ABSTRACT

Title of Dissertation: THE USE OF A MODIFIED SYSTEM DEVELOPMENT LIFE CYCLE (MSDLC) IN A SOCIOLOGICAL ENVIRONMENT TO IMPROVE SOLUTION VALIDATION

Shanelle M. Harris, Doctor of Engineering, May 2019

Dissertation Chair: LeeRoy Bronner, Ph.D., P.E.

Industrial and Systems Engineering Department

This research is investigating the improvements to the current System Development Life Cycle (SDLC). The SDLC has been used since the 1970s with improvements through prototyping and iterations in the system develop phase, however the least expensive phase, system analysis, has not been utilized for improvement. The SDLC lacks the ability to be changing requirements, is not flexible, and lacks user involvement leading to less than complete solutions. With the speed of technology and increase complication of problems facing the world, there is a need to get solutions faster and more accurate. This research will examine the advantages and disadvantages of the SDLC methodologies and provide a validated model and analysis procedure to provide solutions with fewer inaccuracies through an extension of the analysis phase implementing Joint Application Development (JAD) sessions, continuous user involvement, and intermediate artifacts of analysis. The extended analysis phase integrates object–oriented analysis (OOA), Z notation, and Alloy modeling and execution to validate complex solutions. A cardiovascular disease public health case study based in Baltimore City will be used to demonstration the Modified System Development Life Cycle.

THE USE OF A MODIFIED SYSTEM DEVELOPMENT LIFE CYCLE (MSDLC) IN A SOCIOLOGICAL ENVIRONMENT TO IMPROVE SOLUTION VALIDATION

by

Shanelle M. Harris

A Dissertation Submitted in Partial Fulfillment

of the Requirements for the Degree

Doctor of Engineering

MORGAN STATE UNIVERSITY

May 2019

THE USE OF A MODIFIED SYSTEM DEVELOPMENT LIFE CYCLE (MSDLC) IN

A SOCIOLOGICAL ENVIRONMENT TO IMPROVE SOLUTION VALIDATION

by
Shanelle M. Harris

has been approved
March 2019

DISSERTATION COMMITTEE APPROVAL:

______________________________, Chair
LeeRoy Bronner, Ph.D., P.E.

____
Tridip Bardhan, Ph.D.

____
Yvonne Bronner, ScD, RN

____
Guangming Chen, Ph.D.

____
Ian Lindong, MD, MPH

DEDICATION

This journey is dedicated to every little girl. Not all doctors work in a hospital.

ACKNOWLEDGEMENTS

I would like to acknowledge Dr. LeeRoy Bronner, Dr. Tridip Bardhan, Dr. Yvonne Bronner, Dr. Guangming Chen, and Dr. Ian Lindong for being members of my dissertation committee. I would also like to acknowledge the librarians for their assistance in interlibrary loans.

This process has been a long journey coinciding with professional job progression. I would like to acknowledge my supervisors, Mr. Steve Chizmar, Dr. David Mortin, Mr. Myles Miyamasu, and Mr. Tom Tsoutis, for their encouragement and support during my process in completing this dissertation.

Lastly, I would like to thank my family, friends, coworkers, and BSF leaders for their continued prayers, interest in my research, and belief in me. To my mom, Tia Taylor, who has pathed a way to set an example of womanhood and continues to pour her love into me. She is my prayer warrior, strength, and champions all my goals. To Patrick Autry, for being my biggest supporter, listening to my daily events, and telling me I cannot fail.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Motivation and Problem Statement

Motivation

The National Academy of Engineering (NAE) states that as the population grows "the problem of sustaining civilization's continuing advancement, while still improving the quality of life, looms more immediately" also increases (National Academy of Engineering [NAE], 2018). Twenty–first century researchers, engineers, and scientist are facing an array of problems characterized by unstable elements, large of amounts of data, short timelines for solutions, multiple discipline areas, etc. (National Academy of Engineering [NAE], 2018; Numrich & Tolk, 2010). In February 2018, a group of engineers and scientists developed a list of engineering challenges of the 21st century. Included on this list were to secure cyberspace, restore and improve urban infrastructure, advance health informatics, reverse engineer the brain, and prevent nuclear terror (NAE, 2018).

Problem Statement

The traditional System Development Life Cycle (SDLC) methodology is inadequate for many 21st century problems characterized by complexity, dynamics, unstable objectives, and large amounts of data such as cyber security, Human, Social, Cultural and Behavior (HSCB), and big data (Numrich & Tolk, 2010).

Due to the increase in complex dynamic applications, new methodologies are needed to organize and maintain the applications. Dorsey (2000) concluded that to be successful some amount of planning is needed. The hypothesis of this dissertation is that the addition of joint application development (JAD) session, constant user involvement, a combined analytical method, and intermediate solutions will provide a validated model for analysis of a system leading to fewer inconsistencies than using an Iterative methodology.

1.2 Objectives

The purpose of this study is to research the current uses and weaknesses of the SDCL. From this research, the SDLC will be expanded for use in solving a 21st century problems. Object-oriented analysis, Z notation, and Alloy modeling and execution will be integrated to establish and strengthen complex problem solving. The value of intermediate solutions will be identified. Lastly, the improvements of the modified SDLC will be explained with respect to a validated model providing fewer inconsistencies.

Chapter 2 discusses in detail the history and evolution of the SDLC methodology. The state of research, methodologies, and gaps in SDLC models are described. The prominent weaknesses are explored, and the challenges faced due to the weakness. Twenty-first century problems will be defined.

Chapter 3 explains the modifications to the SDLC methodology including JAD sessions, continuous user involvement, and the three-phase analysis approach with object-oriented analysis, Z notation, and Alloy. A background is provided with examples of how to implement the additions.

Chapter 4 will discuss a sociological case study in detail. This case study will provide background information from the 2000 Heart Healthy Initiative to minimize the threat of cardiovascular disease in Baltimore City. The MSDLC construct will be used to highlight the improvements to the SDLC in minimizing the disadvantages and providing an accurate solution with use of predicate logic and set theory. Lastly, the technical contribution of the methodology and future research will be summarized.

Chapter 2: Literature Review

2.1 Introduction

The traditional System Development Life Cycle (SDLC) is used for large engineering and software projects. However, the problems facing society are beyond the scope of SDLC. The SDLC lacks the ability to be flexible, changing requirements, and lacks user involvement leading to less than adequate solutions. The purpose of this research is to develop a validated model and analysis procedure to provide solutions with fewer inaccuracies through an extension of the analysis phase and implementing Joint Application Development (JAD) sessions, continuous user involvement, and intermediate artifacts of analysis. The extended analysis phase integrates object-oriented analysis (OOA), Z notation, and Alloy modeling and execution to validate complex solutions. A sociological case study will be used to demonstration the application of MSDLC in planning a community-based participatory research initiative to minimize the threat of cardiovascular disease in Baltimore City (Harris & Bronner, 2018).

The SDLC is a term used by software and system engineers that is a conceptual model used in project management that describes the stages involved in a system development project, from an initial feasibility study through maintenance of the completed application (Avison & Fitzgerald, 2003; Bender RBT Inc., 2003; Dawson, Leonard, & Rahim, 2015; Council, 2014; Rouse, 2018). The SDLC is used for a variety of projects such as software development, education, military, medical, and political systems (Alexander & Maiden, 2004; Boehm & Hansen, 2001; Heller & Keoleian, 2002; Larman & Basili, 2003). However, with 21st century problems increasing, the boundaries have been reached and an improved approach is needed. The literature review will cover the evolution in the SDLC and the gaps.

2.2 Evolution of System Development Life Cycle

Through the evolution of the SDLC, there are three main categories: Waterfall, Iterative and Agile. These SDLC versions will be expanded upon in the next sub-sections.

2.2.1 Waterfall Model

The SDLC that was developed in the 1960s was designed to help with large business systems. The systems were heavily based on data processing and mathematical routines (Elliott, 2004). The Waterfall model was the first SDLC method employing a linear and sequential phase method. The sequential phase method had a set of defined outputs or deliverables that must be produced before the phase could be deemed complete; iteration was not permitted on previous phases (Avison & Fitzgerald, 2003; Rouse, 2007). However, there is extensive written documentation, reviews, and signoffs before the next can begin (Council, 2014; Davis, 2014). There was an emphasis on "planning, time schedules, target dates, budgets, and implementation of an entire system at one time" (Davis, p96, 2014). The Waterfall model is shown in Fig. 1.

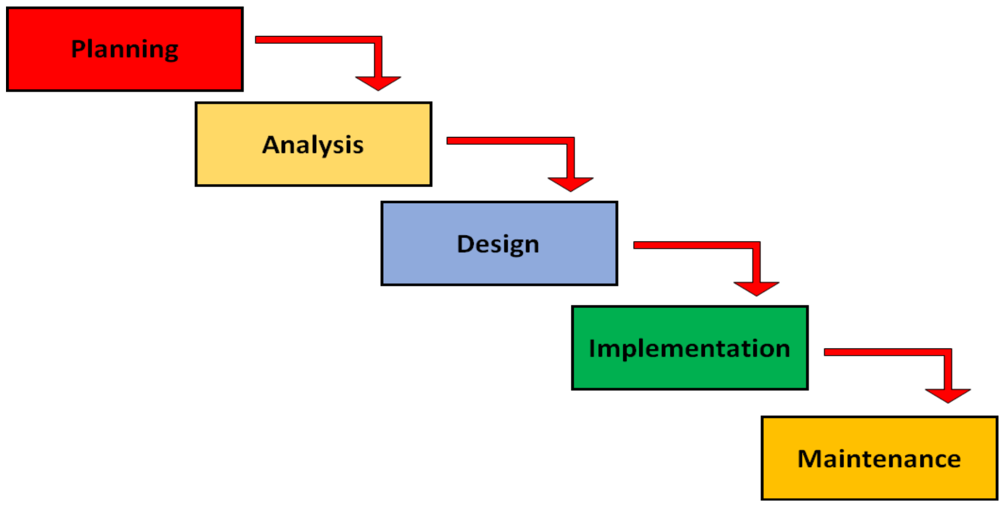


Figure 1: Waterfall Model (Salleh et al., 2011, p. 263)

As illustrated in Fig. 1, the waterfall SDLC has the following defined phases: planning, analysis, design, implementation, and maintenance. During the project planning phase, projects are chosen due to a need or improvement. The project is then broken into levels with an associated goal. The model moves into the analysis phase, where phase functions are defined based on the goals created in the project planning phase and the end user information is analyzed (Council, 2014).

Further refinement occurs during the design phase, providing detailed features and operations of the system. The execution of the features and operations from the design phase are conducted during the implementation phase. The implementation phase also ensures interoperability through testing and checking for errors. During the maintenance phase, any corrections are taken for action during the production and distribution of the project (Boyde, 2014; Council, 2014).

The Waterfall model was found to have limitations in manageability due to lack of iteration, inconsistent requirements, instability, gap between users and developers, failure to meet user needs, overly conservative system designs, difficulty responding to changes, user dissatisfaction, lack of documentation, and the user not obtaining the final product until completion (Avison & Fitzgerald, 2003; Council, 2014; Davis, 2014; Rouse, 2007; Saunders, 2014; Seema & Malhotra, 2012). The methodology did not provide for iterations thus causing complications as the waterfall approach continues. For instance, if the requirements are only discussed with the user during the requirement analysis phase, then the requirements are only partial. The requirements are partial because of what is defined by the user at that moment and what the analyst can interpret from the discussion. This can result in overly conservative system designs and a gap between users and developers. Not meeting with the developers during other phases, the user cannot confirm nor adjust what the analyst has interpreted after the requirement analysis phase. A misinterpretation has lasting effects on the preceding design, implementation, testing, and maintenance phases. This becomes exacerbated by not obtaining the final product until completion. This result in the user's dissatisfaction.

This approach is ideal for problems that are stable, predictable with known interoperability, well defined requirements, or updates to existing systems (Council, 2014; Davis, 2014; Seema & Malhotra, 2012). New large-scale projects require an iterative methodology allowing the user to see the product before final delivery and allowing for the return of previous stages (Wood & Silver, 1995).

2.2.2 Iterative Methods

Winston Royce documented the Waterfall method in 1970; however, the term was not mentioned in the article. It was meant to represent a flawed non-working model (Curtis, Krasner, & Iscoe, 1988; Ragunath, Belmourougan, Davachelvan, Kayalvizhi, & Ravimohan, 2010). This leads to the many failures, a need for a better methodology, greater flexibility, and quicker results (Khurana & Gupta, 2012). The next phase of SDLC development was built upon the biggest flaw, lack of iteration. From the flaw the Iterative methods were introduced in the late 1970s. According to Lehman (1969) and Larman and Basili (2003),

> the basic approach recognizes the futility of separating design, evaluation, and documentation processes in software-system design. The design process is structured by an expanding model seeded by a formal definition of the system, which provides a first, executable, functional model. It is tested and further expanded through a sequence of models, that develop an increasing amount of function and an increasing amount of detail as to how that function is to be executed. Ultimately, the model becomes the system (p3).

The Iterative method can be compared to looped waterfall processes, providing feedback to the next phase, and allowing the development team to provide results earlier in the process (Khurana & Gupta, 2012). Multiple methods are based on the Iterative approach such as Prototyping, Incremental, and Spiral. The first iterative documented project was the first U.S. *Trident* submarine. The project was broken into four time-boxed iterations each lasting about six months to manage complexity and risks (Larman & Basili, 2003).

2.2.2.1 Prototyping Approach

The Prototyping approach is a popular form of Iterative SDLC that produces a small prototype or version of the system that the user can work with to provide suggestions. The approach is not a standalone methodology but an approach to handling portions of the larger whole (Ateeq & Shuaib, 2014; Davis, 2014). The suggestions are then incorporated to make the system fully operational (Council, 2014). The Prototype approach is illustrated in Fig. 2.

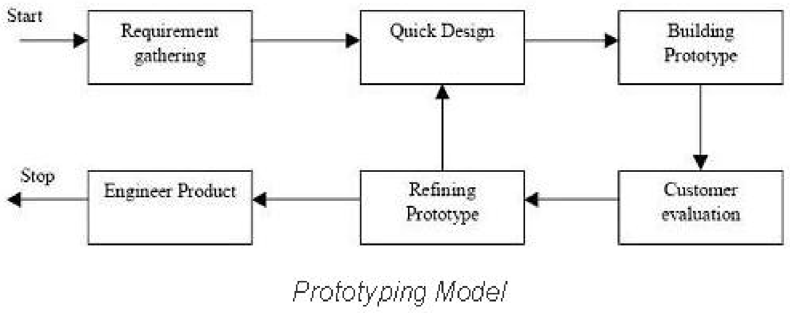


Figure 2: Prototyping Approach (Davis, 2014, p. 97)

The figure shows a loop through quick design, building the prototype, customer evaluation, and refining the prototype. This loop continues until the customer is satisfied with the prototype and refinements have been implemented. At that point, the prototype would then become the product. The Prototyping approach attempts to reduce the risk by having the project in smaller pieces to ease changes needed during the development phase (Davis, 2014).

The Prototype approach does allow for multiple iterations; however, the disadvantage occurs with the multiple iterations. It is assumed that the prototypes will be discarded and unsuccessful. This assumption is partially due to knowing the requirements can change drastically in the next iteration (Council, 2014). For example, the user could require a new feature after several prototypes. The new feature can change the scope of the problem leading to scope creep. This leads to a waste of time and money. Due to the waste of time and money, this approach is not suitable for large-scale projects (Ateeq & Shuaib, 2014). Multiple prototypes are also a management disaster. The multiple changes to satisfy the user not only are difficult to manage but also disrupt the development team (Ateeq & Shauaib, 2014; Kumar, Zadgaonkar, & Shukla, 2013). Multiple prototypes in the design phase are overwhelming, adds complexity to maintainability, and can cause scope creep (Seema & Malhotra, 2012).

The Prototyping approach is best used for short-lived demonstrations or systems that have not been developed (Davis, 2014). These types of systems can begin the foundation due to instability in a new system.

2.2.2.2 Incremental Approach

The Incremental approach is mix of the Waterfall and Prototyping approaches. The Incremental approach develops, implements, and tests a system incrementally until the product is completed, like staggered waterfalls (Council, 2014; Seema & Malhotra, 2012). The Incremental approach is shown in Fig.3 having a staggered analysis, design, coding, and test processes of multiple increments.

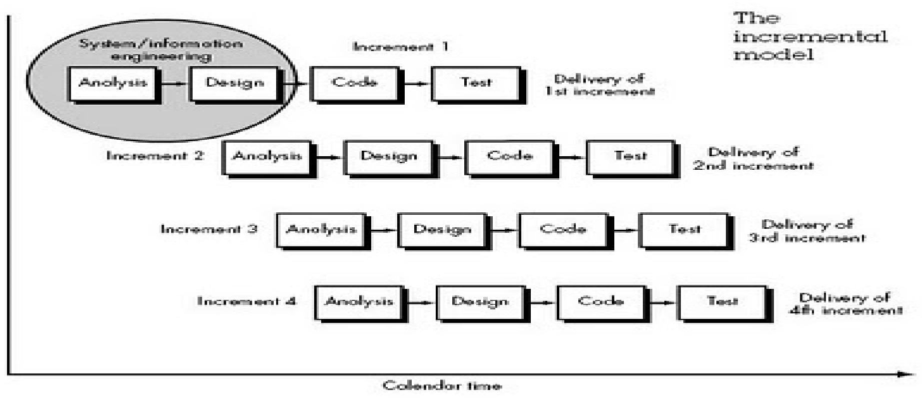


Figure 3: Incremental Approach (ARS, 2010)

Through the series of events, each increment prioritizes the system's requirement. Partial implementations are constructed of a whole system and functionality is added slowly (Seema & Malhotra, 2012). Hence, each increment builds upon functionality until all are implemented. A time reduction occurs due to the user obtaining parts on a piece-wise timeframe (Council, 2014).

The Incremental approach continues to be rigid within the iterations due to the phases not overlapping (Council, 2014; Seema & Malhotra, 2012). The same prominent flaw found in the Waterfall model. This approach also is based upon all requirements being defined completely and early in the process (Seema & Malhotra, 2012). If the requirements are defined incorrectly, the continual builds will result in a product that the user does not want or does not meet requirements.

The Incremental approach is ideally used on projects with known requirements that can evolve over time. Seema and Malhotra (2012) state that the Incremental approach can be used on programs that have a short time to market if it possesses basic functionality or with an exception for a program with a longer schedule. This exception is allowed because the realization of all requirements will occur over time.

2.2.2.3 Spiral Approach

The Spiral approach also expands on the Waterfall and Prototyping approaches by using smaller segments during the development process maintaining a focus on risk assessment (Ateeq & Shuaib, 2014; Council, 2014; Davis, 2014; Khurana & Gupta, 2012). The Spiral approach is shown in Fig. 4.

