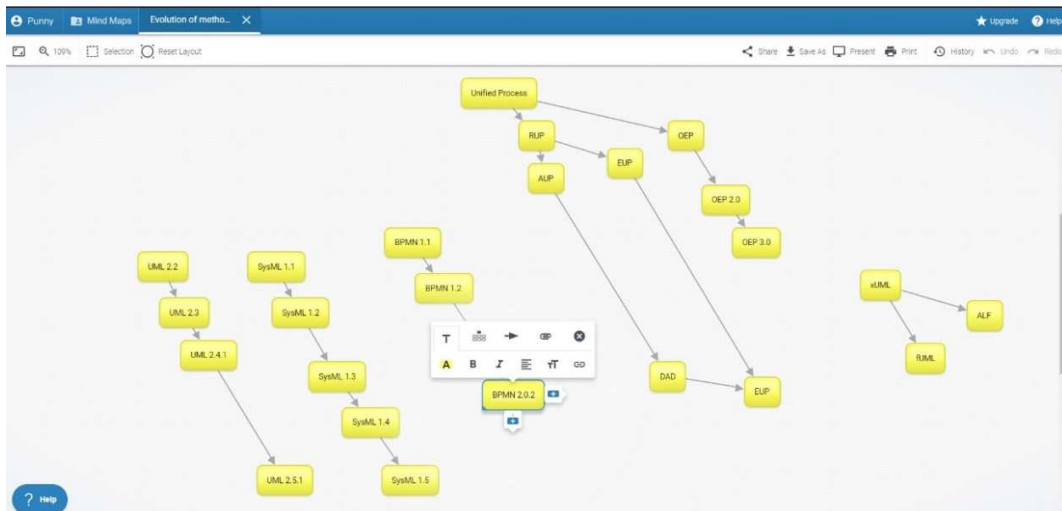


Figure 9. bubbl.us Web-Interface

In order to update “Evolution of Methodology” (Figure 1) to the current time however, bubbl.us was chosen as a platform to generate the chart instead of the StarUML. Bubbl.us is a free to use, web-based mind map generator, which is perfect for a one-timed generation of a graphical chart image. Bubbl.us works the same on all platform since it is web-based software, so it provides a degree of accessibility to the user. The software also able to link itself onto Google Drive to directly deposit the generated chart for easy access. However, bubbl.us is not powerful enough to generate a more detailed diagram to be used as the UML Activity Diagram (LKCollab, 2018).

Figure 9 shows the web-interface of bubbl.us, where the user can make multiple mind-map type diagrams on each tab on the top left. If the user clicked on any of the blue “+” symbols on the right and bottom of the selected item, another bubble for the purpose of making the diagram will be created. Using this web-interface, the user can customize the font and how the text inside the bubble looks, the user can also customize other parts of the bubble such as the color and the pointer to their liking.

3. APPLICATION OF METHODOLOGY

3.1 Source Code Background

Research showed that many software development projects fail because of the inability to deliver the product in a timely and cost-effective manner, i.e. the software crisis. In 1998, Paul Dorsey reported the reasons of systems projects to fail (Dorsey, 1998), which one of those is the lack of use of an appropriate software development methodology and focusing the development efforts on coding. The initial strategy for modeling the system was used as an Agile based methodology. After developing the system's user interface and generic algorithm solutions, the coded components of the system were rapidly produced. Then, after consultation with the stakeholders, the code was refined.

3.2 Modified Methodology

Our study modified the development methodology in use to accommodate the work of the initial strategy. This modification was an iterative reverse engineering process that is illustrated in Figure 10.

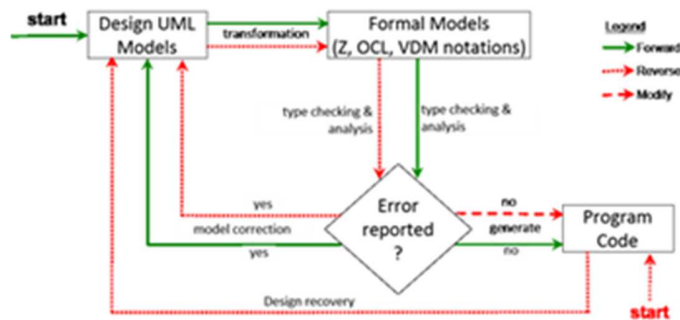


Figure 10. Reverse-Engineering Modified Model-Driven Methodology

The process model of Figure 10 was developed to incorporate a reverse-engineering strategy to complement the forward-engineering activities. This process model also illustrates the use of formal specification techniques for validating the reverse and forward engineering activities. The “Design UML Models” activity of Figure 10 is reflective of the “Conduct High-Level Design 5.2.2” and “Conduct Low-Level Design 5.2.2” of Figure 5, and the “Formal Models” activity of Figure 10 is synonymous to the “Verify High-Level Design 6.3” and “Verify Low-Level Design 6.3” activities of Figure 5. The green (solid) arrowed lines represent the forward engineering path through the process model, while the red (broken) arrowed lines represent the reverse engineering path through the model. The forward engineering process commenced with the “Design UML Models” activities, while the reverse engineering process commenced at the “Program Code” activity.

This modification to the development methodology then transitioned along the reverse engineering line “Design recovery” line, from the “Program Code” to representative “Design (high and low) UML Models”. The UML models were then transformed to a formal representation in the Z notation for analysis during the verification phases of the Figure 5 methodology. If the models pass the verification, then work transition along the “generate” arrowed lines to the production of “Program Code”. Otherwise, work transition along the “model correction” arrowed line to the UML models and the next iteration of the process commenced with the identified errors being corrected in the models (Grant & Ajjimaporn, 2018).

3.3 Project Implementation

After developing an acceptable generic algorithm solution for the airline-gate assignment problem and a user interface for the gate-assignment conflict resolution system were done, then the UML activity diagram was developed for a source code of an aircraft-gate assignment system.

To verify, validate, and system documentation of reverse engineering a set of UML models of the genetic algorithm system, we selected to identify this system as a Level-A DO-178C system, in order to exercise as many of the models-driven methodology's activities.

The main UML model developed was a set of activity diagrams that was implemented at the detailed-level of system modeling. The limitation to producing just one type of UML model was borne out of the airline system administrators' preference for just the necessary models to facilitate any immediate small-scale bug fixes, versus models to be used for system evolution. The nature of the contract between UND and the airline called for the software system's on-going maintenance (evolution) to be further contracted out to a third party. A sanitized example of a segment of one of the UML activity diagrams that were developed is presented in Figure 11 (Grant & Ajjimaporn, 2018).

