A Modified System Development Life Cycle for the Analysis of Complex Systems Using the Formal Specification of Software for a Kitchen Cooking Application

Shanelle M. Harris, LeeRoy Bronner Ph.D., P.E.
Morgan State University

***Abstract*— With the large complex problems facing 21st century researchers, such as Engineering Education, Retention, Big Data, Cyber Security, Human, Social, Culture, Behavior (HSCB), Urban Resilience and Sustainability, there is a need for new research methodologies. The purpose of this research is the development of a methodology to address and analyze large complex systems. The System Development Life Cycle (SDLC) is a standard methodology used to analyze and solve system problems. However, current complex problems are requiring a more in-depth approach to problem solving. To address this problem, a modified version of the SDLC process is presented. Also, a number of analysis technologies are presented. For example, Object-Oriented Analysis (OOA) is used to develop a static model of a problem (i.e., define the problem vocabulary). To provide a generalized formal specification of a problem and its solution, the Z-language based in set theory and predicate calculus is being used. Also, to provide an approach that seeks to validate a problem solution, the Alloy Language is presented.**

***Index Terms*— Alloy Language, Object-Oriented Analysis (OOA), Object-Oriented Models (OOM), Set Theory, Specification Language, System Development Life Cycle (SDLC), Z-language**

I. Introduction

SINCE the turn of the century there has been a large increase in the number of complex problems. Many of the current methodologies lack a complete solution to support researchers. Understanding the lack of attention to detail, in many instances, solving social problems can lead to partial solutions that cannot keep up with the pace of demands created by the problems. The purpose of this research is the development a methodology to address and analyze large complex social problems. This paper proposes a new methodology as well as an integration of technologies and tools. The System Development Life Cycle (SDLC) is being modified to better analyze and solve software and system problems. With the introduction of mixed analytical techniques, object-oriented analysis and a generalized formal model specification, a more robust problem analysis and solution is being proposed.

II. System Development Life Cycle

The System Development Life Cycle (SDLC) is a term used by software and system engineers as a process for planning, creating, testing and deploying a system. The fundamental SDLC process is the waterfall Fig. 1. The waterfall is a predictive model flow of sequential phases where the outputs of stages are the inputs to the preceding stage [1]. As shown in Fig. 1, the process flow is project selection, project planning, analysis, design, implementation and maintenance. The objective of the SDLC is to ensure a high quality product is delivered while reducing inherent risk [2].

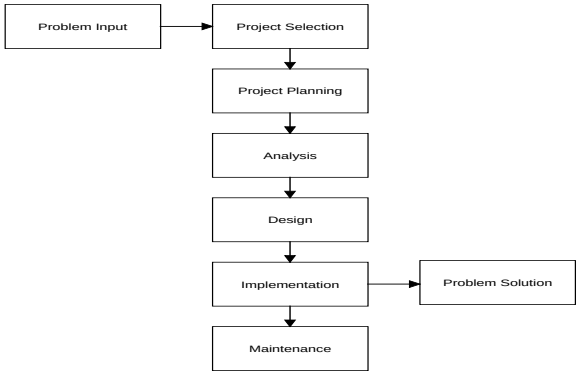

Fig. 1: Standard System Development Life Cycle

The planning phase determines the high levels of the project used to generate goals. System analysis defines functions based on redefining the project goals and analyzes the end user information. During system design, the features and operations of the system are described in detail. Implementation is the execution of all the detailed planning work followed by compiling and checking for errors and interoperability. Lastly, maintenance follows the production and distribution of the project and is for corrections, additions and updates.

The SDLC has proven to assist in many endeavors; however, with the increasingly complex problems being faced the limits of SDLC are exposed. It has been documented that the maintenance phase can take up to 80 percent of a projects' effort [1] [3]. The waterfall process lacks the flexibility, customer involvement, and misalignment of customer's expectations. It assumes the customer is only involved in the early phases and updates are not made throughout the entirety. There is no place to change during implementation if the scope has changed. This would cause a creep in the scope leading to a solution that the user does not require. The lack of flexibility and customer involvement culminate in misalignment of customer's expectations.

III. Modified System Development Life Cycle (MSDLC)

To address more complex problems, along with being able to evaluate the validity of the solutions, a MSDLC process is being proposed. This proposal addresses the normal SDLC

This work was submitted for review on 10 October 2014.