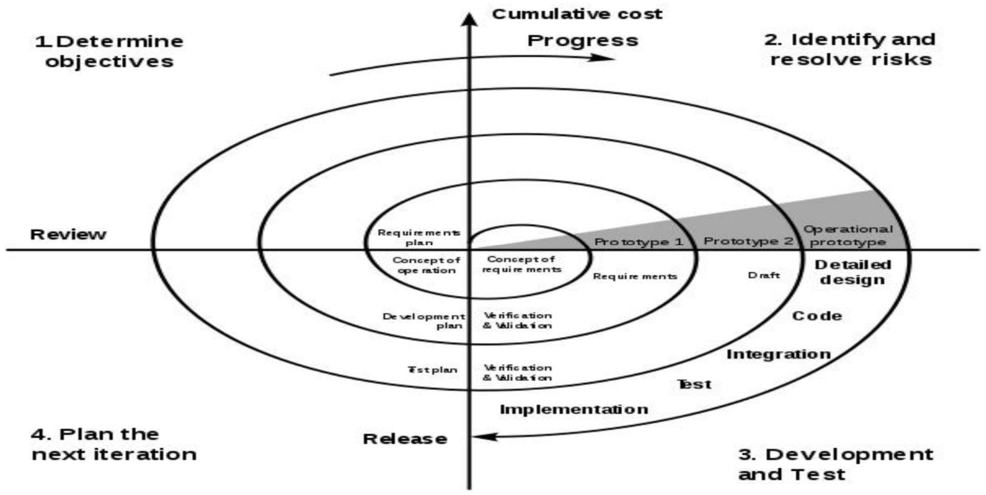


Figure 4: Spiral Approach (Boehm & Hansen, 2001, p.4)

The Spiral approach has four main phases: planning, risk analysis, engineering, and evaluation as presented in Fig. 4, phase one through four, respectfully (Ateeq & Shuaib, 2014; Easterbrook, 2001; Khurana & Gupta, 2012; Park, Ali, & Chevalier, 2011). The phases in the Spiral approach involve (1) determine objectives, requirements, alternatives, constraints, and begin risk assessments; (2) evaluate alternatives, identify and resolve risks, and produce a prototype; (3) produce code, test, implement, and verify prototype; and (4) plan the next iteration after the user evaluates the program (Davis, 2014; Easterbrook, 2001; Khurana & Gupta, 2012; Park et al., 2011). Cost is represented as the radius and progress is shown with angular components.

Building upon the Waterfall and Prototyping models, the Spiral approach can reduce issues of iteration but requires expertise in risk identification, risk projection, risk assessment, and risk management. The associated cost of the risk analysis makes this approach not ideal for small systems. If used on a smaller project, the risk analysis cost would be greater than the entire system cost (Ateeq & Shuaib, 2014; Davis, 2014; Seema & Malhotra, 2012). This approach also requires an experienced project manager to determine the application of the methodology to the system since there are not established controls for moving forward in the approach (Ateeq & Shuaib, 2014; Council, 2014; Seema & Malhotra, 2012).

The Spiral approach is best used for larger projects with a medium to high risks, and/or with requirements that are complex, and the user is unsure about the needs (Council, 2014; Seema & Malhotra, 2012). The extensive risk analysis will assist in reducing the uncertainty in the user's needs. The preliminary design phase allows all possible alternatives to be analyzed. By analyzing the alternatives instead of prototyping the alternatives, the project cost is reduced (Council, 2014).

The Prototype, Incremental, and Spiral approaches are all examples of the Iterative method. These methods are best used on project that have better defined requirements and known interoperability, allowing the project to be broken into small pieces (Council, 2014). The Iterative approach did expand on the waterfall approach; however, prototyping can occur too quickly, testing can be minimized, iterative phases may not overlap, all requirements may not be gathered upfront, and features may be added over iterations

(Ateeq & Shuaib, 2014; Council, 2014; Davis, 2014; Kumar et al., 2013; Seema & Malhotra, 2012).

2.2.3 Agile Approach

The Iterative methods allowed for recursive states during the SDLC. The iterative phases alone and multiple instances were not enough for the types of problems facing industry by the 1990s. Agile prioritizes user satisfaction along with building up the Iterative and Incremental approaches. This allows for the requirements and solution to evolve using iterations of products that include all the tasks and functionality for a release (Cockburn, 2002; Council, 2014, Dawson et al., 2015; Sharma, Sarkar, & Gupta, 2012; Szalvay, 2004). The Agile approach is shown in Figure 5.

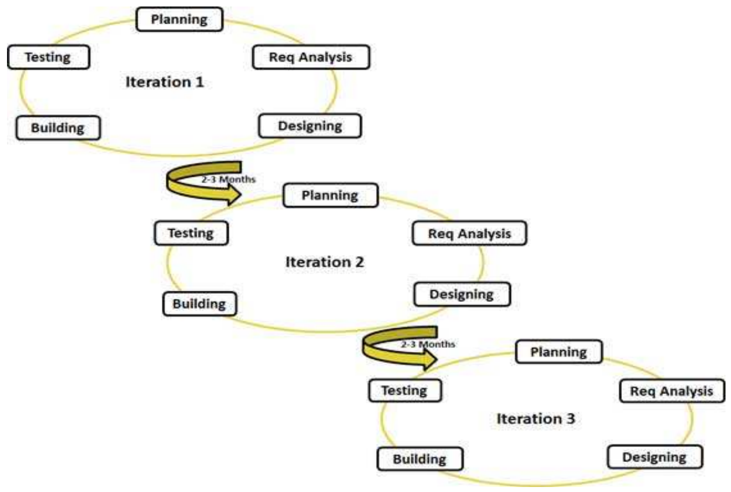


Figure 5: Agile Approach (Tutorialspoint.com, n.d.)

The Agile process has iterations that go through the planning, requirements analysis, design, build, and test phases. An iteration takes two to three weeks before another begins. This idea is built upon the slogan of "build short build often" (Council, 2014). The Agile approach initiates with one distinct difference: user complete requirements are not expected before the project is to begin. Communication with the user is a priority and is conducted throughout the cycle (Sharma et al., 2012). Agile is dependent on user feedback and the phases of the life cycle are revisited continuously (Leau, Loo, Tham, & Tan, 2012; Szalvay, 2004). The iterative nature of the process allows requirements to be implemented later or upon the availability of technologies as a solution. Ideally, the approach will produce a product every three to four weeks because it is divided into small increments instead of one large model allowing the system to be customizable (Cho, 2008; Leau et al., 2012; Sharma et al., 2012).

A lack of willing interactive users can limit Agile. The foundation of Agile is having the user available to make decision on requirements and the approval of iterations. Since communication is a priority, documentation usually is forfeited. Comments in code are used for explanation but do not suffice the traditional means of providing guidelines and clarification to the system (Leau et al., 2012; Sharma et al., 2012). An extension of constant communication with the user is the waste of resources and time consumption in the changing requirements. If the user changes a requirement, then the iteration is not useful, including the "time, effort, and resources required to develop that increment" (Sharma et al., 2012).

Agile is best used for smaller projects. The small increments used on larger projects typically increase time on effort and distort efforts (Balaji & Murugaiyan, 2012; Seema & Malhotra, 2012).

2.3 SDLC Limitations

The response to these disadvantages was the creation of new variations of the SDLC. Since its induction, there have been multiple adjustments for specific problem types. Each variation has a specific case for usage, requirements, advantages, and disadvantages. Each improvement leaves the developer with multiple choices and the decision to leverage limitation.

The SDLC not only has multiple methods but has a failure rate as high as 80% (Dorsey, 2000). Failure is greatly based on scope creep, unclear requirements, and lack of methodology (Baltzan & Phillips, 2010; Dorsey, 2000; Jones, Software Cost Estimating Methods for Large Projects, 2005; Saunders, 2014). Jones (1998, 2005) explains that 60% of defects are introduced in the requirements and design phases, referred to as the 60% rule. Unclear requirements can cause multiple problems further down the life cycle.

Baltzan and Phillips (2010) reveal that the most common reason systems fail is due to missing or incorrectly gathered requirements in the analysis phase. Since requirements are the driver for the system, incorrectly or inaccurately defining the requirements is a poor basis/foundation for a program. This also coincides with the least expensive phase to adjust versus after the development phase.

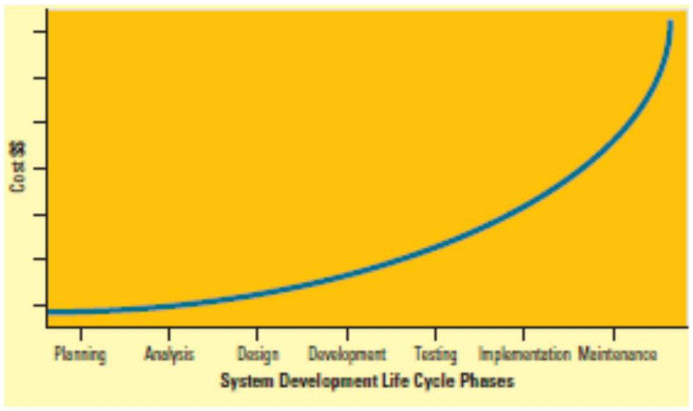


Figure 6: Cost of Finding Errors in System Development Life Cycle by Phase (Baltzan & Phillips, 2010, p. D16)

Figure 6 displays an exponential cost growth for errors found in the SDLC phases. The further along one is in the SDLC, the cost to find errors is exponential compared to the previous phase. Bunting (2012) also found that the largest portion of cost over-runs (32%) was due to missing or incorrect requirements as seen in Figure 7. This finding is based on industry metrics, collected data, and project experience.

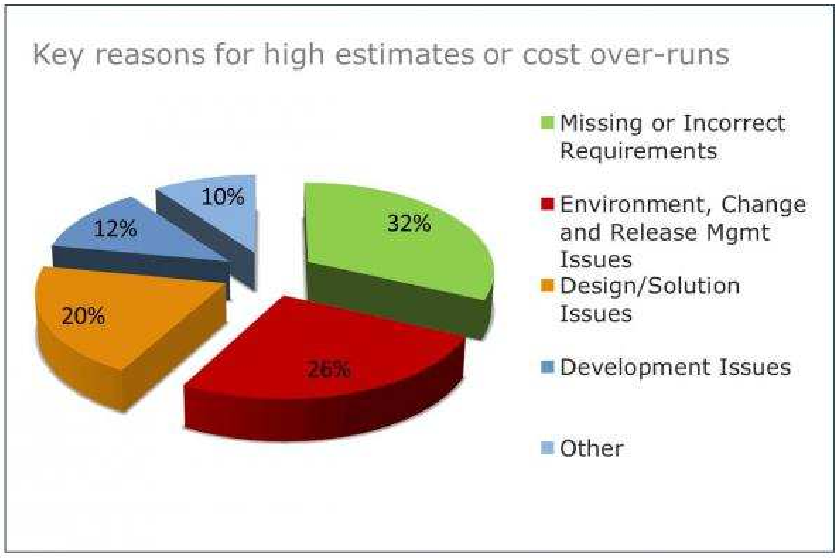


Figure 7: Key Reasons for High Estimates or Cost Over-runs (Bunting, 2012)

Not clearly stating requirements can introduce a phenomenon called scope creep. Baltzan and Phillips (2010) define scope as a collection of requirements. Scope creep is the small changes in the requirements that steer the project in a different direction than what was initially started. The resulting product does not match what the user wants. This can become a significant problem in the Iterative and Agile approaches. Boehm and Hansen (2001) state that due to the nature of iterating, rapidly changing requirements can be the norm for volatile technology and a high marketplace. Boehm and Hansen (2001) also suggest that, in the case of electronic commerce projects, some requirement decision should be defined later in the process. Stating all the requirements upfront proves to be a hindrance for the projects since the time-to-market is scarce and the requirements are likely to change later (Boehm & Hansen, 2001).

The issues resulting from unclear requirements have a foundation in lack of communication and lack of user involvement. Nagpal and Chawla (2012) have found that errors are introduced into the development due to ambiguity, assumptions, and flawed human communication. This occurs by a change of functionality being introduced. The test-fix-test approach is then used to find errors. However, with each addition, there is an increasing risk that the purpose will be lost. Avison and Fitzgerald (2003) have discovered that programmers typically developed individually without the user. This development style leads to poor project management and failure to meet the user’s requirements. The developer may encounter a user with the “I know it when I see it” (IKIWISI) syndrome. Boehm and Hansen (2001) have documented that this occurs with new systems. This syndrome enforces the need for continual user communication. Boehm and Hansen (2001) and Saunders (2014) have found that omitting the user involvement between each phase leads to risks not being detected and system failure. Boehm and Hansen (2001) further explain that not having user participation in early stages of the SDLC can result in a lose-lose situation. The developer loses because time and money has been wasted. The user loses because the resulting product is based on unrealistic assumptions.

Another problem with the SDLC is the abundance of time, energy, and resources spent on maintenance. Tagoug (2012) has found that “large companies currently expend 50 to 70 percent of all programming effort on maintenance.” Bender RBT Inc. (2003) and Dorsey (2000) have documented that the maintenance phase can take up to 80 percent of a project's effort. With the growing cost of maintenance, companies need a method to improve the logic of their projects before the maintenance phase is reached.

Baltzar and Phillips (2010) have found that when programs are behind schedule, phases in the SDLC are skipped. The lack of complete and understood requirements can have lasting effects on development and testing phases. For example, a delay in requirements of one week can lead to a two-week delay in development. The two-week delay in development and one-week delay in requirements can lead to a three-week cut in testing. The amount of testing planned cannot be completed after three weeks have been removed. As a result of not completely testing a program, it is likely not all errors will be found and increasing risk of failure.

2.4 21st Century Problems

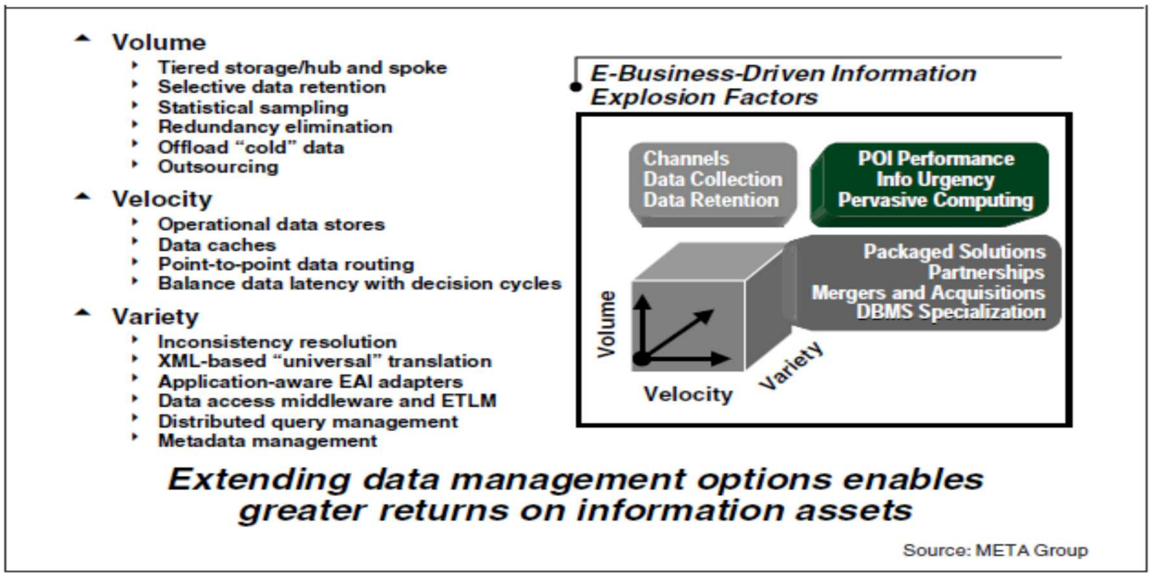
2.4.1 Big Data

Ammu and Irfanuddin (2013) explain big data as an overarching term used for an immense quantity and variety of high-frequency data. Big data can come in multiple forms, i.e., online user-generated content, medical records, blogs, banking transactions, images, or online searches. The goal is to provide decision makers information that can track changes, improve social programs, implement new features, or decide the next menu item (Ammu & Irfanuddin, 2013). The problem is how does one take this data and convert it into relevant information.

The International Data Corporation (IDC) determined that there were only 132 exabytes of available digital data in 2005. Due to the increase in smart devices, connectivity to the Internet, and the number of people online, the growth is estimated to increase 40% a year or almost double every two years. At this rate by 2020, available digital data could surpass 40 zettabytes (trillion gigabytes; ICD,2014).

With the increase in data, Adrian (2013) states, there is a feat to undertake by "speeding up" internet connections and developing software for the load. The growth of data is also pushing the limits of traditional information retrieval. Database system analysis is limited due to lack of coordination. The system analysis packages not only perform SQL querying but data mining and statistical analyses. These additional processes require a longer time to find the specified criterion and analyze the data (Ammu & Irfanuddin, 2013). This time does not exist. The decisions makers still require information immediately. However, if there is a "lack of skills in sorting, big data will most certainly conclude into faulty results and/or truncated data that cannot serve its purpose" (Adrian & Irfanuddin, p34, 2013).

Laney (2001) discusses an approach to manage data in terms of volume, velocity, and variety shown in Figure 8. These factors are alternative solutions to combat big data problems. Determining the combination of actions to provide a robust solution set is the problem to solve.



- Volume
  - Tiered storage/hub and spoke
  - Selective data retention
  - Statistical sampling
  - Redundancy elimination
  - Offload "cold" data
  - Outsourcing
- Velocity
  - Operational data stores
  - Data caches
  - Point-to-point data routing
  - Balance data latency with decision cycles
- Variety
  - Inconsistency resolution
  - XML-based "universal" translation
  - Application-aware EAI adapters
  - Data access middleware and ETLM
  - Distributed query management
  - Metadata management

Extending data management options enables greater returns on information assets

Source: META Group

Figure 8: Data Management Solutions (Laney, 2001)

With the amount of data ever-increasing and the sources of data ever-changing, the problem has compounded. There is a need for a continuous logical method that intelligently compiles, computes, and explains the data. The use of a modified system development life cycle will assist in documenting a data dictionary through using OOA, providing a logical basis through using Z notation, and the execution of Alloy to validate the logic needed to manage the logic for such a dynamic system.

2.4.2 Human, Social, Cultural, and Behavioral Modeling

The Office of Naval Research (ONR) has a vision that Human, Social, Cultural, and Behavioral (HSCB) Modeling is "mastery of social, cultural, and cognitive factors that optimize the warfighters ability to influence human behavior in the full range of military operations" (Appleget, 2010). Objectives are in four areas: tools, methods, training, and development. With the change from force-on-force battle, a switch in focus has provided a wider picture of the operating environment including the human experience. There is a need to assist the Soldier in decision making in the ever-changing space. Ideally, human science models would be used and injected into the kinetic model, but the challenges are great (Numrich & Tolk, 2010). The requirement seems simple but is difficult.

The need for HSCB modeling comes from the new fighting method of the Department of Defense (DoD) — irregular warfare. No longer are wars fought force on force with the use of a battlefield. The new fight is a mix of technology and socio-cultural understanding (Biggerstaff, 2007; Estabrooke, 2009). The problem space for HSBC is shown in Fig. 9. Figure 9 shows a space with desired effects and multiple models needed to reach the effect. The political, military, economic, social, infrastructure, and information systems (PMESII) axis describes the state of the situation; while the Diplomatic, Information, Military, and Economic (DIME) axis describes level of power to influence the PMESII (Hartley, 2009). Some of the models referenced in the figure are population, information, military effect, and economic models all influencing if the desired effect will occur. There is a need and demand for a capability to understand social and cultural terrain and the various human behavioral dimensions (Biggerstaff, 2007).