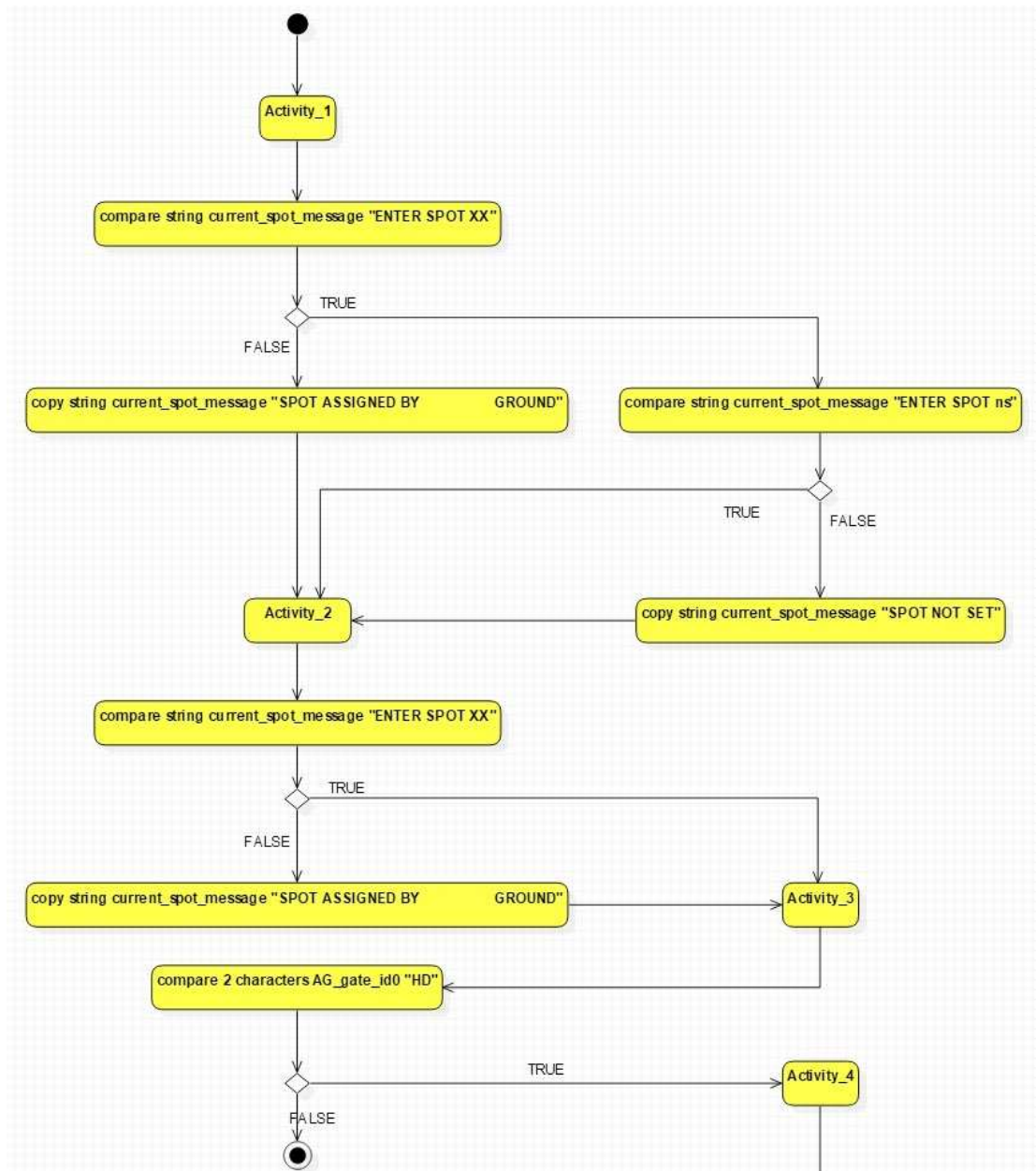


Figure 11. UML activity diagram of Aircraft-Gate Assignment System.

The diagram was made to represent the entire source code, by going through and analyzing the source code line-by-line. Whenever there is a condition in the source code, the diagram branches out and interact with other activities. To directly see the flow of the program, the directional arrows in the diagram were assigned.

UML Activity Diagrams were developed for source codes of an aircraft-gate assignment system. The source codes are written in C which makes the UML Activity Diagram much simpler due to the lack of custom objects. The source code includes many header files that will be excluded in the making of the activity diagram due to their main function is to provide additional functionality to the source code and those functionalities have no need to be represented in the diagram.

Since StarUML support multiple diagrams in a single project file, the decision was made to separate the StarUML's .mdj (metadata-json) files into 3 files to represent each of the c language source code given. Each separate diagram will represent a method from the source code. The flow of the diagram will be from top to bottom, then from left to right in the case that the space to draw the diagrams given by StarUML weren't enough (which there were several cases that weren't). There was some unconventional usage of the diagonal line to connect between activities instead of the traditional rectilinear (only horizon and vertical lines) to lower the amount of cluttering since there were many cases where more than a few activities were linked to a single activity. The diagram seen in Figure 11 was colored to make it easier for viewing in a plain white paper of thesis.

```
cout << "current_dart_message=" << current_dart_message
<< ".xx" << endl;

// GET GATE NUMBER FOR TITLE
hold_gaterecno = gate_recno;
strcpy (EG_gate_id, gate[gate_recno].id);
sprintf (title, "GATE %-4.4s", EG_gate_id);
```

Figure 12. Code Snippet from UML Activity Diagram.

The beginning of the method will have denoted by the initial node icon represented by a black circle as seen on the top left-most of the diagram in Figure 11. Any of the traditional statement in the source code will be summarized by a generic title due to the lack in their choice in the flow of the diagram. Figure 11 does not illustrate any significantly unusual activity diagram modeling technique, but with the exception of the listing of some activities with generic titles, such as “Activity 1”, “Activity 2”, etc. This was done in order to capture very low-level details of the program code, from which the model was reversed engineered. The models were developed in the open-source tool StarUML and the contents of “Activity-Xs” were stored in the documentation fields of the models. As implemented in this activity diagram, “Activity 2” is the snippet of code presented in Figure 11