Shanelle Harris is an Industrial and System Engineering student at Morgan State University, Baltimore, MD 21254, (email: shhar3@morgan.edu).

LeeRoy Bronner is now in the Industrial and System Engineering Department at Morgan State University, Baltimore, MD 21254.

steps with a slight modification to the general flow replacing project selection and planning with problem definition and the use case model and maintenance with system evaluation. The proposed MSDLC has added a Joint Application Development (JAD) [4] session and continuous user involvement. However, the major changes to the SDLC are introduced in the analysis step. All of these topics are discussed in the following sections.

A. *Continuous User Involvement*

One of the most important considerations in the solution of large complex problems is the users of the solution. Unlike the waterfall SDLC when users are met with in the project selection and project planning phases this modified approach has the user involved throughout each phase. It can be seen as an exit criterion to move onto the next phase. This is used to ensure the scope remains and to manage expectations. User involvement cannot be over emphasized.

B. *Joint Application Development (JAD) Session*

As shown in Fig. 2, the Joint Application Development (JAD) [4] session is the first step to be taken in the system analysis. During this step the users and any stakeholder to the project are engaged to discuss, define, and outline the problem to be addressed. Typically, a facilitator is used to keep the flow of discuss on topic. It is known that this is only an initial problem description, since the problem definition, in most instances, will change when more details are known. The JAD session will produce a High Level System Diagram (HLSD) as well as the initial problem definition in narrative. The purpose of a HLSD is to graphically depict the problem elements in a visual diagram for better understanding [4].

C. *Use Case Models*

Using the initial problem definition, the use case model is developed. A use case model is a set of diagrams that define the user's requirements for the problem solution. The model consists of actors, use cases, actor's communication with use cases, and use case scenarios. A use case is a sequence of transactions by a system that produce a measured result seen by the user and a sequence of actions a user must initiate in a system to achieve a goal [5]. The primary purpose of the use case model is to define the requirements of the system from the user's point of view. Using textual analysis the use case scenario is used to select classes and their relationships to produce class diagrams. This process completes the analysis of the use case model.

D. *System Analysis*

The analysis process has three critical phases: 1) object-oriented, 2) Z-Language, and 3) Alloy Language analyses.

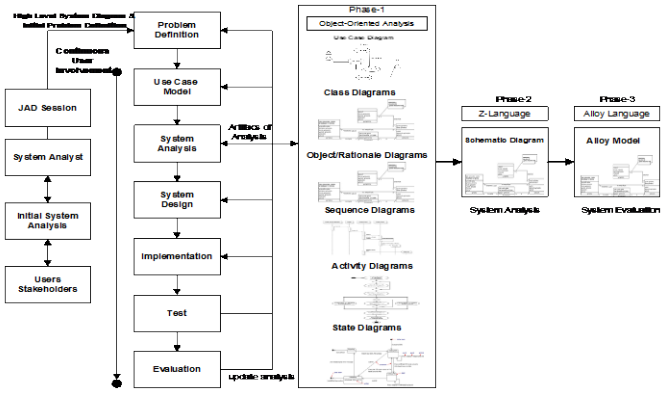

Fig. 2: Modified System Development Life Cycle Diagram

1) *Phase 1*

Phase 1 consists of the object-oriented analysis (OOA) [5]. During the OOA phase, the entire problem is researched and defined. By defined it is meant that the complete vocabulary of the problem will be established. The vocabulary of the problem includes the key entities that define the problem, their attributes and behaviors. For example, a student entity might be defined through its attributes that are germane to the problem, namely, "name, student ID, social security number, etc." and the operations this student is able to perform such as "study, research a problem, take a test, etc." The object-oriented analysis process produces artifacts such as class, object, rationale, sequence, activity, and state diagrams. This problem definition can be captured in the Enterprise Architect (EA) modeling tool designed to archive the analysis artifacts. In most cases, this will be a large database. The tool makes it possible to archive the problem definition and solution which maintains a complete index for searching and accessing the model components [6].

2) *Phase 2*

The second phase of the analysis is the Z-language model development.

a) *Z-Language*

The Z-Language is a highly expressive formal mathematical notation for specifying and designing the behavior of systems [7]. 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. The two advantages of the Z-Language are abstraction and the schema structure. Abstraction is the ability to describe what can be done without stating how it is done. The schema structure is a means of organizing its notation about the definition of the problem entities being analyzed.