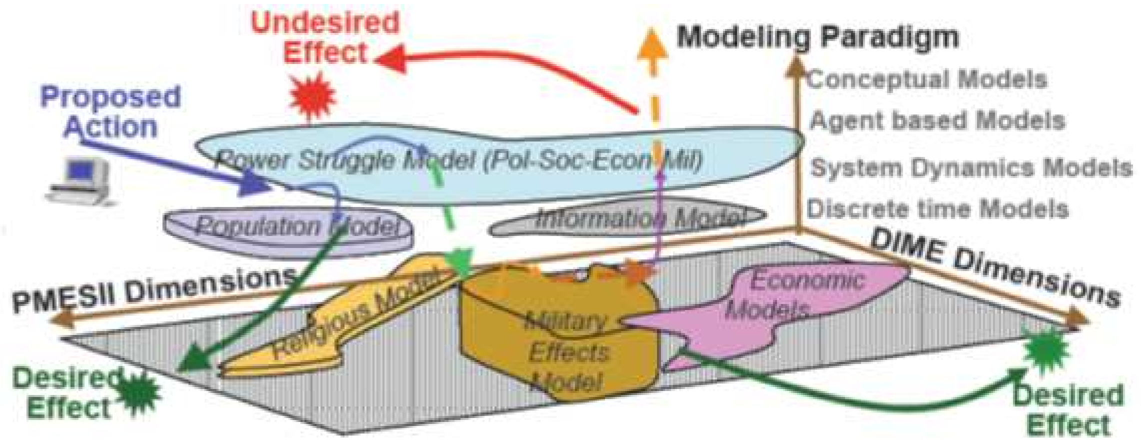


Figure 9: HSCB Complex Problem Space (Biggerstaff, 2007)

It is likely the need for HSCB modeling appeared in an Operational Needs Statement from February 2006 from Maj. Gen. L. Freakley, Commanding General of the 10th Mountain Div., stating “must develop the ability to understand the complex human factors and must incorporate them into all facets of operation” (Biggerstaff, 2007). There was no existing U.S. technical HSCB core leading to a need for government, laboratories, academics, and industry to assist in the solution (Biggerstaff, 2007). MSDLC is a useful tool for HSCB by providing a structure for all communities to interact and explain the problem through conducting a JAD session. OOA artifacts will provide a universal understanding of goals, models, and procedures that the community can expand upon on future work. The use of Z notation and Alloy execution will provide feedback concerning the logic of the OOA structure developed for analysis.

2.4.3 Public Health: Measles

Measles is an airborne-transmitted, highly infectious disease, eliminated from the United States in 2000. Eliminated is defined as an “interruption of year-round endemic measles transmission” (McLean, 2012, p.253).

Hungerford, Cleary, Ghebrehewet, Keenan, and Vivancos (2013) examined an outbreak of measles in Merseyside, United Kingdom in 2012 with an aim to identify associated risk factors of transmission using retrospective matched case studies, univariate, and multi-variate analyses. The findings emphasize the need for timely vaccinations, early diagnosis, isolation, and control measures. Hungerford et al. (2013) also identified independent associations between measles infections and hospital admission.

Helmecke, Elmendorf, Kent, Pauze, and Pauze (2014) describe the impact of two measles cases in the Albany Medical Center during May 2011. Helmecke et al. (2014) successfully did not transmit the disease at the facility. The staff accounts this to facility "rapid case recognition, isolation, health care worker immunity, and multidisciplinary response." The team did experience challenges between the Healthcare Infection Control Practices Advisory Committee and public health guidelines for measles control on the issues of: (1) lack of standard definition for exposure and immuno-suppression; (2) inconsistent application of exposure definition; (3) post-exposure recommendations variation; (4) inconsistent practices; and (5) verbal history instruction and practices.

Hungerford et al. (2013), Vivancos, Keenan, and Farmer (2012), and Weston, Dwyer, and Ratnamohan (2006) have found that the transmission of airborne diseases is high in healthcare settings, and it is difficult to discern measles from other viral infections. Helmeck's et al. (2014) case study shows the benefits of early discernment. How can healthcare facilities implement safety precautions for a disease that is eliminated? Should protocols automatically assume a person has an airborne disease? These questions provide a use for MSDLC. The implementation of MSDLC brings a logical verification for an implementation plan. The use of OOA producing use case models and object diagrams can be translated into parameters that drive the development of a Z notation model. The Z notation model can then be verified with the use of Alloy model and execution providing confidence in the implementation plan. The use of the MSDLC provides cost and time savings of implementing one method followed by multiple changes. The finite changes can be modeled before implementation.

2.5 Analytical Models

To reinforce the limitations of the SDLC a various of methodologies can be used. For the purpose of this research a mathematical model and analyzer have been researched to assist in minimizing the cost over-runs stated by Bunting (2012) as missing or incorrect requirements and Baltzan and Phillips (2010) mentioning missing or incorrectly gathered requirements in the analysis phase.

2.5.1 Z notation

The Z notation is a formal mathematical notation for modeling the behavior of systems (Spivey, 1989). Woodcock and Davies (1996) describe two benefits of using Z notation: documentation and cost savings.

Woodcock and Davies (1996) found that "important information is hidden amongst irrelevant detail, and design flaws are discovered too late, making them expensive or impossible to correct." It is also suggested that using formal methods can result in a precise, structured documented product with an appropriate level of abstraction. The resulting documentation can support the phases following system analysis. (Woodcock & Davies, 1996).

With respect to cost savings, the use of proofs provides "mental checks during reviews" allowing for errors to be corrected earlier leading to a dividend in productivity (Baltzan & Phillips, 2010; Bunting, 2012; Woodcock & Davies, 1996). Woodcock and Davies (1996) state the "mathematical basis has a purpose: to add precision, to aid understanding, and to reason about properties of a design".

2.5.2 Alloy Language and Analyzer

To continue to eliminate limitations in the SDLC, analyzing the models will assist in with the validation and reducing errors. Alloy language describes in time and space structures in a similar was as Z notation. The difference is that Alloy language is simpler than Z notation (Dwivedi & Rath, 2012). The Alloy Analyzer translates constraints to be solved from Alloy into Boolean constraints (Jackson, Software Abstractions, 2012). The Analyzer works to find all instances that violate declaration while satisfying all other constraints (Dwivedi & Rath, 2012). Jackson (Software Abstractions, 2012) believes that with the continued use of Alloy Analyzer, not only will more errors be found but also more succinct and elegant.

Chapter 3: Methodology

3.1 System Development Life Cycle Modification Additions

In chapter 2 the development, changes, and limitations of the SDLC were discussed. Chapter 3 will focus on the added modifications to strengthen the limitations to provide a robust and validated solution before entering the system design phase.

3.1.1 Joint Application Development Session

Traditionally the SDLC begins with a problem statement. However, to modify the SDLC a Joint Application Development (JAD) session will be held before a problem statement is solidified. A JAD session is a meeting that is held over a period of days, allowing all key parties to meet to discuss expectations for the project with the goal of accelerating the development process, increasing productivity and quality, and building developer-client relationships (Shelly & Rosenblatt, 2012; Thierauf, 1999).

Typical participants in the JAD session are the project leader, managers, users, system analysts, and recorders (scribes). The project leader will have the role of facilitator, ensuring the flow of discussion and leading the discussion of the project. Managers participate to show support for the project and provide authorization. The user provides operational-level input, leading to requirements for the project. The analysts are participating to provide technical assistance. The recorder takes notes of discussion, results from the JAD session, and actionable items to be executed post-JAD (Shelly & Rosenblatt, 2012).

The purpose of this meeting is to start developing ideas through phases. JAD sessions have a five-phase process: project definition, research, preparation, the session, and the final document (Wood & Silver, 1995). Table 1 displays the phases and the resulting output of the JAD Session.

Table 1: Joint Application Development Phases (Wood & Silver, 1995, p. 10)

Number	Phase	Resulting Output
1	Project Definition	Management Definition Guide
2	Research	Data models Process Models Preliminary Information Session Agenda
3	Preparation	Working Document Overhead, flip charts, magnetic
4	The Session	Scribe notes and scribe forms
5	The Final Document	JAD Document Signed Approval Form

From the session, participants/team members will construct an outline of the project definition (Wood & Silver, 1995). An initial high level system diagram (HLSD), a narrative of the initial problem definition, will result from the discussion and possible problem solutions. The HLSD is a visual depicting the problem. The visual defines the problem and the interactions between the users and any entity. The HLSD provides a clear and quick understanding of the problem.

Because of the JAD session and HLSD, participating parties have a clear initial problem definition and a path forward. The committed group now has a system design process with formulated scope (Thierauf, 1999; Wood & Silver, 1995)

As an example of a software development project HLSD is shown in Fig. 10. The diagram shows the organization of events that can occur in a kitchen while cooking. Tarkan (2009) uses a kitchen environment to teach children how to program. From the figure, one can start to understand steps involved in cooking a meal and the importance of a HLSD.

The article contains an outline of events, tools, measurements, and appliances in a created world to teach children how to cook without the presence of an adult.

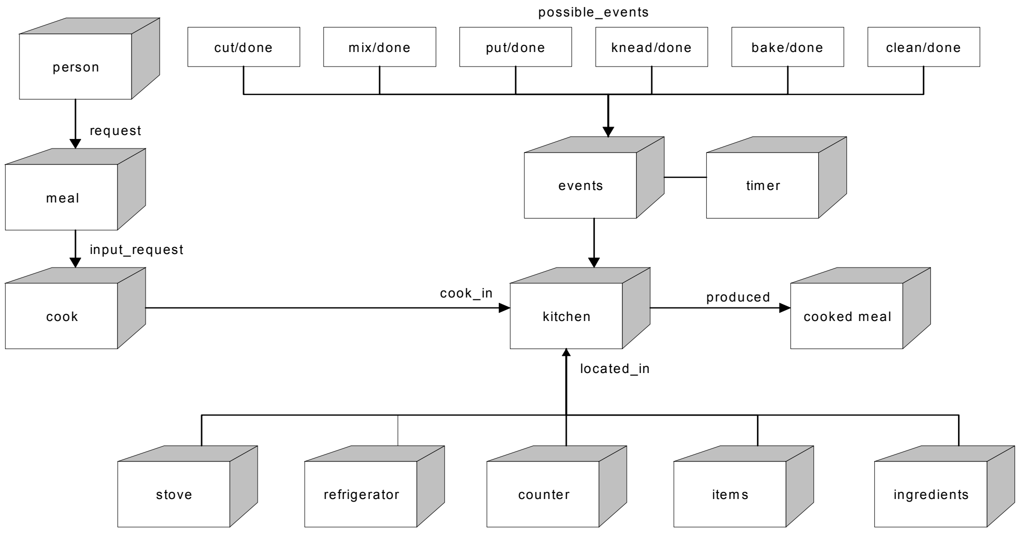


Figure 10: Tarkan's Kitchen Environment High Level System Diagram

The JAD session minimizes unclear requirements and specifications by having those interested parties in one location discussing the topic. The session is an initial step to meeting users' needs by having the users participate in the meeting. The scope of the problem will begin shaping as the problem is defined, leading to a framework of the project. As a result, the JAD session provided a commitment, group cohesion, and productivity to the systems design process (Thierauf, 1999; Wood & Silver, 1995).

3.1.2 User Involvement

An implementation of constant interaction with the user has been added to modify the SDLC. In the past, the users' role was relatively passive in the system development, leaving the IT department with the sole responsibility of systems development (Shelly &

Rosenblatt, 2012). The modification intentionally involves users in each step of the MSDLC.

User involvement in entering and exiting each step minimizes the risk for scope creep and unclear requirements which leads to a product that is likely to meet users' needs. The user involvement also allows the user to have a sense of ownership of the project (Shelly & Rosenblatt, 2012). The implementation of user involved cannot be overemphasized.

3.1.3 System Analysis Phase 1: Object-Oriented Analysis (OOA)

Object-oriented analysis (OOA) was first introduced in the 1990s as a solution to the structured approach for system analysis. Originally, the functional view was the focus; however, with every change, there was a change in the analysis, model, and implementation. With an increase in dynamics and complexity, the structured approach became outdated (Brown, 2001; Tsang et al., 2005). As a result, the OOA became widely popular with objects having the ability to interact with other objects by sending and receiving messages. The once troubling update process has been simplified by allowing the object's data to be manipulated while messages are being sent (Brown, 2001; Tsang et al., 2005). The approach does not only cover the analysis but goes through the development process, making it easier to make consistent models. The object-oriented approach is more stable than the structured approach because changes in the objects are localized, meaning that changes occur in one place instead of descending through the approach. The structure is flexible enough such that the top-down approach can also be used (Brown, 2001; Tsang et al., 2005).

OOA is used in Phase 1 of the MSDLC methodology. After the problem has been defined and the HLSD has been approved by the user, the use case modeling will precede. From the problem definition, requirements will be elicited. Requirements state what a system is to do, not how the system should implement the tasks (Tsang, Lau, & Leung, 2005). The requirements and multiple meetings with the developers and users should provide clarity of what the system is to do and remove uncertainty.

Use case development identifies and defines requirements by capturing scenarios (Tsang et al., 2005). Use cases are transactions performed that produce a measurable result, behaviors seen from the user's point of view; all possible functions (events) provided by the system as a set of events, yielding a visible result (Bruegge & Dutoit, 2004; Tsang et al., 2005). The building of use cases also provides a baseline of participants, elements, activities, and a description of what is to be achieved.

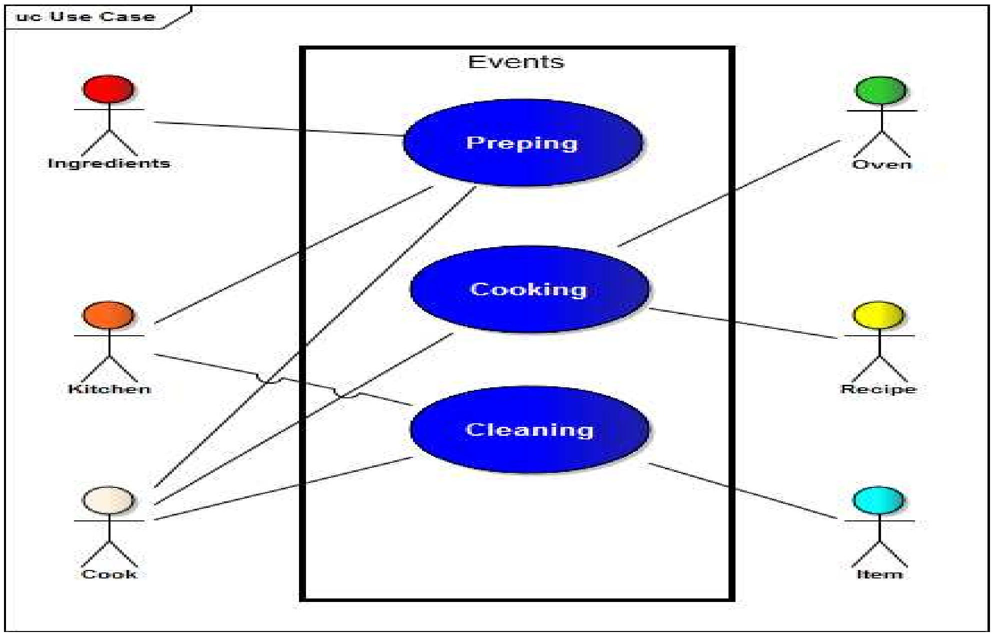


Figure 11: Tarkan Cooking Use Case Diagram

A collection of use cases will provide documentation of what the system will do and what the user is expected to see. Use cases do not provide a method for how the system will do the action. Figure 11 shows some use cases that have been derived from Tarkan's (2009) thesis. In the kitchen one can cook, prep, clean, or return items.

To continue defining the problem, a problem vocabulary is developed. The vocabulary includes the objects, attributes, and behaviors of the model. The objects, attributes, and behaviors are found through the use case scenario process. The objects will be clearly defined as things needed to represent entities in the system. The attributes are characteristics of an object such as name, address, or age. Behaviors are functions that the object can perform such as cook, drive, or withdraw. The products of this analysis are intermediate artifacts such as class, object, rationale, sequence, activity, and state diagrams. The diagrams are produced using the unified modeling language (UML) (Bruegge & Dutoit, 2004; Tsang et al., 2005). The Object Management Group (OMG) (2015) states that

> . . . the UML is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components (OMG, 2015).

The Enterprise Architect (EA) tool is used to create UML models. EA also makes it possible to archive the problem definition and solution, which maintains a complete index for searching and accessing the model components (Spark, 2015). The set of artifacts produced by the OOA being addressed by this research are outlined below.

Class/Object diagrams: Classes and objects are entities used to define elements in a problem space. Classes are used as a template to define objects. Objects are individually identified entities (Bruegge & Dutoit, 2004). Classes are determined from the use case scenario of the use case model. Objects are developed from the classes (Tsang et al., 2005). The resulting class/object diagrams will have the attributes and behaviors defined with relationships between the classes/objects. The diagram visually connects the classes/objects according to these relationships. Figure 12 shows classes such as item, kitchen, ingredients, cook, recipe, and timer. All the classes are connected by relationships. For instance, ingredient is in the kitchen and it can be said that the cook makes food with the ingredients. Each class has associated attributes and behaviors. The attributes are noted with the red lettering, while the behaviors are in blue lettering. The attributes define the state of the class/object. The behaviors are the actions of the class/object (Tsang et al., 2005). For example, food can be hot, cold, raw, etc. Behaviors that food can take are roll off counter, rise in oven, etc. The cook will influence the state of the food, since the cook can make food with the ingredients.