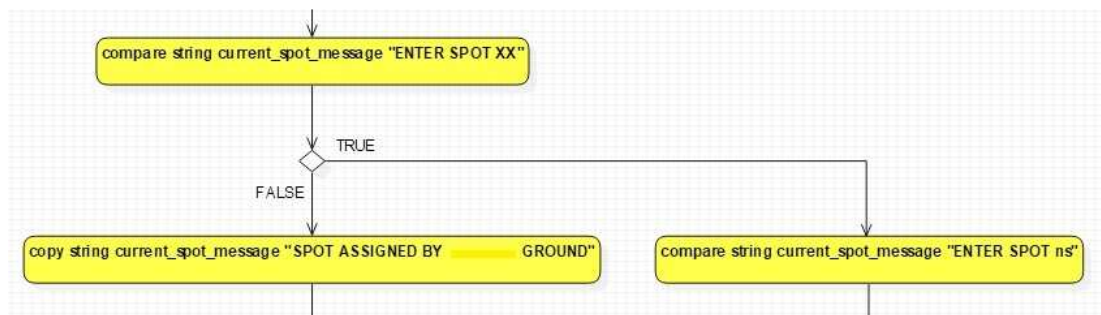


Figure 13. Example of a condition in UML Activity Diagram

The conditional statement such as if...else, for...loop and try...catch will be the statements that potentially change the direction of the flow of the diagram, and thus will be represented by an activity prior to the diamond conditional icon as seen in Figure 13 under the activity labeled “compare string current_spot_message “ENTER SPOT XX”. The conditional statement such as this will direct the flow of the data to an activity if the output to the conditional statement return as TRUE, and to another activity if it returns as FALSE. In any and all conditional statement, the result will always be of a Boolean type, which is either TRUE or FALSE, there is no “in-between” and there is no “no answer”. From the example snippet of the diagram shown in Figure 13, if the result of the

comparison between a string variable “current_spot_message” and “ENTER SPOT XX” is TRUE, then the program will proceed to do the activity to the right, following the control labeled “TRUE”. In the case that the statement returns as FALSE, the program will proceed toward the activity following the control labeled “FALSE”.

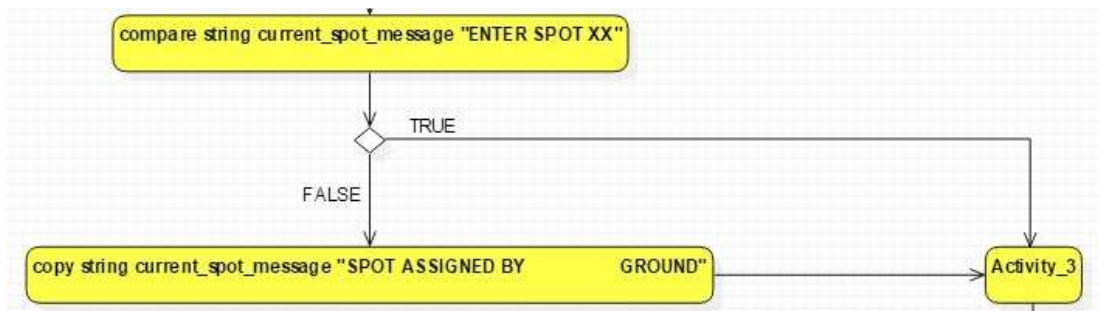


Figure 14. Example of a conditional statement with only one activity generated

The conditional statement seen in Figure 13 is an example of the statement that proceeded to follow by two activities made specifically for the given conditional statement. Sometimes, the conditional statement only needed to generate one activity to flow into for a condition and skipped the generated activity to the next activity in the flow for the other condition. Figure 14 shows the example of this type of conditional statement by that if the conditional statement returns FALSE, the program will proceed to the activity generated specifically for the conditional statement (copy string current_spot_message “SPOT ASSIGNED BY GROUND”), then proceed to the next activity (Activity_3), but if the conditional statement returns TRUE, then the flow proceeds directly to the next activity (Activity_3).

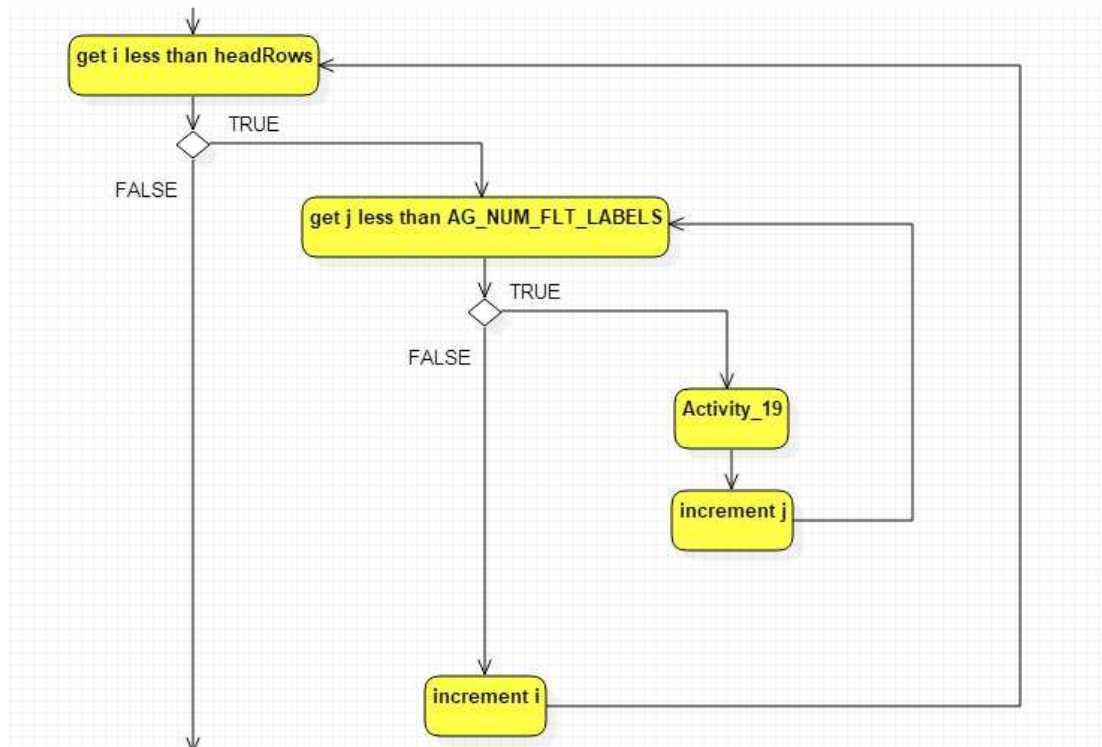


Figure 15. Example of a loop in the UML Activity Diagram

One benefit of using a UML activity diagram to represent the code is how easy it is to identify the usage of a loop in the source code. Loop in the UML activity diagram was shown by the “circular” control flow of the diagram. Figure 15 shows an example of the loop in the diagram by following the cyclic directional control flow from the conditional activity “get j less than AG_NUM_FLT_LABELS” through a TRUE path to “Activity_19” and activity “increment j”, then back to the starting conditional activity of “get j less than AG_NUM_FLT_LABELS”. The example uses a “for...loop” conditional statement that took advantage of the local variable “j” that were created for the specific purpose to be a counter during this loop. The conditional statement uses the local integer variable “j” to compare to the integer variable AG_NUM_FLT_LABELS, if the variable “j” is lower than the given variable, the flow will be directed to an activity, then to a specific activity that increment the variable “j” so that it can be used to compare again in