To introduce a basic type in Z-Language, a given set can be defined such as UNIVERSITY. The set of all universities can be written as [UNIVERSITY]. If it is required that a variable be defined on the set of universities, the following nomenclature is used:

$$a : \mathbb{P}\ \mathrm{UNIVERSITY}\ ,$$

where $\mathbb{P}$ stands for the power set of universities. The power set is all subsets to include the empty set and the set itself [14].

Schemas are used to structure knowledge about a given entity within the problem. 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.

b) Set Theory

Set theory deals with sets, their operations, relationships and statements about these relations [8]. A set is a collection of different types of elements. Sets can be defined as a collection M of definite and distinct objects m of our intuition or thought (which will be called the 'element' of M) into a whole [15]. Simply set theory is "the ability to regard any collection of objects as a single entity" [14]. There is the set of natural numbers (i.e., {0,1,2,3, …}). Also, there are two other very common sets: the set of integers: { … -3,-2,-1,0,1,2,3, …} and the set of positive integers {1,2,3, …}. Shen [8] discusses some of the types of statements that can be used to describe sets are:

a. If x is an element contained in a set A this may be written $x \in A$, which states x is a member of A. Normally, in set theory notation lowercase letters are reserved for members of a set whereas upper case letters are used to define a set. Also, the notation uses {} to enclose the elements of a set.
b. In set theory, it is possible to define a subset within a set. A statement describing a subset is $A \subseteq B$, which states A is a subset of B.

This paper is not to be a treatise on set theory, which is treated in much greater detail in Shen's work. However, set theory is part of a two part foundation including predicate calculus for Z-Language which is the basis in this paper for modeling and analysis of complex systems.

c) Predicate Calculus

Predicate Calculus can be used effectively to express relationships and operations between components of a system. Goldrei [10] defines predicate calculus as "the operations will usually consist of inputs, outputs, and changes to the state of the system. The relationship between input, output, and the before-state and after-state will be described by a predicate relating and constraining these values." Predicate calculus can be used to describe all aspect of a system's functions. Predicate calculus is a branch of mathematics that uses terms and symbols to connect them to form logic statements [10]. In predicate calculus a predicate is applied to a set of terms. Terms are simply names for objects in the logic statement. Also, when a predicate is applied to terms the results of the relationship can be either true or false. The standard connective symbols for logic statements are, namely, ($\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$) whereas the quantifiers are $\exists$ *and* $\forall$. Beckert [9] provides the symbol definitions below in descending order of operator precedence:

$\neg$ negation,
$\wedge$ conjunction,
$\vee$ disjunction,
$\Rightarrow$ implication,
$\Leftrightarrow$ equivalence

The quantifiers are defined as:

$\exists$ there exists,
$\forall$ for all.

Three other very key operators in developing statements in predicate calculus are:

$\cup$ set union,
$\cap$ set intersection,
$\setminus$ set difference.

These are just a few of the symbols available to be used as operators in predicate calculus statements [10]. Through predicate calculus a very rich environment of logic definitions can be presented.

d) Formal Specification Language

The key innovation in the modified methodology is the development of a general formal specification modeling process for complex systems (Fig. 3).The process produces Z-Language schematic diagrams and Alloy models as artifacts.

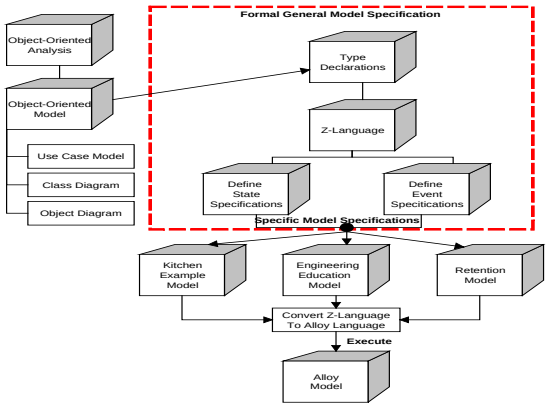

Fig. 3: Specific System Formal Specification Development Process

By using formal language specification methods, a general event-based specification modeling language has been defined. A formal specification is defined as a specification that "uses mathematical notation to describe in a precise way the properties in which a system must behave, without unduly constraining the way in which these properties are achieved [7]." These properties describe what a system must do without saying how it is to be done. This abstraction makes formal specifications useful in developing real world systems." Thereby an event-based modeling language, system operations are modeled in a discrete sequence of events in time. Each event occurs at a particular instant in time and may or may not produce 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.