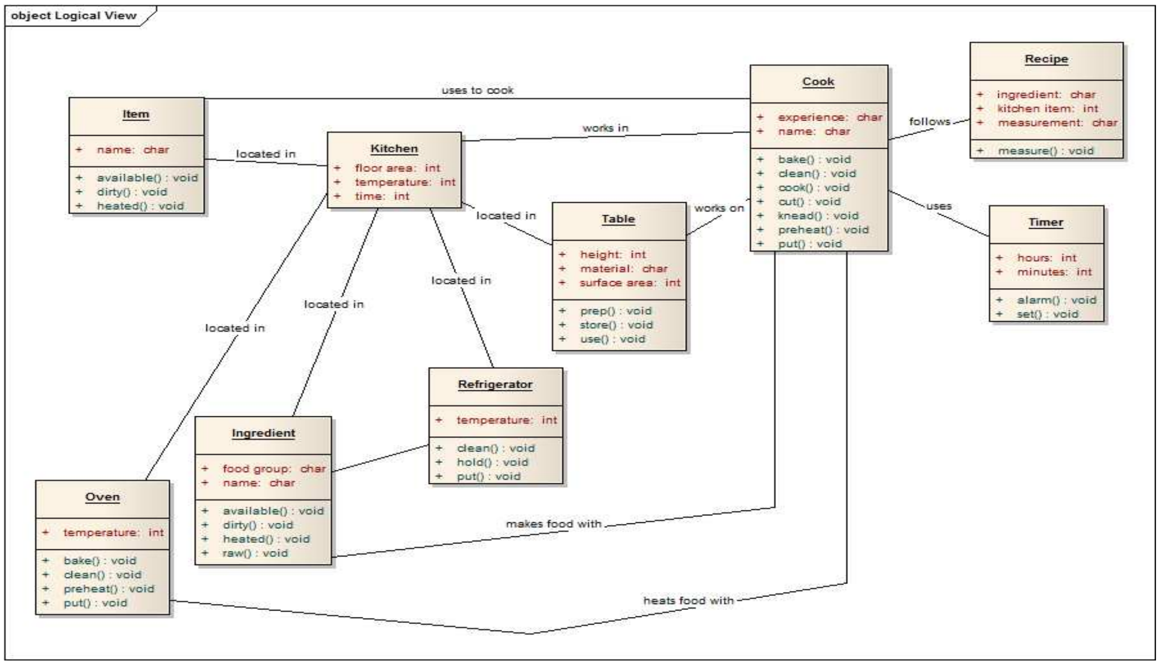


Figure 12: Class Diagram of Tarkan's work

Rationale diagrams: Rationale diagrams represent the reasoning that leads to the system's functionality and implementation. This diagram supports decision making and captures knowledge (Bruegge & Dutoit, 2004).

Sequence diagrams: Sequence diagrams are interaction diagrams that model the behavior of a group of objects working together to achieve a goal (Bruegge & Dutoit, 2004; Tsang et al., 2005). The diagram shows the passing of time on the vertical dimension and the interaction amongst objects on the horizontal dimension. Sequence diagrams assist with the continual definition and clarification of object's function.

Activity diagrams: Activity diagrams assist in modeling the state of an object and the transitions in response to events (Tsang et al., 2005). Specifically, it models the performance of actions in a process or procedure. Activity diagrams can expand upon the use cases to provide detail and confirmation of organization of events.

3.1.4 System Analysis Phase 2: Z notation

The Z notation is a highly expressive formal mathematical notation for specifying and designing the behavior of systems (Spivey, 1989). It is based on set theory and a typed first-order predicate logic model. "Typed" means that variables in Z cannot be defined without knowing the range of values that it can hold and, once the variable is declared, its type cannot change (Spivey, 1989).

Shen (2002) states that set theory deals with sets, their operations, relationships, and statements about these relations. Weiss (2008) defines set theory as the true study of infinity. This alone assures the subject of a place prominent in human culture. Sets are the collection of different elements (Drake & Singh, 1996).

Gries and Schneider (1993) state that "predicate calculus is an extension of the propositional logic that allows the use of variables of types other than Boolean. This extension leads to logic with enhanced expressive and deductive power." Predicate calculus has a foundation of Boolean expressions. Boolean variables are variables that can hold one of two values: true or false. Predicates use the Boolean functions, but the type is different than true or false (Gries & Schneider, 1993). Predicate calculus "usually consist of inputs, outputs, and changes to the state of the system. The relationship between input, output, before-state, and after-state will be described by a predicate relating and constraining these values" (Goldrei, 2005).

Z notation is neither a natural nor a programming language; however, it shares some of the characteristics of both languages. Z is a formal language with a defined syntax, like all programming languages, and some people use it to communicate but on a small range of topics, like natural languages. Z is a specification language based on conventional mathematic notations. The formal specifications of Z notation use mathematical notation to describe in a precise way the properties which an information system must have, without unduly constraining the way in which these properties are achieved (Spivey, 1989). The grammar is based on a mathematical language, but the semantics are of classical mathematics. The term classical is important in the description of Z. Classical mathematics has a foundation on two-value logic and set theory (Spivey, 1989). Z is built upon some of classical mathematics' central theories; however, it has a precise syntax, (Diller, 1994). Z is predominantly used for specifying software because the required result is an error-free computer program. Z notation is independent of program code, allowing the specification to be completed early so that there is an understanding of what the system does for the stakeholders (Spivey, 1989). This occurs due to Z making use of representational abstraction, procedural abstraction, and the structure of the schema (Diller, 1994; Wood & Silver, 1995).

Representational abstraction uses high level mathematical data types in a way without worrying how it will be implemented. Abstraction is the ability to describe what can be done without stating how it is done (Spivey, 1989). The true benefit is to solve a specific problem; one does not have to be constricted to the rigid data types. In short, it allows all possibilities to be considered without focusing on one aspect. The use of procedural abstraction ignores the how and focuses on the what. It is simpler to think of the specifications as declarative statements (Diller, 1994). This is useful since most problems are written as declaratives. The use of object-oriented analysis and design will produce procedures.

The use of abstraction directly links to the schema structure since it is based on declaratives. The schema structure is a means of organizing its notation around the definition of the problem entities being analyzed. Schemas are used to structure knowledge about a given entity within the problem. The pieces are linked by commentary so that the logic can be followed mathematically. The schema structure has two sections, declarative and criterion. The declarative section defines the variables and other schemas establishing the state of the entity. The criterion section defines the conditions establishing the relationships between the state variables. Schemas describe dynamic and static parts of the system. In the dynamic part, the possible operations, the relationships of inputs and outputs, and the changing states are explained. The occupied state and relationships create the static part of the schema (Spivey, 1989).

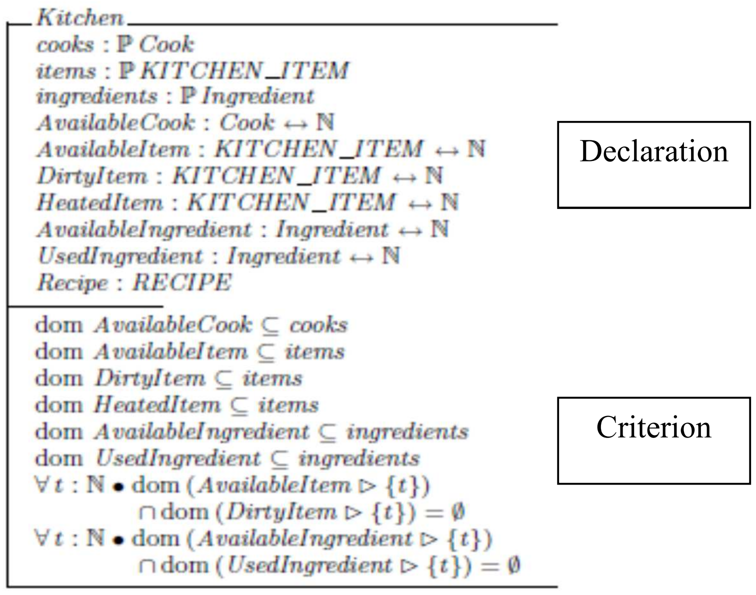
In an event-based modeling language, system operation models a discrete sequence of events in time. Each event occurs at an instant in time and defines a change in state of the system. Also, between consecutive events, it is assumed that there can be no change in the state of the system (Spivey, 1989).

Continuing with the example, in Fig. 13 Tarkan's (2009) schema for the kitchen describes the basis of the kitchen example. In the declarative section, Tarkan (2009) creates a space for any arrangement of cooks, items, and ingredients. Tarkan (2009) also sets limitations on *AvailableCook*, *AvailableItem*, *DirtyItem*, *HeatedItem*, *AvailableIngredient*, and *UsedIngredient* that the count is greater than or equal to zero or a natural number.

These declarations provide a logical baseline of what can occur in a kitchen. The permutations of *cooks*, *items*, and *ingredients* in a kitchen can vary. It is also true that a *HeatedItem* cannot be negative. Tarkan (2009) continues in the criterion section by explaining that *AvailableCook* is only a subset of the cooks defined in the declaration. The criterion:

$$\forall\, t : \mathbb{N} \bullet \text{dom}\,(\text{AvailableIngredient} \rhd \{\, t \,\}\,) \cap \text{dom}\,(\,\text{UsedIngredient} \rhd \{\, t \,\}) = \emptyset$$

states that an ingredient cannot be available and used at the same instant of time.



Kitchen

$cooks : \mathbb{P}\, Cook$
$items : \mathbb{P}\, KITCHEN\_ITEM$
$ingredients : \mathbb{P}\, Ingredient$
$AvailableCook : Cook \leftrightarrow \mathbb{N}$
$AvailableItem : KITCHEN\_ITEM \leftrightarrow \mathbb{N}$
$DirtyItem : KITCHEN\_ITEM \leftrightarrow \mathbb{N}$
$HeatedItem : KITCHEN\_ITEM \leftrightarrow \mathbb{N}$
$AvailableIngredient : Ingredient \leftrightarrow \mathbb{N}$
$UsedIngredient : Ingredient \leftrightarrow \mathbb{N}$
$Recipe : RECIPE$

$\text{dom}\ AvailableCook \subseteq cooks$
$\text{dom}\ AvailableItem \subseteq items$
$\text{dom}\ DirtyItem \subseteq items$
$\text{dom}\ HeatedItem \subseteq items$
$\text{dom}\ AvailableIngredient \subseteq ingredients$
$\text{dom}\ UsedIngredient \subseteq ingredients$
$\forall\, t : \mathbb{N} \bullet \text{dom}\,(AvailableItem \rhd \{t\}) \cap \text{dom}\,(DirtyItem \rhd \{t\}) = \emptyset$
$\forall\, t : \mathbb{N} \bullet \text{dom}\,(AvailableIngredient \rhd \{t\}) \cap \text{dom}\,(UsedIngredient \rhd \{t\}) = \emptyset$

Figure 13: Tarkan's Kitchen Schema (Tarkan, 2009, p. 3)

When converting from OOA to Z notation the classes are first examined. The classes are modelled in the form of schemas as mentioned and shown in Figure 13. The class will have a state schema, an initialized state, and operations that can change the variables. For each class a mathematical condition is applied as the type of variable, i.e., natural number, a specific defined number, a range, etc (Kassel & Smith, 2001; Winter & Duke, 2002). Allowing concepts of class, inheritance between classes and object referencing to be supported. Using the classes from the object-oriented analysis, the use of the attributes and behaviors are mapped to types or functions extended by the class name. Using referencing the object instances are accessed, meaning the operations are not defined operationally but in declarative form (Winter & Duke, 2002). This continues with the thought that the system states what is to be done not how to do it. The criterion section mentioned allows for the relationships to be defined pre- and post-states (Winter & Duke, 2002).

3.1.5 System Analysis Phase 3: Alloy Language and Analyzer

Phase 3 is the Alloy modeling and execution part of the system analysis methodology. Alloy is a modeling language for expressing complex structural constraints and behavior about systems (Jackson, Software Abstractions, 2012). Also, Alloy is a declarative specification language modeling tool employing first-order logic based on the Z notation. Structures in Alloy are described in space and time. A unique characteristic of Alloy is that it analyzes systems with configurations that are undetermined or for those that have the capacity to change dynamically (Jackson, Software Abstractions, 2012). Alloy's ability to conduct incremental analysis allows for the exploration of different designs starting from a small model, which is then scaled up. Alloy can analyze the model at every step. The purpose of converting Z notation to Alloy is to use to find and correct errors in the logic of the Z specification. The converted Z notation can model aspects of the system but not the entire system. Alloy Analyzer makes it possible to check the criteria of the specification to assure correct execution of the solution (Devlin, 1993).

To continue the example of Tarkan's work with the MSDLC, the Alloy code must be developed. The basic building blocks of the declaration section are made into "atoms." These items are the building blocks for all the data types in any schema. Based on Tarkan's work *cook*, *item*, *ingredients*, etc. will be atoms. The next level will be "primitives." Primitives are built upon atoms. Tarkan uses *ENAME* for event names. Events defined by Tarkan are *bakeDone*, *cleanDone*, *cookDone*, *cutDone*, *kneadDone*, *mixDone*, *preheatDone*, and *putDone*. Each of the events would be atoms. The collection of the events will be described as a primitive. The next step in coding will be to create the "initializer." Initializers are based on schema names, and the timing mechanism begins. The last portion needed to run the Alloy code is the "predicate." This file contains the active functions of the schema. The functions specify updates to the variables throughout the kitchen phases (Fletcher, Garcia, Nwachukwu, Nwaogu, & Reaves, 2014). Through the initiation, logic problems can be resolved and reiterated to improve the OOA models and Z notation.

Alloy attacks the notion of software or system abstraction in problem solution from a unique point of view. The assumption is that the current approach to problem solution does not work well. Therefore, Alloy addresses solving complex problems with three elements; logic, language, and analysis (Jackson, Software Abstractions, 2012). Logic provides the building blocks for the language. All logic structures within Alloy are represented as relations and operations. Problem states and executions are described using constraints (i.e., formulas or Boolean expressions) (Jackson, Micromodels of Software: Lightweight Modelling and Analysis with Alloy, 2002). Having a language adds syntax and structure to the logic descriptions. This approach supports classification and incremental refinement in the analysis. The analysis phase is not a solution through a theorem but the use of an instant process. This analysis approach is a form of constraint solving. A process of simulation is used to find instances of states or executions that satisfy a given property (Jackson, Software Abstractions, 2012). To check the model, a counterexample is found that violates a given property. The search for instances that satisfy the problem statement is done within a scope defined by the user. Within this scope or space, multiple instances can be run to analyze the problem (Jackson, Software Abstractions, 2012).

When converting from Z notation to Alloy language to execute there is a tradeoff of expressiveness and complexity in Z for the automation and graphical component of Alloy Analyzer. Alloy and Z are similar in the mathematical foundation of first order logic but allows the gaps between Z and OOA to be filled. Since Alloy is focused on relationships, the sets of Z notation should be converted to relationships (Dwivedi & Rath, 2012). The specified notation in Z based on set theory should be simplified to Alloy functions. This step is required since Alloy Analyzer only has ASCII notation (Jackson, Micromodels of Software: Lightweight Modelling and Analysis with Alloy, 2002).

3.2 Modified System Development Life Cycle

Big data, public health, and cyber security are all 21st century problems larger and more difficult to solve than in previous times (NAE, 2018). By implementing the changes to the SDLC a Modified System Development Life Cycle (MSDLC) methodology is created and shown in Fig. 14. The blue boxes are the modification added. The MSDLC will address the problems to combat the weakness of the SDLC, specifically scope creep, unclear requirements, lack of methodology, inability to see the solution until the system is complete, and lack of user involvement. This proposal addresses the standard SDLC steps with modifications to the general flow adding Joint Application Development (JAD) Sessions, constant user involvement, use case models, and an integration of analysis methods (i.e., Object-Oriented Analysis, Z notation, Alloy model and executor) using an integrated process.

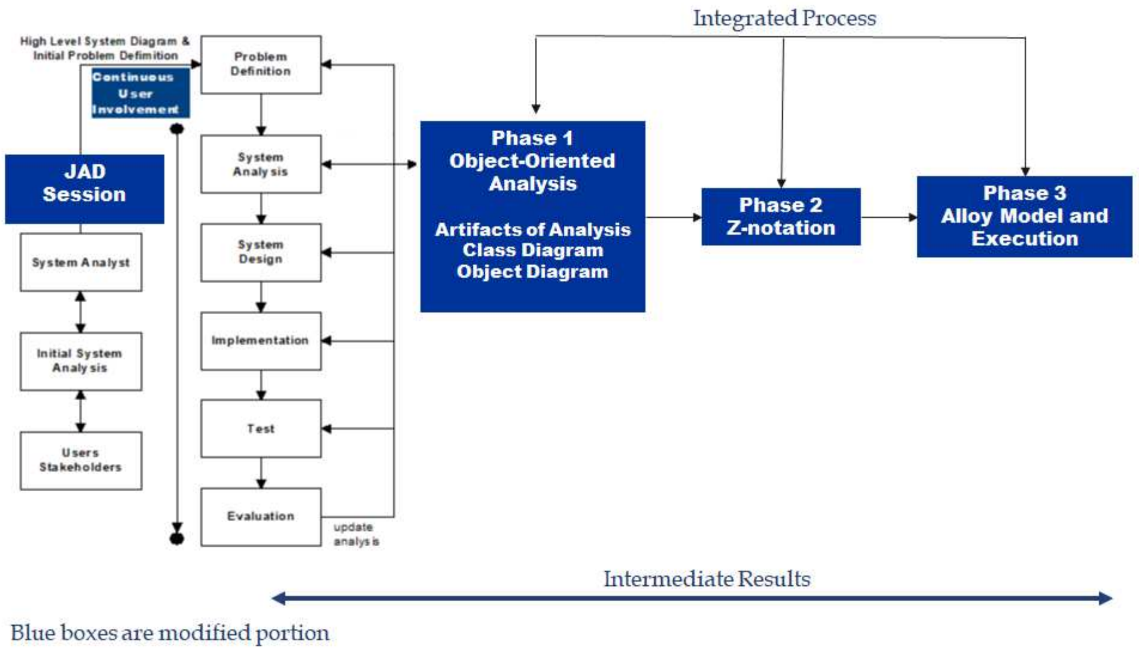

Integrated Process
High Level System Diagram & Initial Problem Definition
Continuous User Involvement
Problem Definition
System Analysis
System Design
Implementation
Test
Evaluation
update analysis
JAD Session
System Analyst
Initial System Analysis
Users Stakeholders
Phase 1 Object-Oriented Analysis
Artifacts of Analysis Class Diagram Object Diagram
Phase 2 Z-notation
Phase 3 Alloy Model and Execution
Intermediate Results
Blue boxes are modified portion

Figure 14: Modified System Development Life System (MSDLC)

Chapter 4: Research Results: Case Study

4.1 Background

The basis of the case study for comparison is the "Healthy People 2010." The Federal Government created the initiative to develop a national health promotion and disease prevention. In response to this statement, the National Heart, Lung, and Blood Institute (NHLBI) devised a new heart health agenda that identified four performance goals, each targeting a stage in the progress of cardiovascular disease (CVD). To achieve the four goals, NHLBI utilized community-based partnerships to bring CVD information to those in areas at high risk for CVD. This approach resulted in the creation of Community-Based Enhanced Dissemination Utilization Centers (EDUCs) which have been set up in high risk communities across the country (NIH, 2002).

The action centers are used to determine what communication strategies work best for an area or population. The Housing Authority of Baltimore City (HABC) joined the partnership as an EDUC in April 2001. The primary objective of the HABC EDUC is to improve the cardiovascular health (CVH) of the residents in public housing communities. The initiative specifically targets low-income African American communities in the City of Baltimore. HABC EDUC forged a partnership called Baltimore City Cardiovascular Health Partnership (BCCHP) which was responsible for conducting the project, appropriately titled "Healthy Hearts in Public Housing" (HHPH).

During the community-based CVH workshop in Baltimore City in September 2001, the following statistics on coronary heart disease (CHD) and strokes were presented. Respectively, based on 1996 to 1998 data, the age adjusted death rate for CHD and rate of stroke for African Americans in Baltimore city were 43 percent and 37 percent higher than the national rate. This results into more than 3,600 Baltimore City African Americans who died of CHD and stroke during this period (NIH, 2002).

Case Study

The hypothesis of the research is that by adding the modifications to the SDLC a validated system model and analysis process is produced leading to fewer errors in solving complex and dynamic problems in comparison to the Iterative method.