the starting conditional statement. The example loop was designed to run through Activity_19 several times until the variable “j” no longer contains value less than the given value. In computer science, a loop can be thought of in the terms of multiplication of the activity, on how many times an activity will run. Figure 15 also shows the “double loop” which contains a loop inside another loop, which means that with each incrementation of the outer loop, the inside loop will re-initialize the local counter. The double loop is a multiplication of multiplication in terms of the activities ran. From the double loop in Figure 15, every time the conditional statement “get j less than AG_NUM_FLT_LABELS” finished its loop, it increments the local variable “i” counter by 1, which plays part in the outer loop of “get i less than headRows”. It means that when the activity flows back to the “get j less than AG_NUM_FLT_LABELS”, the “j” variable re-initialized so that the “j” variable returns to its initialize values.

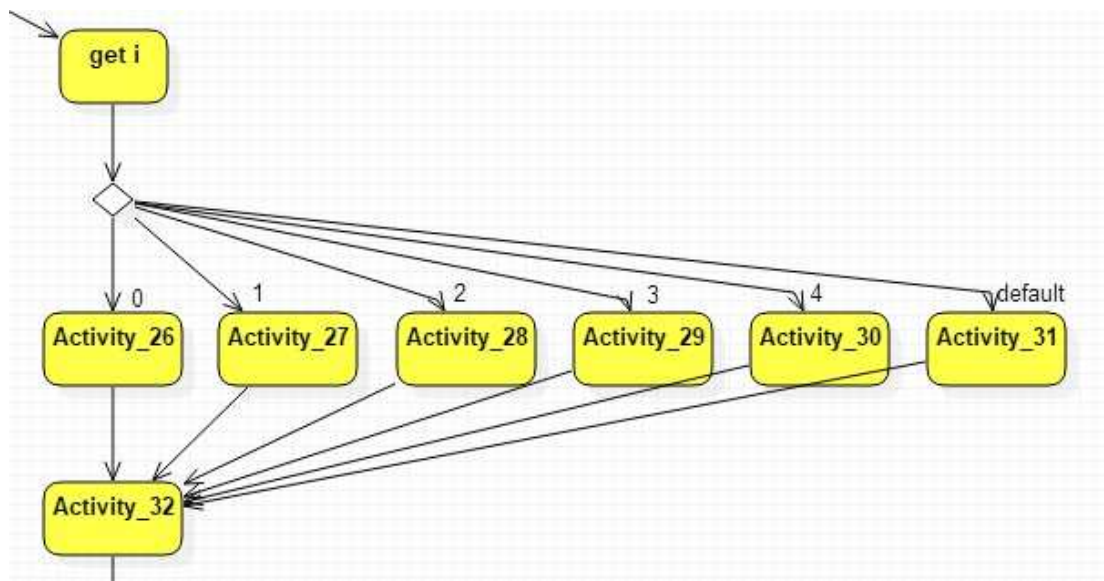


Figure 16. Example of a switch...case statement in UML Activity Diagram

An exception to the conditional statement was made when a “switch...case” was encountered. For switch...case, the conditional statement no longer flows as a binary

result of TRUE or FALSE, but rather with the multiple result branches from the conditional diamond as shown in Figure 16.

At the end of every diagram, or when an encounter with the statement “return” in the source code marks the termination of that method. The termination node was represented by a black solid circle inside a black circle outline as shown at the lower left in Figure 11. The termination node in Figure 11 was a result from the conditional statement which can either result in termination of the method or proceeds to activity “Activity_4”. Every activity node should either flow to another activity, to a diamond conditional node, or to termination node.

4. RESULTS

With the data flow of the source code being represented in the visual format of the UML Activity diagram, the interaction of each component can be easily understood and identified. The user can see the information that goes into each method, and what each method required. Once the user understands the flow of data within the program, the user can validate and verify that the software was developed with a correct method.

4.1 Types of Model Developed

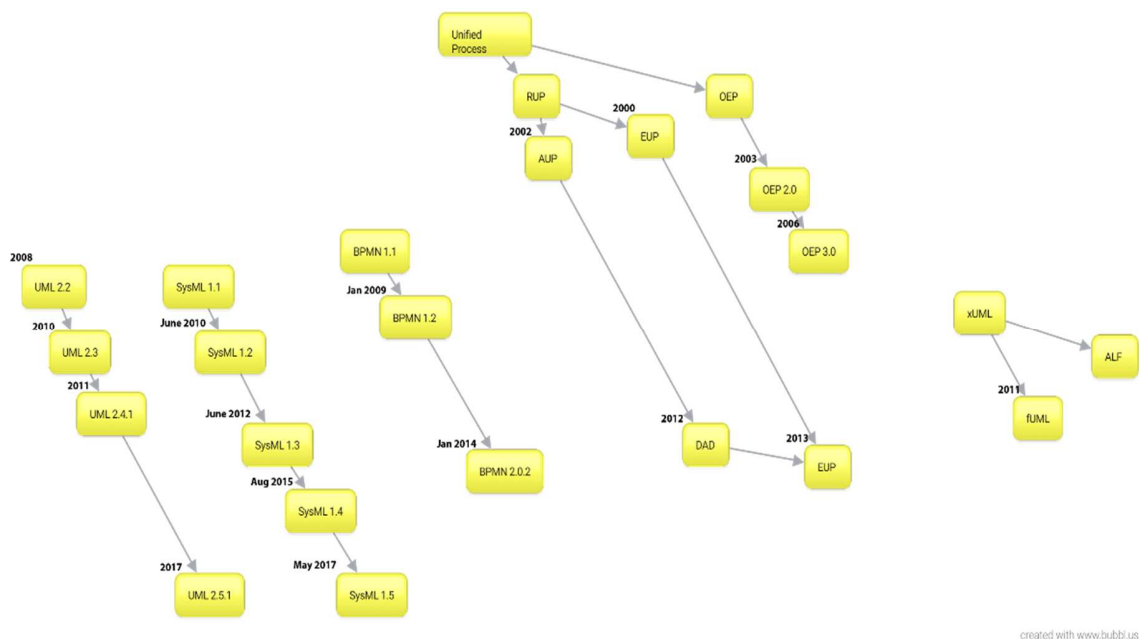


Figure 17. expanded diagram figure 1

The continuation of figure 1 chart shows the latest version of UML notations being as recent as version 2.5.1 from December 2017. This continuation chart was developed for this research work. (Object Management Group, 2017) There are methods and notation that works on a more specific system such as Unified Component Model for Distributed, Real-Time and Embedded Systems. This model specification is very recent as their first