To clearly understand this general modeling approach, examples from the "Formal Specification of a Kitchen Environment" [11] will be used. For example, base modeling objects in the Kitchen Specification would be the Kitchen, Timer, InitKitchen, and InitTimer schemas that contain cooks, kitchen items, cooking ingredients, recipe, etc. The schema named "Kitchen" is the primary schema in the example. Some of the supporting objects would be entities defining the events taking place while cooking in the kitchen, such as, cook, items, ingredient, and Recipe. The object items can hold the state of available, dirty or heated. An example of one of the criterion in the Tarkan's kitchen example is:

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

The statement says for all t (where t associated with time) of type N, the intersection of the domain of AvailableItems at time t with the domain of DirtyItems is equal to the empty set. In essence, the statement is being made that a kitchen item cannot be available and dirty at the same time.

The formal model specification is initiated by defining the basic types used for describing the state space of the problem. For example, the kitchen specification contains a number of Primitives and Atoms for defining the state space of the Kitchen Example (Fig. 4). A few of them are outlined below.

Primitives:
KITCHEN_ITEM == KNAME x N
MEASUREMENT == MNAME x N

Atoms:
ENAME (event name): bakeDone | cleanDone | cookDone | cutDone | kneadDone | mixDone | putDone |
MNAME (measurement name): teaspoon | tablespoon | cup | pint | quart | ounce | pound | package | [11]

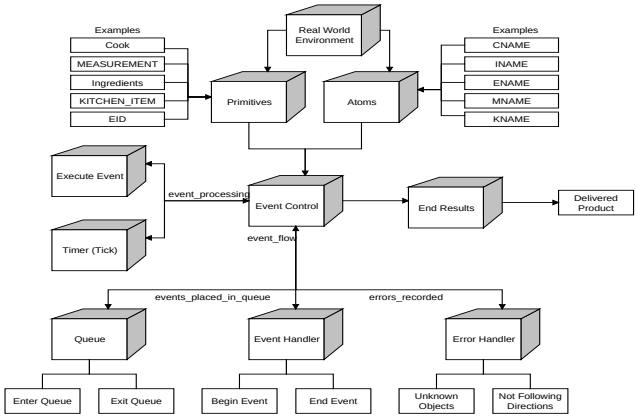

Fig. 4: General Model for Development of Formal Specifications of Complex Systems

The state of the system is defined by schema that establishes the state of the problem. In the kitchen example, the key schema that define the state of the system are the Kitchen, Timer, InitKitchen, and InitTimer. In this example, the Kitchen schema contains the primary data elements for the problem. Since all events are governed by time, the Timer schema controls the beginning and end of all kitchen events (e.g., Cook, CookDone). For example, some kitchen events are the cut, cutDone, mix, mixDone. These and other events make it possible to cook a meal. The kitchen example has an Event schema that includes the Kitchen schema where the state of the kitchen can be changed.

Schemas are the structures used to define the states of entities in a model in the form of a declaration section, criteria section and name [7]. The declaration section defines the state variables and other supporting schema. Attributes and behaviors in object diagrams may be used to define types in declarations. The criteria section defines the relationships that govern the declared variables. Rationale diagrams can represent the reasoning that leads to the systems functionality and implementation. This diagram supports decision making and captures knowledge [12]. The information provided in rationale diagrams can be used to set criteria.

3) Phase 3

Phase 3 is the Alloy modeling and system evaluation part of the MSDLC methodology. Alloy is a modeling language for expressing complex structural constraints and behavior about systems. Also, Alloy is a declarative specification language modeling tool employing first order logic based on the Z-language [13]. 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. 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 is able to analyze the model at every step. The purpose of converting Z-Language to Alloy is to use Alloy to find and correct errors in the Z specification. The converted Z-Language can model aspects of the system but not the entire system. Alloy makes it possible to check the criteria of the specification to assure correct execution of the solution [13].

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 through the use of three elements, logic, language, and analysis. 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). 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 instance 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. 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, a large number of instances can be run to analyze the problem.