Based on data collected from the 2000 Heart Healthy Initiative the MSDLC will be implemented through the analysis phase. To conduct the study, a qualitative analysis will be completed to show the benefits in using the MSDLC approach on modeling the problem, capturing the dynamics of the problem, and minimizing the amount of errors.

4.2 Intermediate Artifacts of a JAD Session and Constant User Involvement

To begin the MSDLC process, time would be used to determine the scope and areas to evaluate. Using a JAD Session to map the entirety of the Heart Healthy Board and goals. Figure 15 is an intermediate artifact to state the members of the board and the goals. This diagram is one of many high level system diagrams allowing the group a visual of the agreed upon goals. From this first artifact a scope is given to be used as an example for the MSDLC method. Figure 15 also shows the scope highlighted in yellow as the Train the Community Health Workers (CHW). As the discussion in the JAD Session would continue another intermediate artifact would be produced with the activities and entities that assist in the activities.

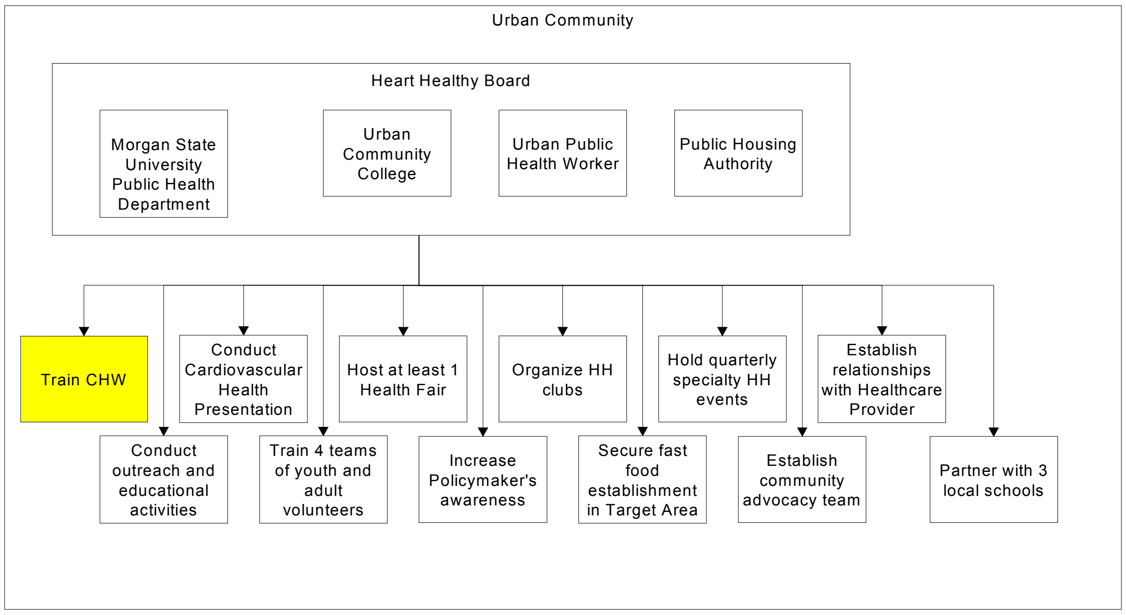

Urban Community
Heart Healthy Board
Morgan State University Public Health Department
Urban Community College
Urban Public Health Worker
Public Housing Authority
Train CHW
Conduct Cardiovascular Health Presentation
Host at least 1 Health Fair
Organize HH clubs
Hold quarterly specialty HH events
Establish relationships with Healthcare Provider
Conduct outreach and educational activities
Train 4 teams of youth and adult volunteers
Increase Policymaker's awareness
Secure fast food establishment in Target Area
Establish community advocacy team
Partner with 3 local schools

Figure 15: Heart Healthy Board Goals

Figure 16 is the first description of the requirements to train a community health worker. The figure shows some of the interactions between the Heart Healthy Board. Specifically, the Public Housing Authority will provide a training location where the Urban Community College will teach. There is evidence that a test is required. The Urban Community College will administer a test that the Public Housing Resident will take. Figure 16 is another high level system diagram that will be reiterated on to describe the activities and requirements needed to complete training of a CHW.

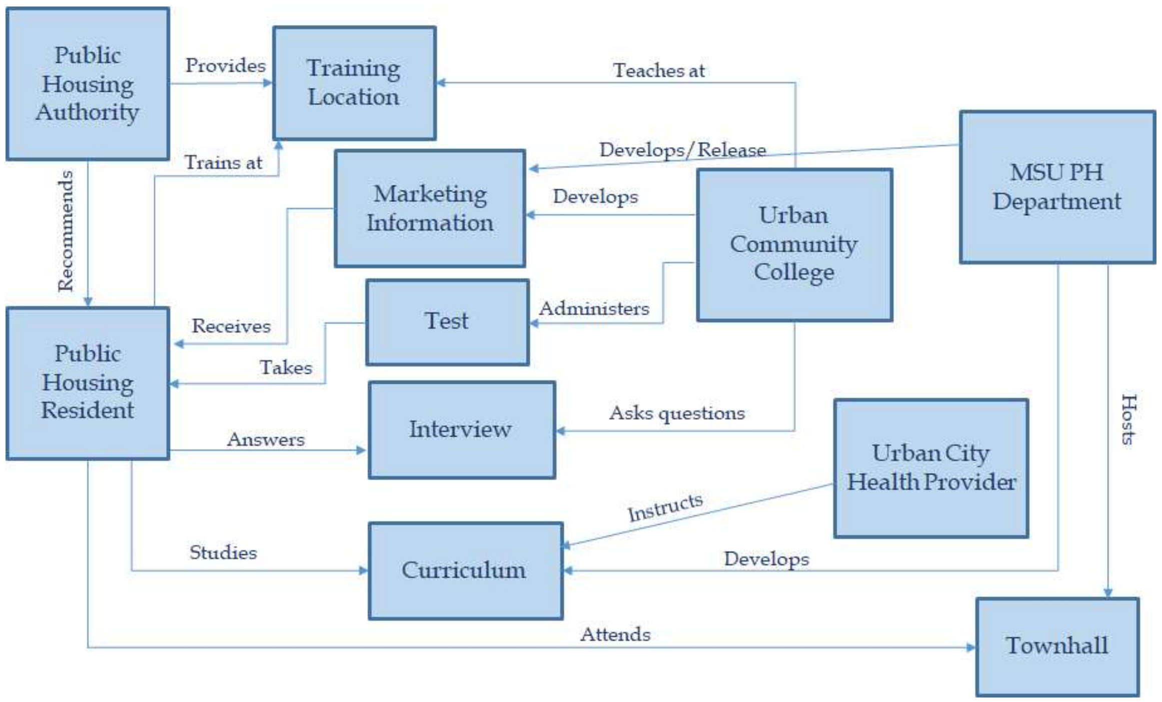


Figure 16: Train the Community Health Worker High Level System Diagram

4.3 Object Oriented Approach Train Community Health Workers

The JAD Session would end with a baseline of what is needed to train a CHW. The ending of this process would move to further refine the process by using Object Oriented Analysis (OOA). The OOA would begin with an overview of the major activities needed to train a CHW. Those major activities are developing a curriculum, recruiting for future CHW, interviewing Public Housing Residents, and training the future CHWs. Figure 17 is an intermediate solution that provides a clear understanding of key events that will occur to train the CHWs.

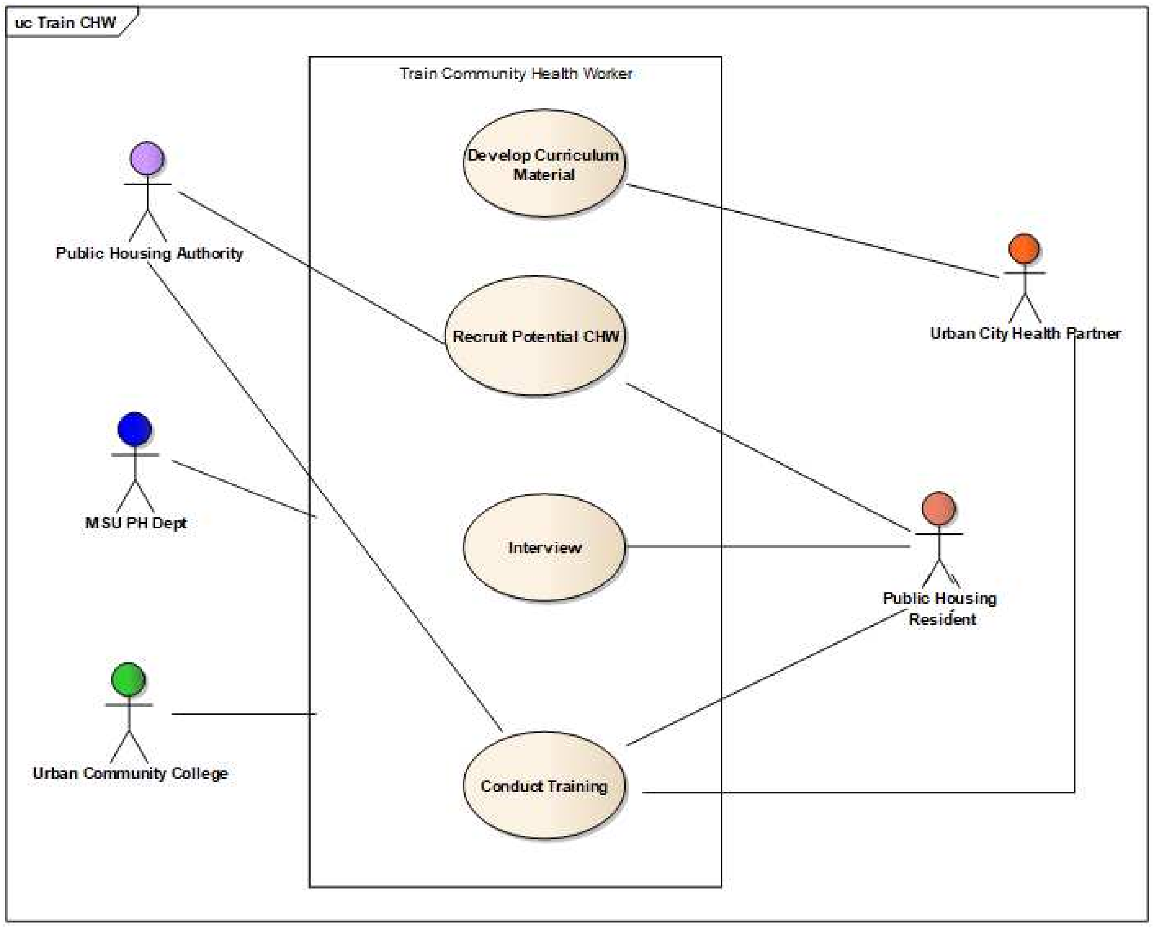


Figure 17: Training CHW Scenario Overview

The overview shows that to train a CHW, the main actors are the public housing authority, urban city health partner, public housing resident, the Morgan State University Public Heath (MSU PH) department, and the urban community college. The use case also shows the main events as previously mentioned. The overview diagram uses people figures to demonstrate the actors with lines connecting them to the events. MSU PH Department and Urban Community College have lines connected to the use case box indicating that both entities are in all scenarios.

Using Fig. 17 as a roadmap, each of the events can be broken into a separate use case. Figures 18 to 21 show details to the events in Fig. 17.

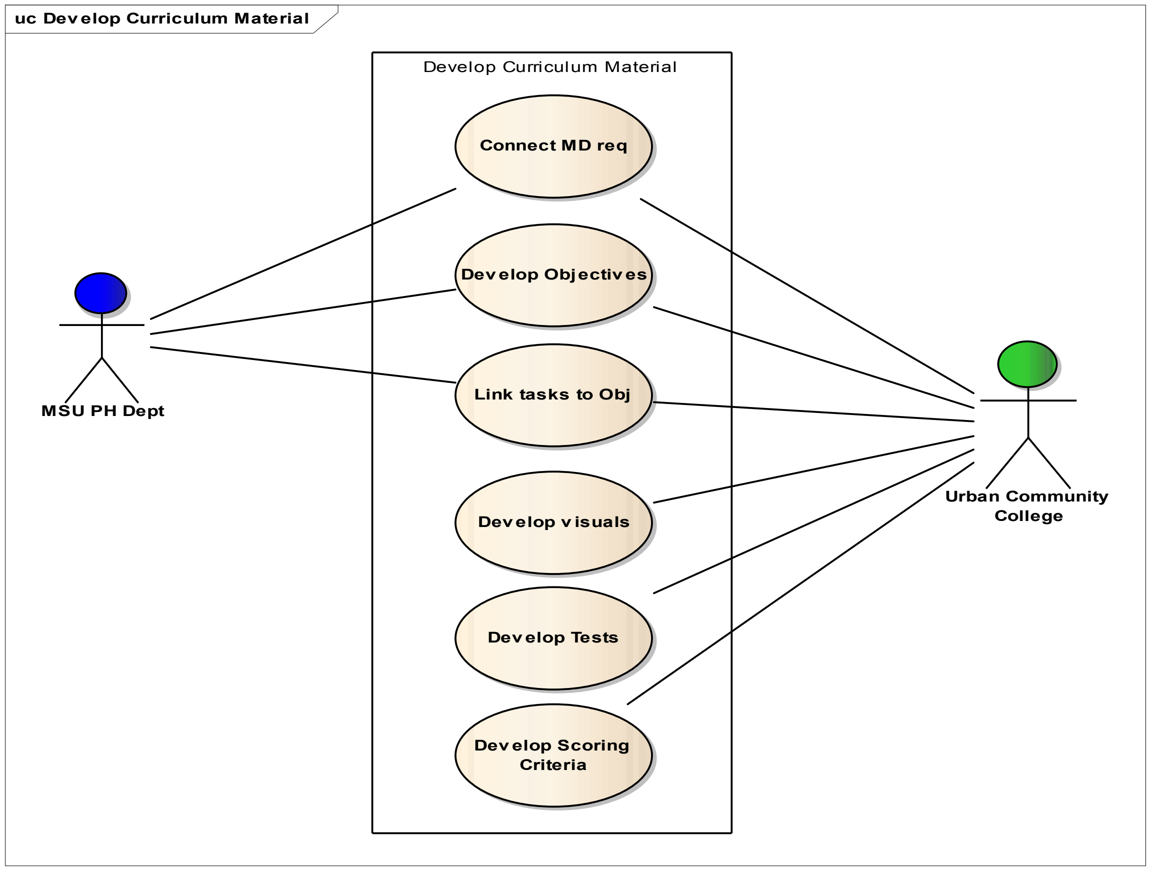


Figure 18: Develop Curriculum Material Use Case

Expanding on the details in Fig. 17, Fig. 18 has the events needed to develop the curriculum material. The actors are MSUPH department and the urban community college. The main activities are: connect Maryland requirements, develop objectives, link CHW tasks to objectives, develop visuals for curriculum, develop tests for curriculum, and develop scoring criteria for test and practicum. The continuity of Fig. 17 and Fig. 18 can be seen with the respective actors. Figure 18 does provide more detail in requirements and actions needed to complete developing the curriculum.

Figure 19 is the Recruit Potential CHW use case. The use case continues to provide more details to successfully complete the event. The actors are the public housing authority, MSUPH department, public housing resident and urban community college. The events are: prescreen community, recommendation, host townhall, hand out applications, and accept applications. These events are all needed to recruit potential CHW.

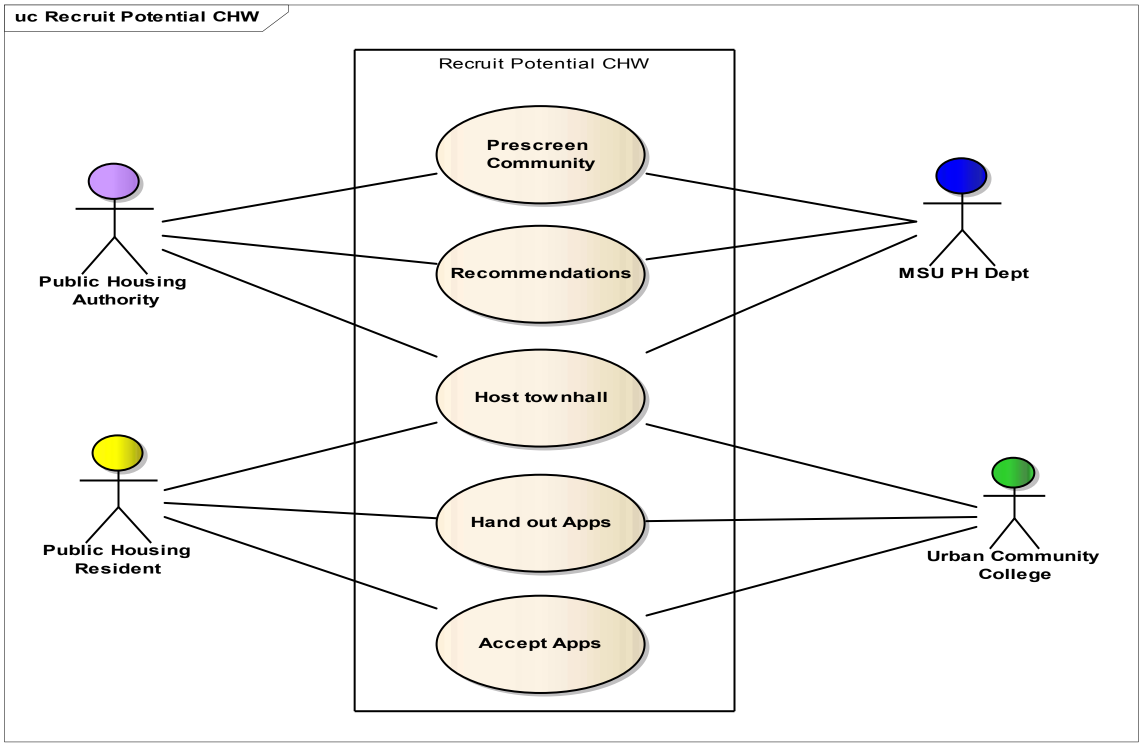


Figure 19: Recruit Potential Community Health Worker Use Case

Figure 20 is the Interview Public Housing Residents use case. The key actors are MHUPH department, urban community college, and public housing resident. The events needed to interview are: prepare question list, ask questions, answer questions, score answered questions, and contact participants.