version, 1.0 was adopted on Jan 2018. (Object Management Group, 2018) Most of the modern methodologies that used Unified Process now evolve to be based on Disciplined Agile Delivery (DAD) or the more modern Enterprise Unified Process (EUP) instead. Figure 17 also shows the current version of SysML of 1.5 since its branched out from UML 2.1.2 back in 2007. The Business Process Modeling and Notation or BPMN was created to represent business processes in a graphical way so that end-users can understand processes concept (Campos & Oliveira, 2013). The chart also shows how the relationship between the Unified Process-based models such as RUP, AUP, EUP, DAD and new EUP. Lastly, Figure 17 shows the last relatives of OOSA, the xUML and how it now has branched into Foundational UML (fUML) and the Action Language for fUML (Alf).

During the work for this thesis, the Department of Computer Science researchers formed three teams. The first team focused on developing the genetic algorithms to implement the aircraft-to-gate assignment solution. The second team focused on the design and implementation of the user interface of the system. The third team focused on the documentation of the system, and this work of the thesis is from the third team (team III).

The effort was centered on that of reverse engineering a set of UML models of the genetic algorithm system for the purpose of verification, validation, and system documentation. It was opted to identify this system as a Level-A DO-178C system, in order to exercise as many of the model-driven methodology's activities, as represented in Figure 4. The intent was to garner as many pedagogical benefits as possible for incorporation into the software engineering curricula of the department and provide comprehensive system documentation artifact to the stakeholders.

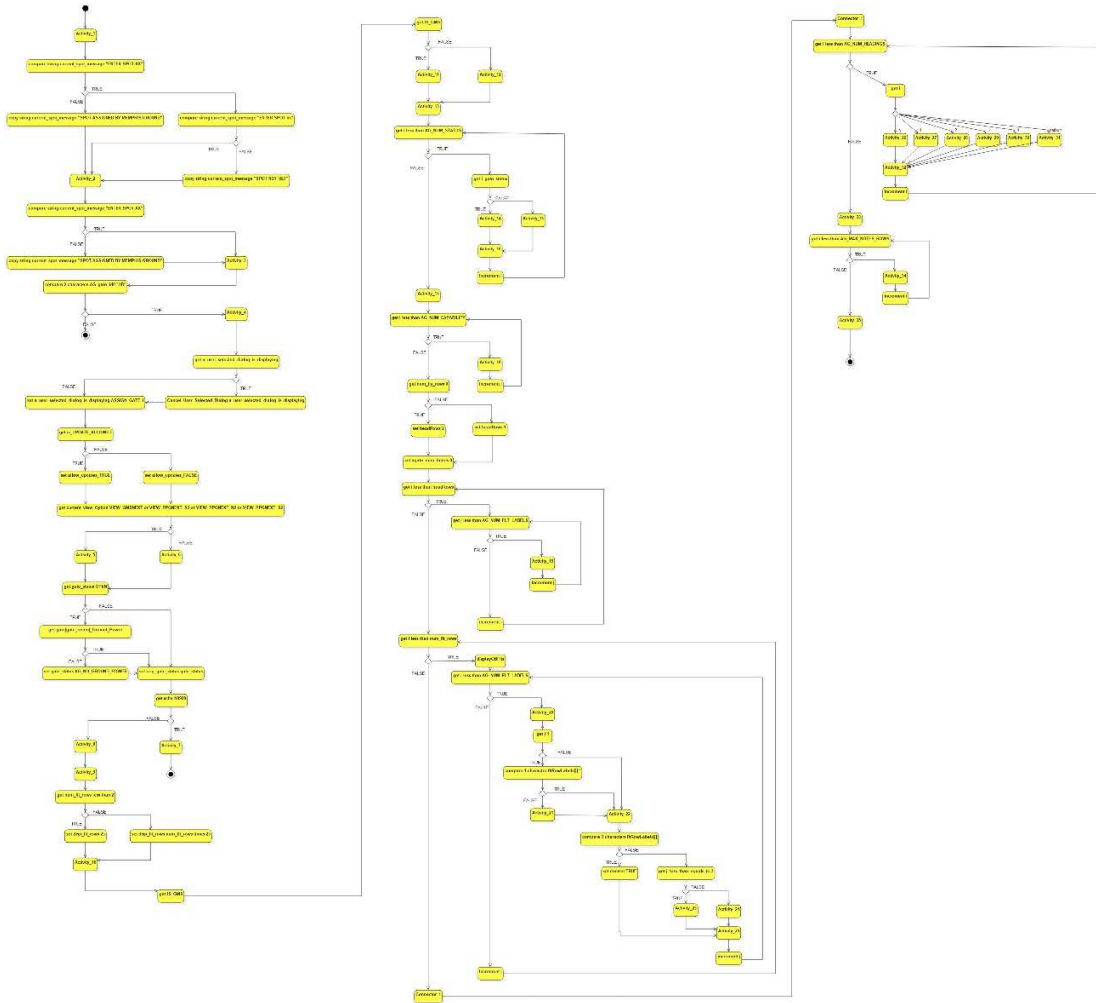


Figure 18. Example of a full-size diagram

Figure 18 shows an example of one of the full-size diagram developed for this thesis. The diagram was made small to capture the detail of the software system and to show the size and work done. The example shows the many components of the UML activity diagram and how they come together into a single diagram.

4.2 Description of Activity Diagram

Modified UML activity diagrams were created to answer the requirement in reverse-engineering for the purpose of verification and validation of the source code. There were 18 diagrams created from the 3 source files given, with the majority of the diagrams belong to the assign gate source code. The average size of the models produce was approximately 100 bubbles and 30 decision nodes from an average line of code of each source code at 1300 lines. The main UML model developed was a set of activity diagrams that was implemented at the detailed-level of system modeling. The limitation to producing just one type of UML model was borne out of the airline system administrators' preference for just the necessary models to facilitate any immediate small-scale bug fixes, versus models to be used for system evolution. . The amount of time spent at each stage of the work are: approximately 10 hours understanding the code and its internal connections, 15 hours researching for an effective diagram to be representing the code, 25 hours implementing the conversion of the code to diagram and up to 50 hours verifying and checking for errors in the diagram. The time described do not includes the changes to the code as the project progress and the changes made to the initial diagram.