E. Design

Design is the process that follows the analysis of a system. The analysis phase focuses on the system as a conceptual entity. The design portion answers the question of how a

problem should be solved and not what should be used in the system solution. Design is concerned with the physical aspects of the system. For example, design will address the real world system specifications. At the design stage, the analyst is focused on the developer and providing him with a clear picture of the physical system to be implemented. It should be noted that the system design is driven by technical problems and becomes the model for construction and implementation of the system.

F. Implementation

After system design is completed and approved the implementation phase can begin. Implementation is the process of construction and installation of the system design. In some circles this process is call the "roll-out" or placing the system in production. This production system can be a sociological system in a community or an educational system in academia as well a manufacturing system in a plant. The system development process set forth in this paper applies to most any type of a system where there are knowledgeable experts to define and aid in the development of the system. As pointed out earlier the user or human element is critical to system modeling, analysis and implementation.

G. Test

Testing is a very critical part of system development. Complex systems usually have many subsystems. Testing involves subsystem testing as well as testing of the entire system. Development of relevant test scenarios is a very import part of the testing process. It is very important that the development of test scenarios is done in close concert with the system users. Any test scenario, data, and best practices developed during the design phase can be very helpful to the system test process. As shown in Fig. 2, the results of testing can be used to make necessary changes to the design, analysis, use case model or the problem definition steps.

H. Evaluation

In the use case model development process, the system specifications are defined. Using the system specifications, they are evaluated against the system solution. The questions to be answered through this evaluation are the functions of the system meeting the requirements of the specifications. As with the testing phase, the results of evaluation can be used to make changes in the other steps of the system development process.

IV. CONCLUSION

Analysis and solution of complex system problems pose a great problem for the 21st century. It will require new research methodologies, new tools and very innovative approaches to problem solving. Problems in this decade will be large and very data intensive. Also, modeling and evaluation must move to another level from instance evaluation to a proof of solutions. A more rigorous validation of solutions will make performance of production systems more stable and error free.

To meet this challenge, it will require close collaboration between industry and academia. Academia must address real world problems and industry needs the talent and resources of academia. Therefore, as we move into the 21st century there must be a close partnership between academia and industry to address these complex problems.

V. REFERENCES

1. Bender RBT Inc. (2003). *System Development Life Cycle: Objectives and Requirements.* [Online]. Available: http://www.benderrbt.com/Bender-SDLC.pdf.
2. Mohd Arif, R.; Khalifa, O.O., "Online tutoring system in college: Case study in private education," *Computer and Communication Engineering (ICCCE), 2012 International Conference*, vol., no., pp.608-611, Jul. 2012.
3. Paul Dorsey, "Top 10 Reasons Why System Projects Fail". Dulcian Inc. 2000.
4. J. Wood and D. Silver. *Joint Application Development.* New York: John Wiley & Sons, Inc. 1995.
5. Curtis H. K. Tsang, Clarence S. W. Lau, and Ying K. Leung. *Object-Oriented Technology 3rd ed.* London, McGraw-Hill, 2005.
6. G. Spark. Systems. Enterprise Architect (EA) Software .EAP File UML. Internet: http://www.sirc.org/.
7. J. M. Spivey. The Z Notation, A Reference Manual. New York: Prentice Hall, 1989.
8. A. Shen and N. K. Vereshchagin. *Basic Set Theory (Student Mathematical Library, V.17).* Providence, Rhode Island: American Mathematical Society, 2002.
9. Bernhard Beckert. "The Specification Language – Formal Specification of Software." Internet: http://formal.iti.kit.edu/~beckert/teaching/Spezifikation-SS04/11Z.pdf
10. Derek Goldrei. *Propositional and Predicate Calculus.* London, England: Springer-Verlag, 2005.
11. Sureyya Tarkan, "The Formal Specification of a Kitchen Environment," M.S. thesis, University of Maryland College Park, Maryland. 2010.
12. Bernd Bruegge and Allen H. Dutoit, "Rationale Management," in *Object-Oriented Software Engineering Using UML. Patterns, and Java*, 2nd ed. New Jersey: Prentice Hall, 2004, pp. 488-529.
13. Daniel Jackson. *Software Abstractions*. Cambridge: The MIT Press, 2012.
14. Keith Devlin. "Naïve Set Theory" in *The Joy of Sets: Fundamentals of contemporary Set Theory*. New York: Springer-Verlag, 1993, pp. 1-28.
15. F. R. Drake and D. Singh. *Intermediate Set Theory*. New York: John Wiley and Sons LTD, 1996, pp. 1-36.