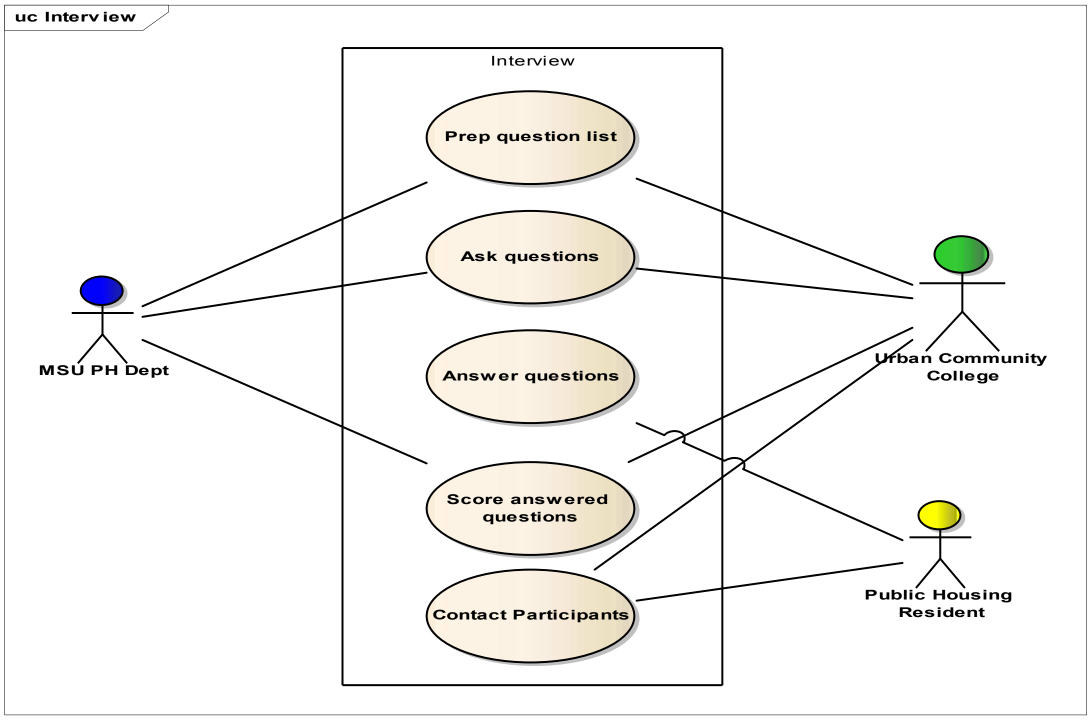


Figure 20: Interview Public Housing Residents Use Case

Figure 21 is the Conduct CHW Training use case. The key actors are MSUPH department, public housing authority, urban community college, public housing resident, and urban city health partner. The events are present lesson, practicum, study, test, and evaluate.

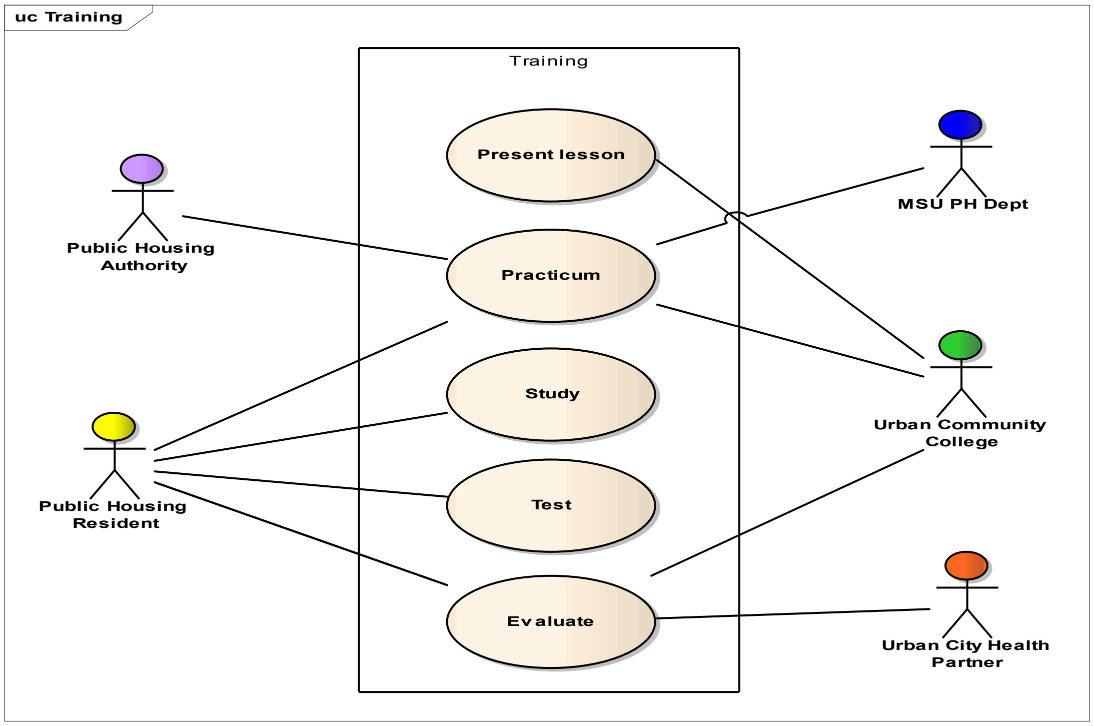
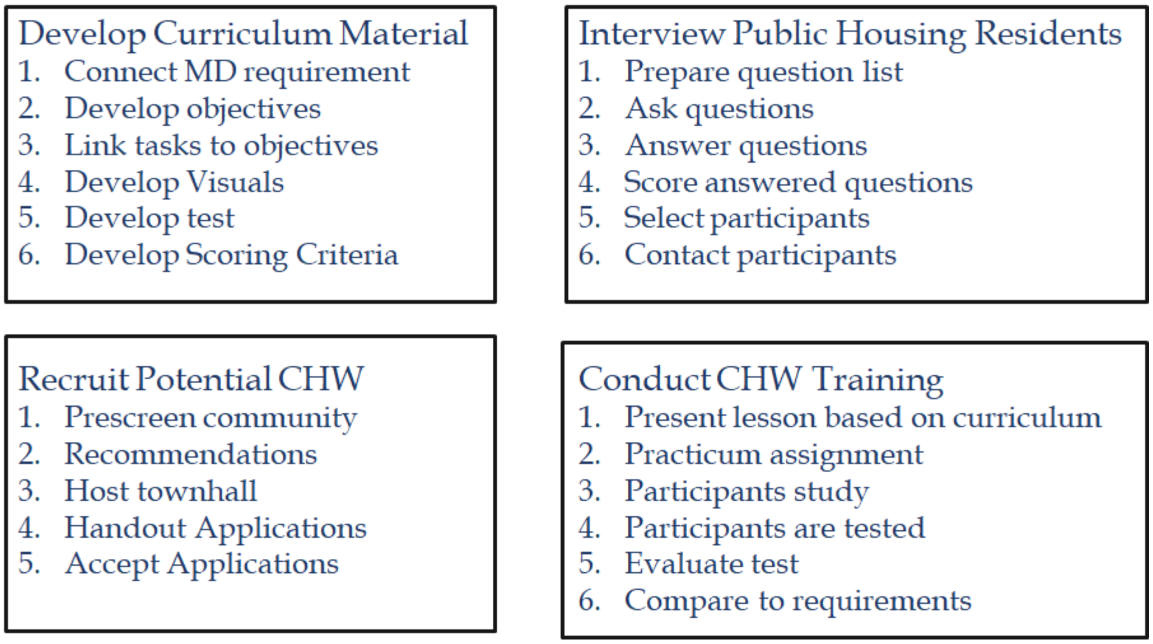


Figure 21: Conduct CHW Training Use Case

The combination of the use cases shown in Figs. 18 to 21 will encapsulate the training of a CHW. The aggregate of the use case events provides a detail understanding of what actions are needed to train a CHW. Each of these intermediate artifacts can be adjusted and revisited in preparation for training the CHWs. The use cases are a first instance at a deeper level to a successfully plan to train the CHWs.

To expand on the scenarios in the use cases, Fig. 22 has a written explanation of each. The requirements needed to satisfy each scenario are described in a step process.



Develop Curriculum Material
1. Connect MD requirement
2. Develop objectives
3. Link tasks to objectives
4. Develop Visuals
5. Develop test
6. Develop Scoring Criteria

Interview Public Housing Residents
1. Prepare question list
2. Ask questions
3. Answer questions
4. Score answered questions
5. Select participants
6. Contact participants

Recruit Potential CHW
1. Prescreen community
2. Recommendations
3. Host townhall
4. Handout Applications
5. Accept Applications

Conduct CHW Training
1. Present lesson based on curriculum
2. Practicum assignment
3. Participants study
4. Participants are tested
5. Evaluate test
6. Compare to requirements

Figure 22: Train the Community Health Worker Scenarios

Continuing with the OOA process, class diagrams would be created to provide data needed for collection and to describe the actions being conducted. Figures 23 to 26 will expand upon Figs. 18 to 21 with additional details from the scenarios described in Fig. 22. In Fig. 23 the requirements from Fig. 18 are shown in more detail. Figure 18 showed the key entities that were involved in the requirement. The class diagram in Fig. 23 shows the exact interaction that both MSU PH Department and Urban Community College are connecting requirements that will inform the objectives for training. The objectives are developed by again MSU PH Department and Urban Community College. Urban Community College will visualize tasks that are linked to objectives. Lastly Urban Community College will develop the test.

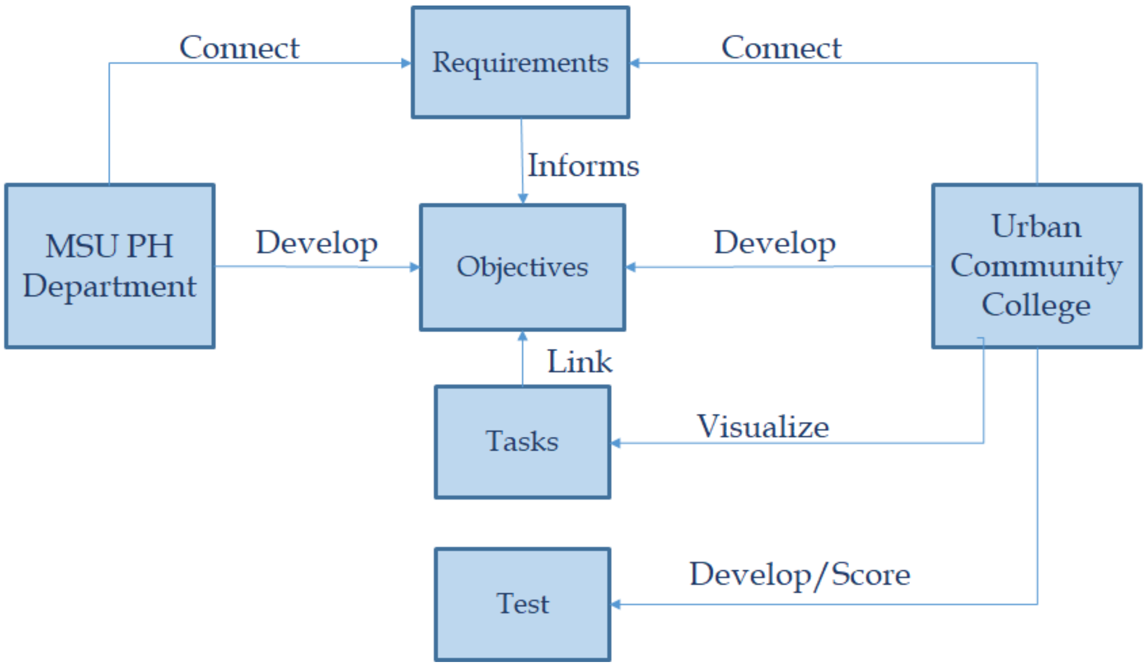


Figure 23: Develop Curriculum Material Class Diagram

In Fig. 24 the Recruit Potential CHW class diagram is shown. The class diagram shows the Public Housing Authority prescreening the community to make recommendations with MSU PH Department for Public Housing Residents. Public Housing Authority, MSU PH Department, and Urban Community College will host the townhall. The Public Housing Resident will attend the townhall and a fill out and return the application. The Urban Community College will handout and accept the applications. Figure 25 is the Interview Public Housing Residents class diagram showing MSU PH Department and Urban Community College will prepare questions and select the Public Housing Residents to be potential CHWs. The Urban Community College will ask questions during the interview that the Public Housing Resident will answer.

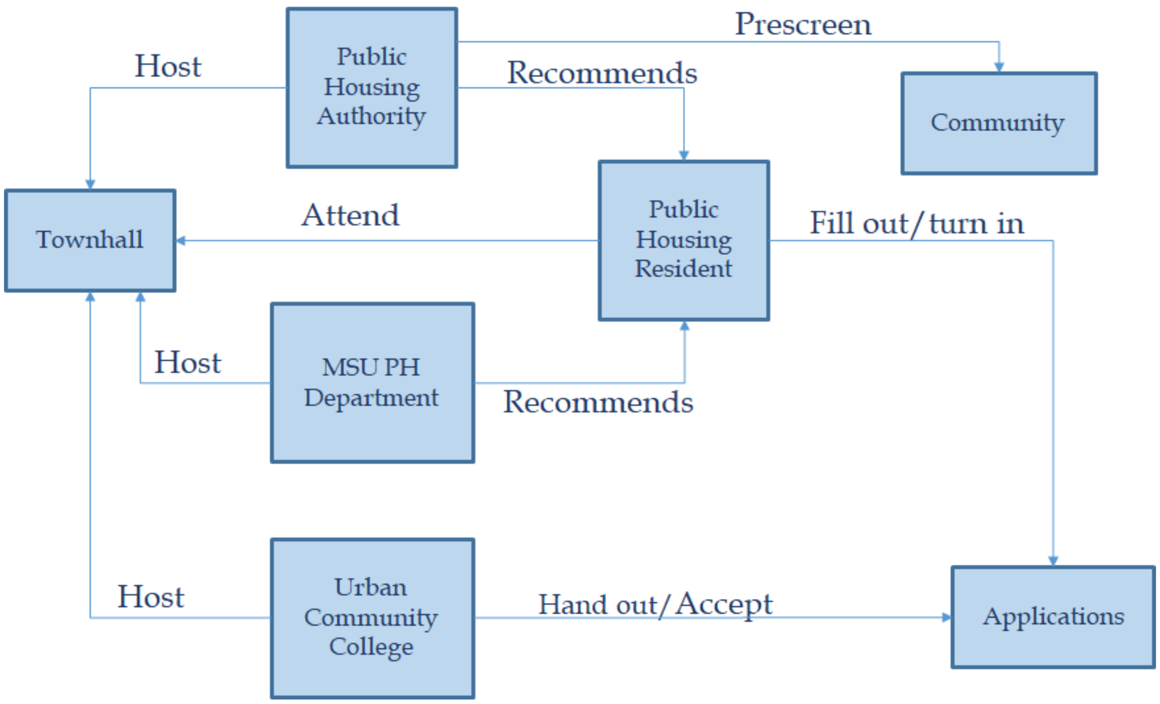


Figure 24: Recruit Potential Community Health Worker Class Diagram

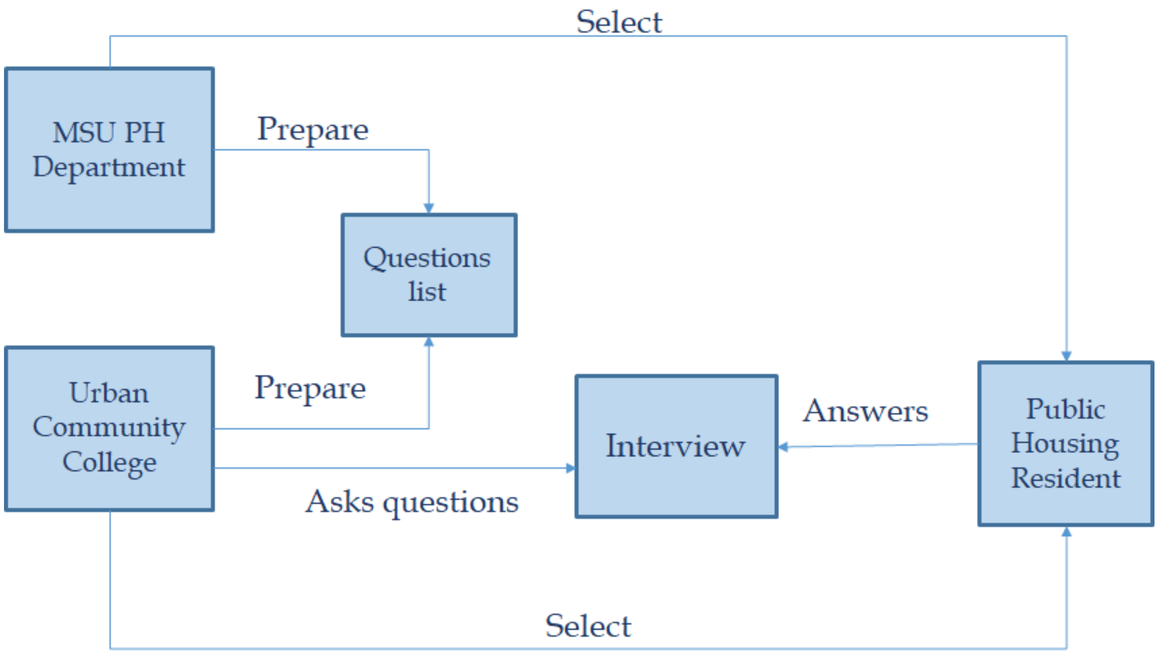


Figure 25: Interview Public Housing Resident Class Diagram

Figure 26 is the Conduct CHW Training class diagram showing the Public Housing Authority recommending the Public Housing Resident and providing a training location. Urban Community College will teach at the training location while the Public Housing Resident will train at the training location. MSU PH Department will develop the curriculum with the Urban Community College instructs the curriculum. The Public Housing Resident will study the curriculum and participate in a practicum.

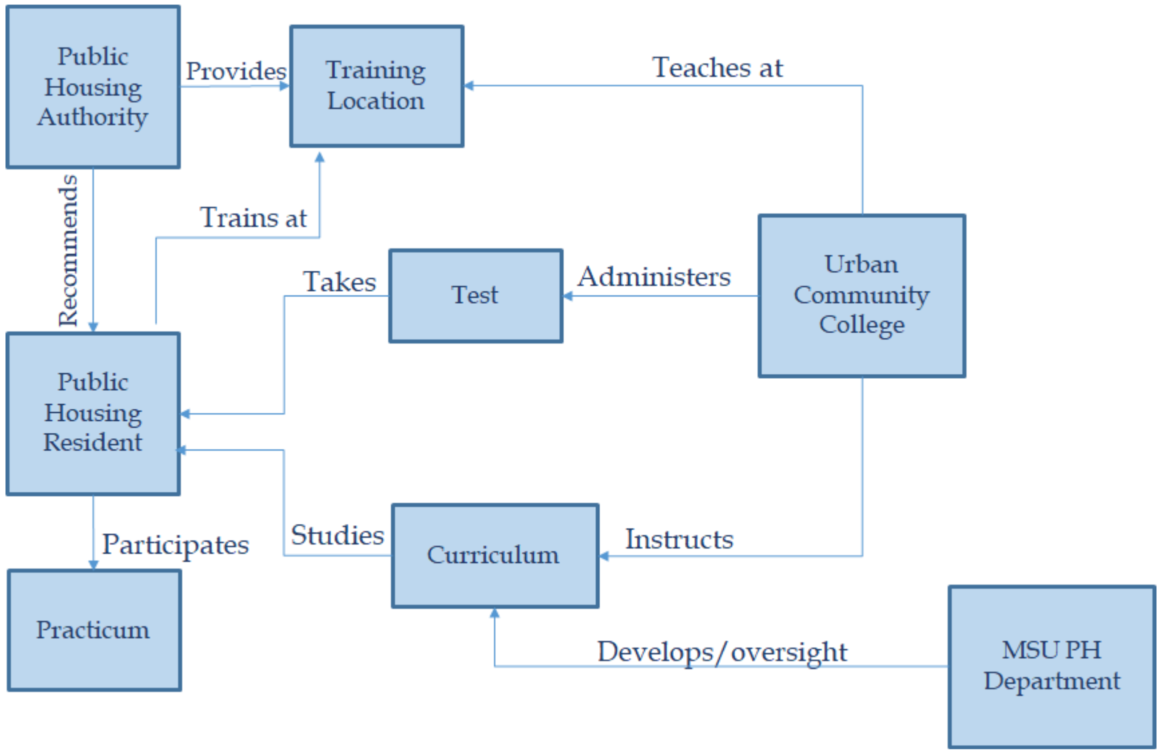


Figure 26: Conduct Training Class Diagram

4.4 Z notation and Alloy Model of Train the CHW

Based on the information that was used in the object-oriented diagrams leads to the use of Z notation and Alloy modeling. The use of predicate logic, set theory, and model execution will be used in this section to determine the validation of the model. For each use case scenario examples will be used to showcase the ability of the information to flow from OOA to Z to Alloy, that assumptions are made and can be evaluated, and that the architecture is model driven.