The reverse engineering of the source code acts as a verification method to ensure that the source code produced the correct result. the UML activity diagrams show the work of the low-level design of the software system and its data flow, making the process of knowing whether the software met the specification easier. Using the graphical models such as UML activity diagram for specification manages complexity and improves reusability and analytical capabilities (Beckmann, Vogelsang, & Reuter, 2017), (Apfelbaum & Doyle), (Vogelsang, Eder, Hackenberg, Junker, & Teufl, 2014).

With the data flow of the source code being represented in the visual format of the UML activity diagram, the interaction of each component can be easily understood and identified. The user can see the information that goes into each method, and what each method required. Once the user understands the flow of data within the program, the user can validate and verify that the software was made with a correct method.

Figure 19 shows the steps taken from the reverse-engineering modified model seen in figure 10. UML activity diagrams of the source code were made to capture the very low-level details of the program flows. Each activity of the diagram can be traced back to a section of code within a method. This process is called traceability, which plays a major part in verification process of DO-178C (Jacklin). With each method being a function to perform an action, one can easily see a link between the source code and the action it performed. By crafting the activity diagram straight from the source code, this allows the viewer to see that each line has a purpose that leads to the fulfillment of the low-level requirements of the system. The user can verify that each line has purpose, and thus, ensuring that the system is complete. This shows the low-level requirements aspect and verification aspect of the software system which specifically shown as parts of the requirements needed for DO-187C compliant model-driven methodology as “Conduct Low-Level Design 5.2.2” and “Verify Low-Level Design 6.3”. By complying to the DO-187C model, UML activity diagram has a place in the domain of safety-critical system. The UML activity diagram also works as part of the “required documentation” step of the system which cuts down the need for many other diagrams that would be generated for the system. Another aspect of UML activity diagram that make it the contender to fit the preferred model for any safety-critical system, due to its graphical nature, is its ability to be interpreted by any user and not just those that took part in the development phase of the system.

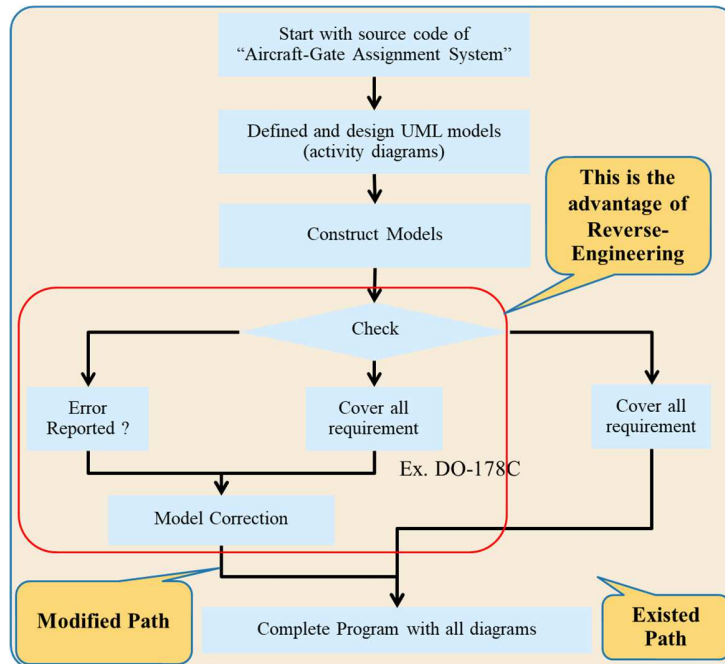


Figure 19. Steps of reverse-engineering modified model

5. DISCUSSION

5.1 Lesson Learned

UML-based development methods were already used successfully for integrating with safety-critical systems in embedded software (Anda, Hansen, Gullesen, & Thorsen, 2006). There were also a few approaches to reverse-engineer using UML diagrams. These were class diagrams and sequence diagrams, the combination of which can help with the verification component of reverse engineering similar to the activity diagram (Viktoria Ovchinnikova, 2014).

Reverse-engineering in software developments are usually synonymous with the usage of model-based methodology and the usage of the diagram since the visual modeling represents the behavior of the system at a higher abstraction level. It is important to create a well-constructed model due to its unambiguous and easy to understand nature. Otherwise, the model can be misinterpreted (Angyal, Lengyel, & Charaf).

The UML activity diagram was made to capture the operational workflow of the software system, and thus not useful in trying to obtain a static class structure, which is captured best by UML state diagram or the internal behavior, which is captured best by UML sequence diagram. UML diagrams are imprecise and do not apply well to the distributed system. The reason why UML diagrams are imprecise is due to the flexible nature of UML diagram, which leads to subjective interpretation, but that can always be fixed with a heavy usage of documentation (Buse).

5.2 Difficulties Using UML Activity Diagram

A UML activity diagram represents the possible flow of the activities' interactions, but it does not show what specific variable in the source code was responsible for the flow (outside the conditional statements). There is a reason why UML diagrams are not

consisting solely of an activity diagram, but also many others such as class diagram, sequence diagram, etc. Using a combination of the diagrams can help with managing the complexity of software systems and making it easier to spot both desirable traits and undesirable traits of the programs. Due to the nature of the UML activity diagram structure, it is hard to capture the specific variable that passed into the method without the usage of other diagrams or breaking the UML activity diagram model.