4.4.1 Develop a Curriculum

The Develop a Curriculum scenario will be defined as one of the types of scenarios using the Z notation

Scenarios::= Develop | Recruit | Interview | ConductTraining

Scenarios is broken into the four requirements to training a CHW: develop a curriculum, recruit potential CHW, interview Public Housing Residents, and conduct CHW training. To determine the requirements for any training criteria the union of requirements from Urban Community College and MSU PH Department will be a subset of the Maryland training requirements. The objectives for the training are also a union. The tasks are a function of the objectives defined.

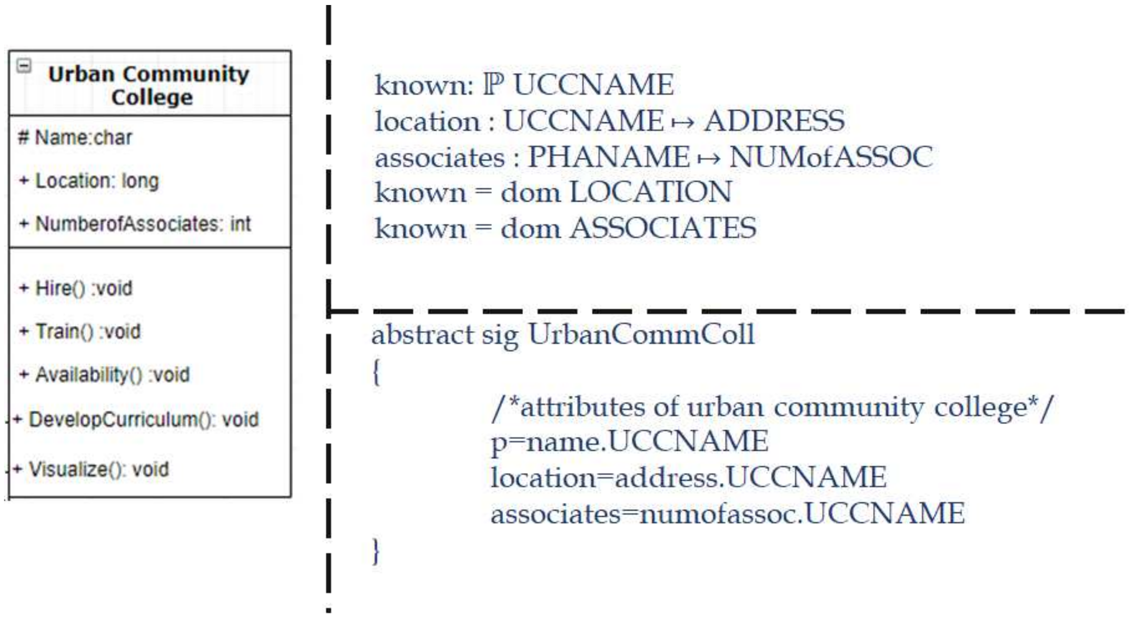
$Requirements = dom(RequirementsUcc) \cup dom(RequirementsMPD) \subset standard? \in dom(RequirementsMD)$

$OBJ = dom(ObjectiveUcc) \cup dom(ObjectivesMPD)$

$Tasks : OBJ \mapsto TASKS$

In the Develop a Curriculum use case, as shown in Fig.27, has all three of the analysis phases with OOA on the left, Z notation on the top right, and Alloy on the bottom right. In the OOA section the attributes and behavior are shown for the Urban Community College. These section shows that the information can be transformed from object oriented to Z notation and to Alloy. Z and Alloy allows for the expansion of the information. In the object-oriented analysis, the information is defined. Moving the information into Z and Alloy, the information is kept and allowed to be passed for future changes in the scenarios.

The ability to transform the data is part of model driven architecture. The future transformations will allow for the Alloy analyzer to interpret the potential solution space. This specific structure also allows for the assumptions and relationships for Urban Community College to be transferred. Similar information for a resident will be used in in later scenarios and the determine the basis for interviewees and trainees.



```
known: ℙ UCCNAME
location : UCCNAME ↦ ADDRESS
associates : PHANAME ↦ NUMofASSOC
known = dom LOCATION
known = dom ASSOCIATES
```

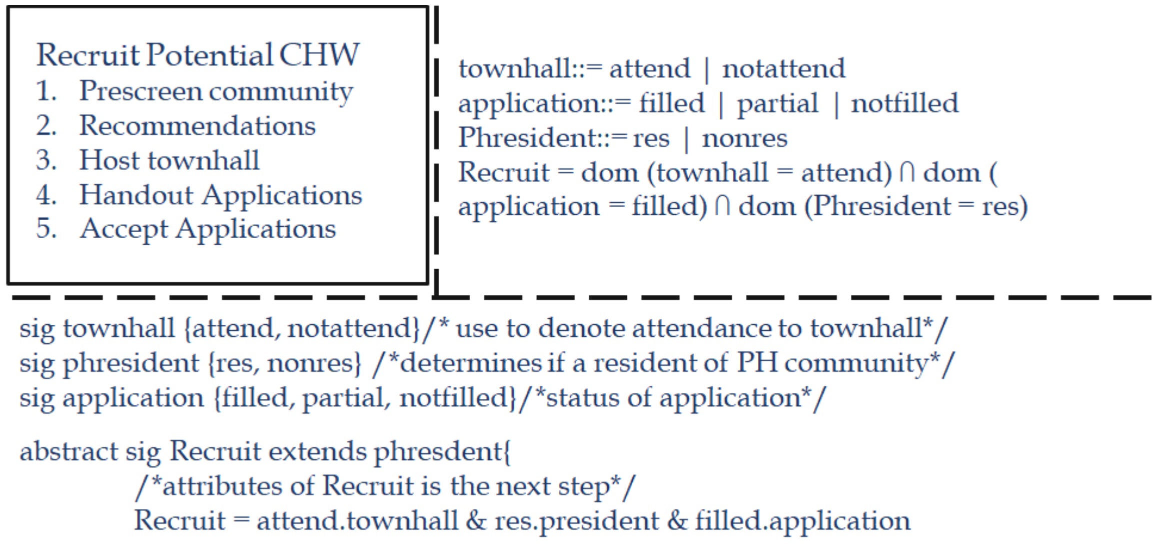
```
abstract sig UrbanCommColl
{
        /*attributes of urban community college*/
        p=name.UCCNAME
        location=address.UCCNAME
        associates=numofassoc.UCCNAME
}
```

Figure 27: Develop Curriculum Material Structure in Z and Alloy

4.4.2 Recruit Potential Community Health Workers

The recruit potential community health workers, shown in Fig. 28, has the expanded scenario. OOA is on the top left, Z notation on the top right, and Alloy on the bottom. From the OOA section, also reflected in Fig. 22, in step 3 a townhall will be held. The Z notation on the top right provides the rules and assumptions that a person has either attended or not attended the townhall. The Z notation also sets the states that the application can be: filled, partially, or not filled. The residency requirement is created in the *phresident* status. The Alloy model in the lower half continues with that information that the recruit is an extension of a Public Housing Resident (*phresident*). The extension also provides the rule that to be a possible recruit one must be a resident of Public Housing.

The rules and assumptions in the recruit example show another instance of model driven architecture. The assumptions and rules are assisting in developing and verifying the solution space.



Recruit Potential CHW
1. Prescreen community
2. Recommendations
3. Host townhall
4. Handout Applications
5. Accept Applications

townhall::= attend | notattend
application::= filled | partial | notfilled
Phresident::= res | nonres
Recruit = dom (townhall = attend) ∩ dom (application = filled) ∩ dom (Phresident = res)

```
sig townhall {attend, notattend}/* use to denote attendance to townhall*/
sig phresident {res, nonres} /*determines if a resident of PH community*/
sig application {filled, partial, notfilled}/*status of application*/

abstract sig Recruit extends phresdent{
        /*attributes of Recruit is the next step*/
        Recruit = attend.townhall & res.president & filled.application
```

Figure 28: Recruit Potential Community Health Worker Analysis

To define the requirements for a potential recruit, the logic would define the space as the intersection of Public Housing Resident, townhall attendees, and filled applications.

$$Recruit = dom\ (townhall = attend) \cap dom\ (application = filled) \cap dom\ (Phresident = res)$$

The scenario has that MSU PH Department, Public Housing Authority, and Urban Community College are to host the event. Since all three entities are also members of the Heart Healthy Board, the hosts are a subset of the board.

$$HHM = PHA \mid MPD \mid UCC \mid UCHP$$
$$Host \subset HHM$$

The Public Housing Authority has the tasks of prescreening the community and recommending Public Housing Residents. Since the potential CHW must be residents an assumption can be made that the residents are a subset of the community.

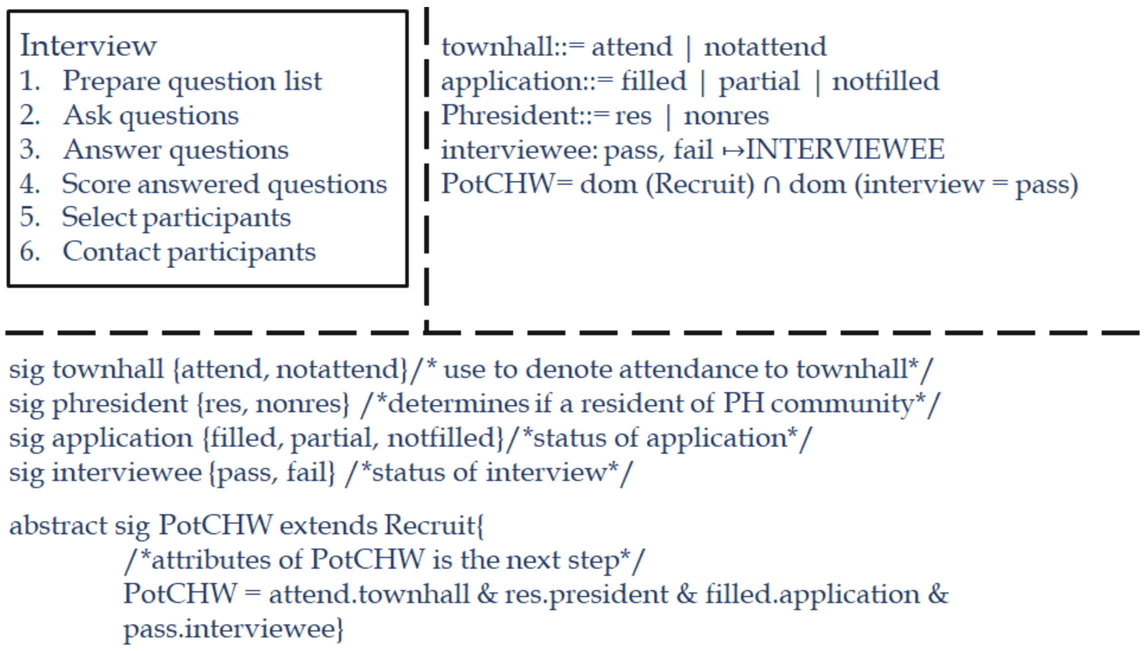
$Residents \subset Community$

Lastly, the behaviors taken by the key entities can be captured with a notation before the entity. Below the notation for recommendation precedes both Public Housing Authority (PHA) and MSU PH Department (MPD).

prescreen.PHA
recommend.PHA
recommend.MPD

4.4.3 Interview Public Housing Residents

The Interview Public Housing Residents use case scenario is shown in Fig. 32 below. OOA is on the top left, Z notation on the top right, and Alloy on the bottom. The example highlights the validation of assumptions. For a person to be an interviewee they must have attended the townhall and be a public housing resident. This is an assumption in the object-oriented analysis but can be confirmed through logically stating in Z then transformed to Alloy for execution. The execution of the Alloy model will allow for each person to be verified that they are a resident and have attended the townhall. The information passed in the *phresident* and *townhall* is used. The Z notation also validates that the *POTCHW* (potential CHW) is the intersection of those who are residents and have passed the *INTERVIEWEE* phase.



Interview
1. Prepare question list
2. Ask questions
3. Answer questions
4. Score answered questions
5. Select participants
6. Contact participants

```
townhall::= attend | notattend
application::= filled | partial | notfilled
Phresident::= res | nonres
interviewee: pass, fail ↦INTERVIEWEE
PotCHW= dom (Recruit) ∩ dom (interview = pass)
```

```
sig townhall {attend, notattend}/* use to denote attendance to townhall*/
sig phresident {res, nonres} /*determines if a resident of PH community*/
sig application {filled, partial, notfilled}/*status of application*/
sig interviewee {pass, fail} /*status of interview*/

abstract sig PotCHW extends Recruit{
        /*attributes of PotCHW is the next step*/
        PotCHW = attend.townhall & res.president & filled.application &
        pass.interviewee}
```

Figure 29: Interview Public Housing Residents Analysis

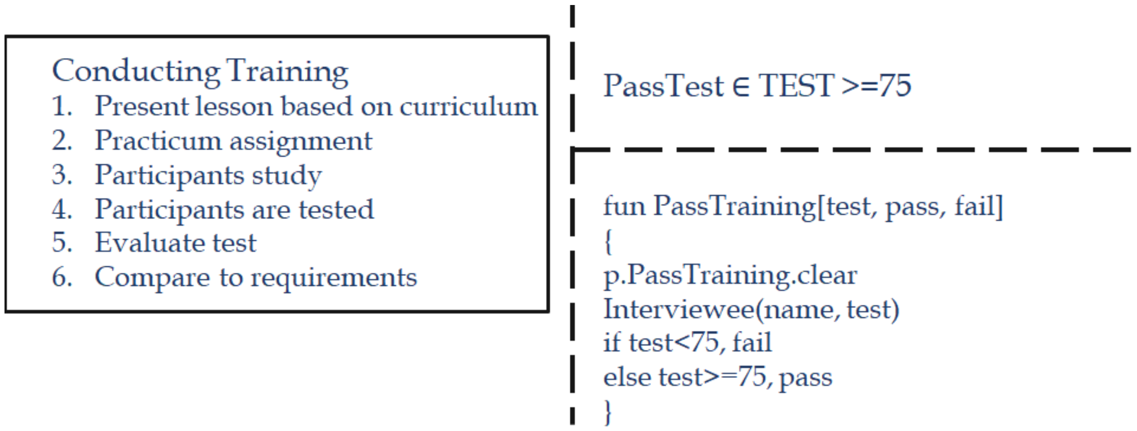
The Interview Public Housing Residents scenario states that MSU PH Department and Urban Community College will prepare questions for the interview. The questions asked during the interview is a subset of the combination of the questions that have been prepare by MSU PH Department and Urban Community College. Also, the questions asked during the interview are the same for each recruit.

$$QUESTIONS \subset dom(QuestionsUcc) \cup dom(QuestionsMPD)$$

$$QUESTIONS \mapsto \mathbb{N}\ Recruit$$

4.4.4 Conduct Community Health Worker Training

The conduct CHW training example is shown in Fig. 34 with OOA on the left, Z notation on the top right, and Alloy on the bottom right. The validation of a test score is explored. A test score of 75 has been selected for passing the test. Z and Alloy allow for the checking of the test scores for the interviewees. All the interviewees will have the name and test score passed along. Not every interviewee will have a test score, however the score will be compared to the passing score. Passing the test is a requirement to become a community health worker. The schema provides a percentage of a student attending class. The following line: $PassTest \in TEST >= 75$, will confirm if a test score meets the minimum requirement for passing the test.



Conducting Training
1. Present lesson based on curriculum
2. Practicum assignment
3. Participants study
4. Participants are tested
5. Evaluate test
6. Compare to requirements

PassTest ∈ TEST >=75

```
fun PassTraining[test, pass, fail]
{
p.PassTraining.clear
Interviewee(name, test)
if test<75, fail
else test>=75, pass
}
```

Figure 34: Conduct Community Health Worker Training Analysis

The Conduct CHW Training schema allows for a check in scheduling and room availability. The logic statement to determine the class time is the intersection of the location and trainer is below will confirm if at a specific time if the location for the class is available

$ClassTime = dom(AvailableLocation \rhd \{currTime\}) \cap dom(UCCTrainer \rhd \{ currTime \})$

The Public Housing Authority owns many buildings. The training location will be one of the building locations belonging to the Public Housing Authority. The logic statement for the training location is below.

$phabuildings : \mathbb{P}\ PHAbuildings$

$TrainingLocation \subset PHAbuildings$

4.5 Benefits of MSDLC Solution and Deliverables

The findings from the SDLC research resulting in the MSDLC provide multiple insights. The MSDLC process including the JAD Session, continuous user involvement, a three-tier analysis phase, and intermediate artifacts resulting in a validated solution before entering the system design phase.

The MSDLC provides a formal definition for complex systems. By modifying the SDLC, specifically the use of a JAD Session and OOA allows for a clear definition of the problem. The purposeful meeting of key stakeholders, users, and project managers presents a time to discuss the problem and get a clear understanding of potential solutions. Taking the information from the JAD Session then is further refined using OOA. OOA provides a foundation of the requirements as a result of the potential solutions from the JAD Session. The requirements focus on what to do rather than how to do a task provides opportunity to iterate and clarify what the requirements are and remove uncertainty. Developing use case scenarios from the requirements refines the transactions in a measurable manner including the behaviors seen from the user. The result is a formal definition for complex systems.

The MSDLC provides an improved scientific structuring of problem. Z notation is used to provide an improve scientific structure of the problem. With a formal definition derived from the JAD Session and OOA, Z notation provides a space to logically process what the requirements are and any restrictions. The use of mathematical notation allows the properties to be explained without being prescriptive to how the property is achieved. This concept is abstraction. The benefit of abstraction is the ability to consider all the possibilities without bias toward one aspect. This continues to build upon the focusing on what to do rather than how to do a task.

MSDLC improved the specification and validation of criteria and assumptions. After exiting the OOA phase, a foundational definition is provided. Each entity has attributes and behaviors that link directly to the scenarios. Z notation provides a space to improve and logically document the assumptions and criteria for the requirements. Alloy Analyzer takes the logical space and determines if the conditions apply to validate the criteria and assumptions.

Intermediate results are made available for use in system analysis through the MSDLC. High level system diagrams, class diagrams, and object diagrams assist in communicating to the user. These visuals are a simplified way to see the requirements, the process, and determine if changes are needed. The intermediate results also provide a historical documentation of changes to the problem. The diagrams are a timeline of changes and enhancements made to the problem. This can be substituted for lengthy written documents.

Lastly, MSDLC provides greater confidence in solution through Alloy Analyzer. The Alloy Analyzer automatically analyzes requirements and design specifications for potential errors and inconsistences. The analyzer is a validation tool checking the logic provided through all possible constraints. Using this tool provides the instances that do not conform to the restrictions provided. Knowing the instances that do not meet the criteria provided gives added value to know what will not work in the application. Having this information in the analysis phase will save costs versus realizing in a later phase.

The MSDLC methodology has defined an interface of OOA, Z notation, and Alloy Analyzer. The interface previously described allows for a transformation of information between the three analysis phases. The structured defined problem and organized foundation of requirements created in OOA were translated into the criteria and assumptions in Z notation. The conversion of Z notation to the Alloy Analyzer will provide a confident and validated solution space because all possible instances are evaluated. Throughout the conversions the foundational information is captured and moved in all three phases. Having a method to keep the critical information means each phase has the same information. As each phase has the same information the potential solutions are also within the same space. The assumptions are reaffirmed because the behaviors are the same from OOA. The solution space from Alloy Analyzer is then also purely based on the assumptions and behaviors. This process facilitates other researchers in the field to have a structured approach for using the MSDLC methodology.

4.6 Future Work

Continued analysis can be done on the Training of a Community Health Worker by maximizing the persons that will complete and pass the training. Generally, the limitations of training are the failure rate and drop-out rate. Additional training completion factors can be formed into questions for the questionnaire or the interview. That information can be integrated into the logic of Z notation and executed in Alloy Analyzer.

By the middle of the 20th century infectious diseases, even with a high mortality rate, were largely conquered by innovations in medications and vaccinations. However, in 1988 the Institute of Medicine declared that public health was in a state of disarray (Institute of

Medicine (IOM), 1988). Since 1988 the profession has done much to improve its structure as it moved from a focus on addressing problems associated with infectious diseases to the current epidemic of non-communicable diseases. Since that time, public health has been confronted with the current epidemic of non-communicable diseases such as diabetes, hypertension, cardiovascular disease, and cancer where changed individual choices and behaviors are necessary to improve health outcomes. With the recognition that obesity is related to the early onset and severity of these diseases, public health has been forced to adjust its approach to encompass the social determinants of health (SDOH). The SDOH is defined as the conditions where people are born, grow, work, age with the pervasive influence of money, power and politics (Solar & Irwin, 2007). This expanded public health approach has brought with it changing levels of complexity and big data.

Since public health science relies heavily on epidemiology and biostatistics for analytical tasks, the current levels of complexity and big data have presented many challenges that the MSDLC methodology can inform. In recent years, system science and network modeling have been used to move public health forward in managing this complex environment with massive amounts data. The MSDLC will allow even more advancement because it will make it possible for system models to not only be archived for future use but to standardize conceptual thinking using object-oriented models translated into Z notation and the findings validated through the Alloy Analyzer. Innovations to assist in methodologically standardizing and validating information will help public health become more consistent in moving from conceptual modeling to testing and refining outcomes with the hope of being able to bend the curve on these very costly non-communicable diseases.

In the future this approach can be used in various other applications such as financial and system acquisition. The Department of Defense Planning Programming Budgeting and Execution (PPBE) process used to fund the departments initiatives. The process begins almost two years before execution. Tracking the requirements that set a basis for fund that is to be executed is a complex task. The MSDLC can be used to logically flow funding from the requirements to actualization in overseeing the process from start to finish (Shevin-Coetzee, 2016). The MSDLC will assist in providing a logical connection between the requirements of planning to the execution of the funding. The MSDLC will assist in confirming the amount planned and programmed can be linked to an executed amount used to meet the mission.

References

Adrian, A. (2013). Big Data Challenges. *Database Systems Journal, IV*(3), 31–40. DOI: 10.4172/2324-9307.1000133.

Alexander, I. F., & Maiden, N. (2004). *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*. West Sussex: John Wiley & Sons Ltd.

Ammu, N., & Irfanuddin, M. (2013). Big Data Challenges. *International Journal of Advanced Trends in Computer Science and Engineering*, 2(1), 613–615. DOI: 10.13140/RG.2.2.16548.88961.

Appleget, J. (2010, July 14). NPS Initiatives in Human Social Culture Behavior (HSCB) Modeling FY 2010–2011. California, USA.

ARS. (2010, May 24). *S.D.L.C. Software Development Life Cycle & its different models*. Retrieved from ARSORACLE: http://arsoracle.blogspot.com/2010/05/sdlc-software-development-life-cycle.html.

Ateeq, S., & Shuaib, M. (2014). COMPARISON OF VARIOUS SDLC MODELS. *Global Journal of Multidisciplinary Studies*, 3(11), 176–181.

Avison, D. E., & Fitzgerald, G. (2003). Where Now for Development Methodologies? *Communications of the ACM*, 46(1), 79–82.

Balaji, S., & Murugaiyan, M. S. (2012). WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC. *International Journal of Information Technology and Business Management*, 2(1), 26–30.

Baltzan, P., & Phillips, A. (2010). *Appendix D. The Systems Development Life Cycle Basics*. In M: Information Systems (pp. D1–D18). McGraw-Hill.

Bender RBT Inc. (2003). *System Development Life Cycle: Objectives and Requirements*. Retrieved from Bender RBT: http:www.benderrbt.com/Bender-SDLC.pdf.

Biggerstaff, S. C. (2007). *Human Social Culture Behavior Modeling (HSCB) Program*. Retrieved from National Defense Industrial Association: https://ndiastorage.blob.core.usgovcloudapi.net/ndia/2007/disrupt/Biggerstaff.pdf

Boehm, B., & Hansen, W. J. (2001, April). *The Spiral Model as a Tool for Evolutionary Acquisition*. Retrieved from Software Engineering Institute Carnegie Mellon University: http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00sr008.pdf

Boyde, J. (2014). *A Down-to-Earth Guide to SDLC Project Management 2nd ed*. San Bernardino: Createspace Independent Publishing.

Brown, D. W. (2001). *An Introduction to Object-Oriented Analysis: Objects and UML in Plain English*. New York: John Wiley & Sons.

Bruegge, B., & Dutoit, A. H. (2004). *Object-Oriented Software Engineering Using UML, Patterns, and Java 2nd ed*. New Jersey: Prentice Hall.

Bunting, J. (2012, December 11). *How We Reduce Costs And Waste By Aligning Testing With The SDLC. (Optimation)* Retrieved from Optimation: http://www.optimation.co.nz/our-work/blog/how-we-reduce-costs-and-waste-by-aligning-testing-with-the-sdlc.

Cho, J. (2008). Issues and Challenges of Agile Software Development with SCRUM. *Issues in Information System*, IX(2), 188–195.

Cockburn, A. (2002). *Agile Software Development*. Boston: Addison-Wesley Longman Publishing Co.

Council, W. I. (2014). *Chapter 2 System Development Life Cycle Methodology*. Retrieved from Western India Regional Council: https://www.wirc-icai.org/material/2-System-Development-Life-Cycle-Methodology.pdf.

Curtis, B., Krasner, H., & Iscoe, N. (1988, November). *A Field Study of the Software Design Process for Large Systems*. pp. 1278–1287.

Davis, A. (2014). A Comparative Study of Prescriptive Process Models. *VISTAS*, 3(1), 94–100.

Dawson, M., Leonard, B., & Rahim, E. (2015). Chapter 16 Advances in Technology Project Management: Review of Open Source Software Integration. In *Technology, Innovation, and Enterprise Transformation* (pp. 313–324). Hershey: Business Science Reference.

Devlin, K. (1993). Naive Set Theory. In *The Joys of Sets: Fundamentals of Contemporary Set Theory* (pp. 1–28). New York: Springer-Verlag.

Diller, A. (1994). *Z: An Introduction to Formal Methods*. New York: John Wiley & Sons, Inc.

Dorsey, P. (2000). *Top 10 Reasons Why System Projects Fail*. Retrieved from Dulcian: www.dulcian.com/articles/dorsey_top10reasonssystemsprojectsfail.pdf.

Drake, F., & Singh, D. (1996). *Intermediate Set Theory*. New York: John Wiley & Sons.

Dwivedi, A. K., & Rath, S. K. (2012). Model to specify real time system using Z and Alloy languages: A comparative approach. *International Conference on Software Engineering and Mobile Application Modelling and Development (ICSEMA 2012)* (pp. 1–6). Chennai: IEEE. DOI: 10.1049/ic.2012.0149.

Easterbrook, S. (2001). *Software Lifecycles*. University of Toronto Department of Computer Science.

Elliott, G. (2004). *Global Business Information Technology: An Integrated System Approach*. Pearson Addison Wesley.

Estabrooke, I. (2009). *Directions for Human, Social, Cultural and Behavioral Sciences at the Office of Naval Research*. Retrieved from iBrarian: http://www.ibrarian.net/navon/paper/Directions_for_Human__Social__Cultural_and_Behavi.pdf?paperid=15892619.

Fletcher, J.-D., Garcia, S., Nwachukwu, K., Nwaogu, O., & Reaves, K. (2014). *Z to Alloy Kitchen Environment.* Baltimore.

Goldrei, D. (2005). *Propositional and Predicate Calculus*. London: Springer-Verlag.

Gries, D., & Schneider, F. B. (1993). *Predicate Calculus. In A Logical Approach to Discrete Math* (pp. 157–177). New York: Springer-Verlag.

Harris, S., & Bronner Ph.D, L. (2018). Extension of the System Development Life Cycle (SDLC) for the Analysis of Complex Problems. *International Journal of Science Technology and Engineering*, 102–107.

Hartley, D. (2009). *Background*. Retrieved from DIME/PMESII Community of Interest: https://home.comcast.net/~dshartley3/DIMEPMESIIGroup/DPGroup.htm

Heller, M. C., & Keoleian, G. A. (2002, May). Assessing the sustainabilty of the US food system: a life cycle perspective. *Agricultural Systems*, pp. 1007–1041. DOI: 10.1016/S0308-521X(02)00027-6.

Helmecke, M. R., Elmendorf, S. L., Kent, D. L., Pauze, D. K., & Pauze, D. R. (2014). Measles Investigation: A Moving Target. *American Journal of Infection Control*, 42, 911–915. DOI: 10.1016/j.ajic.2014.04.024.

Hungerford, D., Cleary, P., Ghebrehewet, S., Keenan, A., & Vivancos, R. (2013). Risk Factors for Transmission of Measles During an Outbreak: Matched Case - Control Study. *Journal of Hospital Infection*, 86, 138–143.DOI: 10.1016/j.jhin.2013.11.008

IDC. (2014). Executive Summary Data Growth, Business Opportunities, and the IT Imperatives. Retrieved from *The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things* April 2014: http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm

Institute of Medicine (IOM). (1988). *The Future of Public Health*. Washington, DC: The National Academic Press. DOI: 10.17226/1091.

Jackson, D. (2002, February). *Micromodels of Software: Lightweight Modelling and Analysis with Alloy*. Cambridge, MA.

Jackson, D. (2012). *Software Abstractions*. Cambridge: The MIT Press.

Jones, C. (1998). *Estimating Software Cost*. New York: McGraw Hill.

Jones, C. (2005, 04). Software Cost Estimating Methods for Large Projects. *CROSSTALK The Journal of Defense Software Engineering*, 8-12. Retrieved from Jones, Capers. "Software Cost Estimating Methods for Large Projects." CrossTalk: April, 2005. <www.stsc.hill.af.mil/crosstalk/2005/04/0504Jones.html>

Kassel, G., & Smith, G. (2001). Model Checking Object-Z Classes: Some Experiments with FDR. *Proceedings Eighth Asia-Pacific Software Engineering Conference* (pp. 445–452). Macao, China: IEEE. DOI: 10.1109/APSEC.2001.991453.

Khurana, G., & Gupta, S. (2012). STUDY & COMPARISON OF SOFTWARE DEVELOPMENT LIFE CYCLE MODELS. *International Journal of Research in Engineering & Applied Sciences*, 2(2), 1513–1521.

Kumar, N., Zadgaonkar, A. S., & Shukla, A. (2013, March). Evolving a New Software Development Life Cycle Model 2013 with Client Satisfaction. *International Journal of Soft Computing and Engineering (IJSCE)*, 3(1), 2231–2307.

Laney, D. (2001). *3D Data Management: Controlling Volume, Velocity, and Variety* . Stamford: META Group Inc.

Larman, C., & Basili, V. R. (2003, June). Iterative and Incremental Development: A Brief History. *IEEE Computer Society*, pp. 1–11. DOI: 10.1109/MC.2003.1204375.

Leau, Y. B., Loo, W. K., Tham, W. Y., & Tan, S. F. (2012). Software Development Life Cycle AGILE vs Traditional Approaches. *IPCSIT*, 37, 162-167.

Lehman, M. M. (1969*). The Programming Process*. Yorktown Heights: IBM Research.

McLean Ph. D, H. (2012, April 20). Measles — United States, 2011. *Centers for Disease Control and Prevention Morbidity and Mortality Weekly Report*, 61(15), pp. 253–257. Retrieved from http://www.cdc.gov/mmwr/preview/mmwrhtml/mm6115a1.htm

Nagpal, K., & Chawla, R. (2012). Improvement of Software Development Process: A New SDLC Model. *International Journal of Latest Research in Science and Technology*, 1(3), 217–224.

National Academy of Engineering. (2018, February). *14 Grand Challenges for Engineering in the 21st Century*. Retrieved from NAE Grand Challenges for Engineering: http://www.engineeringchallenges.org/challenges.aspx.

Newman, B. (2013, March 27). *Waterfall Method of Software (web) Development*. Retrieved from Agileana: http://www.inqbation.com/waterfall-method-of-software-web-development/.

NIH, N. I. (2002). *Mobilizing African American Communities to Address Disparities in Cardiovascular Health: The Baltimore City Cardiovascular Health Partnership Strategy Development Workshop Summary Report*. Washington, DC: National Institutes of Health.

Numrich, S. K., & Tolk, A. (2010*). Challenges for Human, Social, Cultural, and Behavioral Modeling*. SCS M&S Magazine, 1–9.

OMG, O. M. (2015). *Unified Modeling Language™ (UML®)* Resource Page. Retrieved from Object Management Group: http://www.uml.org/

Park, K., Ali, M., & Chevalier, F. (2011, July). A Spiral Process Model of Technological Innovation in a Developing Country: The Case of Samsung. *African Journal of Business Management*, 5(13), 5162–5178. DOI: 10.5897/AJBM10.1370

Ragunath, P., Belmourougan, S., Davachelvan, P., Kayalvizhi, S., & Ravimohan, R. (2010). Evolving A New Model (SDLC Model-2010) For Software Development

Life Cycle (SDLC). *International Journal of Computer Science and Network Security*, 10(1), 112–119.

Rouse, M. (2007, February). *Waterfall Model*. Retrieved from Tech Target: http://searchsoftwarequality.techtarget.com/definition/waterfall-model.

Rouse, M. (2018, 05 01). *Definition software development life cycle (SDLC)*. Retrieved from Search Software Quality: http://searchsoftwarequality.techtarget.com/definition/systems-development-life-cycle.

Salleh, M., Idzwan, M., Rosman, M., Rahimi, M., Raja, Y., & Yusoff, Z. (2011). Managing students' electronic disciplinary records via E-merit web content management system. *IEEE Conference on Open Systems, ICOS*, 261–266. DOI: 10.7763/JACN.2015.V3.175.

Saunders, C. (2014). *Project Management*. Retrieved from www.ou.edu/class/mis5003/mbapm.ppt

Seema, & Malhotra, S. (2012). Analysis and tabular comparison of popular SDLC models. *International Journal of Advances in Computing and Information*, 277–286.

Sharma, S., Sarkar, D., & Gupta, D. (2012). Agile Processes and Methodologies: A Conceptual Study. *International Journal on Computer Science and Engineering*, 4(5), 892–898.

Shelly, G. B., & Rosenblatt, H. J. (2012). *System Analysis and Design 9th ed*. Boston: Course Technology.

Shen, A., & Vereshchagin, N. (2002). *Basic Set Theory*. Providence: American Mathematical Society.

Shevin-Coetzee, M. (2016, February 6). *The Labyrinth within Reforming the Pentagon's Budgeting Process*. Retrieved from Center for a New American Security: https://www.cnas.org/publications/reports/the-labyrinth-within-reforming-the-pentagons-budgeting-process.

Solar, O., & Irwin, A. (2007). *A Conceptual Framework for Action on the Social Determinants of Health*. WHO Commission on Social Determinants of Health.

Spark, G. (2015). *Systems Enterprise Architect (EA) Software EAP File UML*. Retrieved from http://www.sirc.org.

Spivey, J. (1989). *The Z Notation, A Reference Manual*. New York: Prentice Hall.

Szalvay, V. (2004). *An Introduction to Agile Software Development*. Danube Technologies Inc.

Tagoug, N. (2012). Maintainability Assessment in Object-oriented System Design. *IEEE*.

Tarkan, S. (2009). *The Formal Specification of a Kitchen Environment*. University of Maryland.

Thierauf, R. J. (1999). *Knowledge Management Systems for Business*. Westport: Quorum Books.

Tsang, C. H., Lau, C. S., & Leung, Y. K. (2005). *Object Oriented Technology 3rd ed*. London: McGraw-Hill.

Tutorialspoint.com. (n.d.). *SDLC - Agile Model*. Retrieved from TutorialsPoint Simple Easy Learning: http://www.tutorialspoint.com/sdlc/sdlc_agile_model.

Vivancos, R., Keenan, A., & Farmer, S. (2012, July). An Ongoing Large Outbreak of Measles in Merseyside, England, January to June 2012. *Euro Surveill*, p. 20226. DOI: 10.2807/ese.17.29.20226-en.

Weiss, W. R. (2008, October). *An Introduction of Set Theory*. Retrieved from http://www.math.toronto.edu/weiss/set_theory.pdf.

Weston, K., Dwyer, D., & Ratnamohan, M. (2006). Nosocomial and Community Transmission of Measles Virus Genotype D8 Imported by a Returning Traveller from Nepal. *Commun Dis Intell*, pp. 358–365.

Winter, K., & Duke, R. (2002). Model Checking Object-Z Using ASM. In M. Butler, L. Petre, & K. Sere, *Integrated Formal Methods Lecture Notes in Computer Science, vol 2335* (pp. 165–184). Berlin, Heidelberg: Springer.

Wood, J., & Silver, D. (1995). *Joint Application Development*. New York: John Wiley & Sons, Inc.

Woodcock, J., & Davies, J. (1996). *Using Z: Specification*, Refinement, and Proof. Upper Saddle River: Prentice Hall, Inc.