6. CONCLUSIONS

6.1 Acceptance of Work

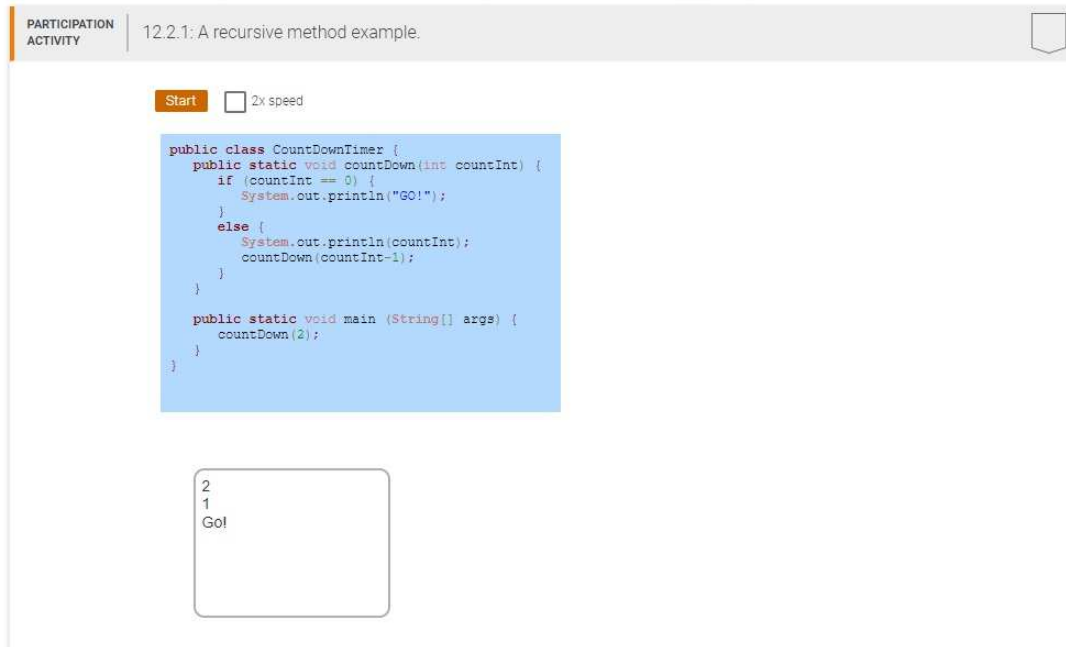
As the activity diagram represents the pseudo code of the program in a graphic way, it should be a good candidate to be used as a tool to help in reverse-engineering the safety critical systems of the airline system. The UML activity diagram represents many aspects required in the deconstruction phase of the reverse-engineering methodology. With the source code in the graphic diagram form, it should be much easier to identify the structure and functions and determine how each aspect of the program interact with each other so that the activity diagram can be used in the formal methodology for reverse-engineering (Beckmann, Vogelsang, & Reuter, 2017).

6.2 Experience in Conducting Reverse Engineering

University of North Dakota's Data Engineering course taught the graduate students to use reverse-engineering to create an android application and web-based application. The android application created for Data Engineering class must integrate Android server connection and some form of utilization for Google Maps API. The student learns both of the techniques by following step-by-step instructions on how to set-up the application, but in order to apply the concept to the application, students were first introduced to the example programs that work with both concepts fully integrated into the program. The student then deconstructs the example code to locate the part of the example program that is responsible for the connection to the android server, and for the Google Maps API. After locating the specific chunk of code and fully understanding its concept and functionality, the student proceeds to integrate those methods and functions into their own code, making an original application with the two functions. A similar concept was applied for the web-based application programming exercise. This showed that the concept of reverse-engineering had already been applied broadly throughout the computer science field, even though the concept itself had not been taught as a class.

12.2 Recursive methods

A method may call other methods, including calling itself. A method that calls itself is a *recursive method*.



The screenshot shows a Java IDE window titled "12.2.1: A recursive method example." The code defines a `CountDownTimer` class with a recursive `countDown` method and a `main` method that calls `countDown(2)`. The output window shows the execution result: "2", "1", and "Go!".

```
public class CountDownTimer {
    public static void countDown(int countInt) {
        if (countInt == 0) {
            System.out.println("GO!");
        }
        else {
            System.out.println(countInt);
            countDown(countInt-1);
        }
    }

    public static void main (String[] args) {
        countDown(2);
    }
}
```

2
1
Go!

Figure 20. Example code used in the Computer Science II

Another computer science class taught at the University of North Dakota, Computer Science II, is also taught with the idea of reverse-engineering as a background practice. In Computer Science II, the students learn basic to intermediate object-oriented programming concepts in Java, from objects to pointers. The students learn the concept aspect of the topic during the lecture. Once a week, the student gets to apply the concept to practice during the lab hours where the teaching assistant assigns a complex exercise based roughly on the current concept covered in the lecture during the previous week. The assignment is usually accompanied by an example code where the concept of the exercise is applied roughly in a similar fashion. The students can use the example and modify it to fit the exercise description in the same fashion as described earlier, with reverse-engineering. Figure 20 shows an example of the code the student may deconstruct to fit their own use during the learning of “Recursive Methods” concept in Java (zyBooks,

2018). The students can see the example code and its generated result to match with the concept of recursive methods, a method that called itself recursively. The students would notice that the method of “countDown(int countInt)” has a statement that called itself on “countDown(countInt-1);” This kind of statement was never shown before in the class, and they should be able to figure out that this method will first be given a starting number, then call itself with the same number lowered by 1 until the method itself had a 0 as the countInt variable to be able to terminate out from the recursive loop.

6.3 Contribution to the work

Principal contribution to this work made by the author was in the development of the activity diagrams by searching for a meaningful and accurate representation of the methodology and apply the diagram to the code, modification of the applied methodology, interviewing the code developers, and in developing the code components in the activity diagram.

6.4 Future of Reverse Engineering

Reverse software engineering will be a major study topic for as long as there is innovation in the field of computer science. As the program gets more and more complex, the need to understand the base functionality of said software will increase and the usage of reverse-engineering will be a must. Since UML activity diagram has potential to help reverse-engineer safety-critical software, it should be applied to other fields such as the medical field and the military field. The usage of UML activity diagram in the medical field can help with conveying the information about the patient to others due to the easy-to-understand nature of diagrams. Either way, the use of diagrams in the process of reverse-engineering of a safety-critical system helps in reducing the risk of loss of life and make the process of understanding a complex system easier.

Due to the importance of the role of the diagrams in the process of reverse-engineering, there should be an automatic process in generating a different kind of UML diagram

directly from the source code, where the user can decide which UML diagram will represent the desired traits of the program for the purpose of verification and validation. Since the coverage of a single type of UML diagram might not fully cover all the desired traits of the process of reverse-engineering, it might be more useful for multiple types of UML diagrams and models to be applied to the program at one time.

REFERENCES

- Agile Alliance. (2019). *Agile 101*. Retrieved from agilealliance.org.
- Anda, B., Hansen, K., Gullesen, I., & Thorsen, H. K. (2006). Experiences from introducing UML-based development in a large safety-critical project. *Empir Software Eng.*
- Anderson, K. (2019, October 4). *What is Rapid Application Development and When Should You Use It?* Retrieved from capterra.com.
- Angyal, L., Lengyel, L., & Charaf, H. (n.d.). An Overview of the State-of-The-Art Reverse Engineering Techniques. *Magyar Kutatok 7. Nemzetközi Szimpoziuma.*
- Apfelbaum, L., & Doyle, J. (n.d.). Model Based Testing. *Software Quality Week Conference.* 1997.
- Asif, N. (2003). *Reverse Engineering Methodology to Recover the Design Artifacts: A Case Study.* Retrieved from Semanticscholar.org.
- Association of Modern Technologies Professionals. (2018). *Software Development Methodologies.* Retrieved from IT Knowledge Portal.
- Barone, S., & Mandredi, E. (2007, August). *Reverse Engineering As A Learning Process.* Retrieved from Researchgate.net.
- Beckmann, M., Vogelsang, A., & Reuter, C. (2017). A case study on a specification approach using activity diagrams in requirements documents. *Requirements Engineering Conference.* IEEE.
- Booch, G., Rumbaugh, J., & Jacobson, I. (1998). *Unified Modeling Language User Guide, The.* Addison Wesley.
- Bothe, K. (2001). *Reverse Engineering: The Challenge of Large-Scale Real-World Educational Projects.* CSEE&T.
- Buse, R. (n.d.). UML: A Selected Overview of the Unified Modeling Language. Retrieved 2018, from [http://www.petrik.hu/files/Tananyagtar/KRAJNYAK_ATTILA/13_OOP_UML%20diagramok/\(C\)%20UML%20diagrams.pdf](http://www.petrik.hu/files/Tananyagtar/KRAJNYAK_ATTILA/13_OOP_UML%20diagramok/(C)%20UML%20diagrams.pdf)

- Campos, A., & Oliveira, T. C. (2013). Software Processes with BPMN: An Empirical Analysis. *Product-Focused Software Process Improvement* (pp. 338-341). Paphos: PROFES 2013.
- Centers for Medicare & Medicaid Services. (2008, March 27). *Selecting a Development Approach*. Retrieved from CMS.gov.
- Dorsey, P. (1998). 10 Reasons Why Systems Projects Fail. *Technical Report*.
- Engard, B. (2016, November 21). *The Power of Reverse Engineering*. Retrieved from thesoftwareguild.com.
- Gannod, G. C. (1994). The Application of Formal Methods to Reverse Engineering of Imperative Programming Code.
- Grant, E. S., & Ajjimaporn, P. (2018). Pedagogical Benefits from an Exercise in Reverse Engineering for an Aviation Software Systems.
- History of Object Oriented Modeling Languages*. (2018). Retrieved from wikipedia.
- Hood, V. (2018, June 27). *Bethesda Suing Warner Bros. For 'Blatant Rip-Off' of Fallout Shelter*. Retrieved from IGN.com.
- Innovation Enterprise Channels. (2019). *The important Role of Software Reverse Engineering*. Retrieved from theinnovationenterprise.com.
- Jacklin, S. A. (n.d.). Certification of Safety-Critical Software Under DO-178C and DO-278A. Retrieved 2018, from <https://ti.arc.nasa.gov/publications/5357/download/LKCollab>.
- (2018). *Bubbl.us*. Retrieved from bubbl.us.
- Merriam-Webster. (2018). *Methodology | Definition of Methodology*. Retrieved from Merriam-Webster.
- Merriam-Webster. (2018). *Reverse Engineer*. Retrieved from Merriam-Webster.
- Muller, H. A., Wong, K., & Tilley, S. R. (1994). *Understanding Software Systems Using Reverse Engineering Technology*. University of Victoria, Department of Computer Science. Retrieved 2018, from <https://pdfs.semanticscholar.org/e274/aa66a3ee57be143aca2cc518333ba093426b.pdf>
- New World Encyclopedia. (2018). *Reverse Engineering*. Retrieved from New World Encyclopedia.
- NPD Solutions. (2016). *A Methodology for Reverse Engineering*. Retrieved from NPD-solutions.com.
- Object Management Group. (2017, December). *About the Unified Modeling Language Specification Version 2.5.1*. Retrieved from Object Management Group.
- Object Management Group. (2018, January). *About the Unified Component Model for Distributed, Real-Time and Embedded Systems Specification Version 1.0*. Retrieved from Object Management Group.
- Oleksandrova, O. (2018, June 1). *Infographic: A Brief History of Software Development Methodologies*. Retrieved from Intetics.com.
- Pal, S. K. (2018). *Software Engineering Spiral Model*. Retrieved from GeeksforGeeks.
- Powell-Morse, A. (2016). *V-Model: What is it and how do you use it?* Retrieved from Airbrake.io.

- Powell-Morse, A. (2017, March 14). *Rational Unified Process: What Is It And How Do You Use It?* Retrieved from airbrake.io.
- Rashid, N., Salam, M., ShahSani, R. K., & Alam, F. (2013, July). Analysis of Risks in Re-Engineering Software Systems. *Internal Journal of Computer Applications*, 73.
- Rouse, M. (2018). *Prototyping Model*. Retrieved from TechTarget.
- RTCA. (2011). Software Considerations in Airborne Systems and Equipment Certification. DO-178C. *Radio Technical Commission for Aeronautics*.
- Schwaber, K. (1993). Agile Project Management with Scrum. *Microsoft Press*.
- sebokwiki. (2018). *System Modeling Concepts*. Retrieved from sebokwiki.org.
- StarUML. (2018). *StarUML Documentation*. Retrieved from StarUML.io.
- TutorialPoint. (2018). *SDLC-RAD Model*. Retrieved from Tutorial Point.
- TutorialsPoint. (2018). *SDLC-Waterfall Model*. Retrieved from TutorialsPoint.
- TutorialsPoint. (2018). *UML-Activity Diagrams*. Retrieved from Tutorialspoint.com.
- TutorialsPoint. (2019). *SDLC - Spiral Model*. Retrieved from tutorialspoint.com.
- University of North Dakota. (2018). *Requirements*. Retrieved from University of North Dakota.
- Viktoria Ovchinnikova, E. A. (2014). Overview of Software Tools for Obtaining UML Class Diagrams and Sequence Diagrams from Source Code within TFM4MDA. *Baltic J. Modern Computing*, 260-271.
- Visual Paradigm. (2018). *Why UML Modeling*. Retrieved from visual-paradigm.com.
- Vogelsang, A., Eder, S., Hackenberg, G., Junker, M., & Teufl, S. (2014). Supporting concurrent development of requirements and architecture: A model-based approach. *2nd international Conference on Model-Driven Engineering and Software Development*. MODELSWARD.
- Zockoll, G., Scheithauer, A., & Dekker, M. D. (2009, October 6). History of Object Oriented Modeling Languages. Retrieved from https://en.wikipedia.org/wiki/Unified_Modeling_Language#/media/File:OO_Modeling_languages_history.jpg
- zyBooks. (2018). *12.2 Recursive methods*. Retrieved from zyBooks.