An Ameliorated Methodology to Implement UML Syllogism in Suigeneris Software Testing

Dr. Shivanand M. Handigund
Dept. of CSE, Bangalore Institute of Technology, Bangalore-560004, India.

Dr. Ajeet Chikkamannur
Dept. of ISE, New Horizon College of Engineering, Bangalore-560004, India.

Arunakumari B N
Dept. of CSE, Visvesvaraya Technological University - RRC, Belgavi, 590018, India

Shakuntala Sajjanar
Dept. of CSE, Bangalore Institute of Technology, Bangalore-560004, India.

***Abstract*—The current software testing technique test the programming system for correctness, robustness, and efficacy through different test cases and test oracles. Normally, testing requires the test of correctness, completeness, robustness, efficiency and efficacy. So, even the perfectionist tester test 60% of features. In our au-courant software testing technique test is carried for 100% out of which a duende of top down and bottom up constellation of Unified Modeling Language (UML) design diagrams test 80% of features. This paper presents a Suigeneris methodology for the 80% of the features fully utilizing the constellation property of UML. The technique delineates the higher level diagram into number of intaglios and uses each intaglio to test the pragmatics of lower level diagrams. This will be carried out for all constellation hierarchy of diagrams. The approach attempts to highlight how 80% testing is made free from arbitrary human skills. Moreover, the testing technique does not require the domain knowledge and the sample data.**

Keywords—Reverse engineering, UML semotics, Usecase levels

I. INTRODUCTION

Vision: To utilize the multilevel Unified Modeling Language (UML) design diagram constellation property in software testing

Mission: To abstract multilevel design information in the form of UML diagrams and to test the programming system through the establishment of constellation using top down and bottom approach in the juxtaposed way.

Objectives:

- To test correctness of the programming system through the design of control flow and data flow graphs (Correctness)
- To carry out behavioral and intercommunication test through the establishment of constellation between the consecutive levels. (Completeness)
- To reorganize the programming system into three different perspective views to enhance structuredness and modularity of the programming system (efficiency).
- To use top down and bottom up approaches interleaving for the abstracted design and testing through for constellation delineations (efficacy).

A. Motivation

UML was developed by three amigos [1] and is considered as de-facto standard design language. The concinnity of UML design diagrams in the different hierarchical levels contains the different hierarchical levels of abstract information this enables the documentation. Testing the UML lower level diagrams correspond to unit testing of the programming system, in the middle level related integration testing and in the higher level will carry out the system testing. The synecdoche of semiotics of UML designed such way that cluster of syntactics and semantics of higher level diagram represent pragmatics of immediate lower level. And the syntactics and semantics each diagram is complement to some other diagram or a diagram may enhance the features of other diagram by this if error could not be identified from one diagram it can be identified from other diagram. In the present testing technique test is attempted somehow the correctness, robustness and efficacy but may not be 100% but they not at all attempted the completeness and efficiency. The existing software testing techniques test maximum up to 60% of features. Even though we use all techniques of connoisseurs we cannot test programming system 100% and if experts also cannot check 40% of the programming system. Thus, in the present testing more than half of the programming system should not be performed. Querulous mind set of human being give attention to human being to identify errors in the different diagrams of UML.

The au-courant software testing needs different hierarchical levels of testing for testing the correctness, completeness, robustness, efficiency and efficacy. In this paper we achieve the correctness completeness efficacy and efficiency through syllogism of UML design diagrams within the design.

B. Literature Survey

The existing software testing techniques [2] are not correct, complete, robust, efficient and efficacy. The lack of completeness is evidenced by “the software testing identifies the presence of errors and not the absence of errors”. Current testing is carried out through test cases. The test cases are chosen intellectually by the querulous tester with “the less number of test cases to identify more number of errors”. Thus, the success of the current software testing technique which tests at most 60% depends on the domain knowledge and human skill. Thus, the available software testing techniques are only labelled art. There is a need to transfer this art into engineering via science. The UML is the de facto standard design language. Unfortunately, the innovators have not defined correct pragmatics for their diagrams but UML can be irradiated as good engineering design discipline. However, the

syntactics and semantics of UML are well defined to suite state-of-the-art design language. If the transformation of programming system components into UML diagrams with well-defined unique pragmatics for each design diagram and each diagram constellates immediate lower level diagram. This can be used to test 100% feature like correctness, completeness, efficiency and efficacy. In this paper an attempt is made to utilize the constellation property for software testing. The automation of programming system into class diagram and then utilizing the constellation property in the forward and reverse direction we show how 80% of the software testing is carried out without need of domain knowledge, test cases and the human intelligence.

The control flow graph for the program decides the execution flow of the statements. Therefore this paper adopts the methodology developed by [3] but it was limited to COBOL programming language. The technique has been appropriately modified by [4] to suite 3rd generation programming language. In this paper we design control & data flow graphs for entire programming system and transformed into tabular form for checking the correctness of programming system. Slicing criteria was developed by Weiser and it was for the debugging purpose in the program [5]. It was further refined by [3] for the abstraction of object methods in case of programs. In our paper we used the same concept, but here we have used for the abstraction of different slices from the programming system.

II. TAXONOMY

- *Usecase:* it is a sequence of activities performed by the system for an actor on an occasion.
- *Work process:* Collection of activities for a particular information object class performed with tools & techniques with input from other work processes / actors, producing output to be flown to the work processes/ actors.
- *Work*: It is a Sequence of activities between two deliverables. Part of these deliverables may be from actor/work processes within the organization.
- *Syntactics:* The syntactic relates the symbols to the meaning. In UML the symbols are assigned to different model elements of the object oriented technology.
- *Semantics:* The semantics organizes the symbols of model elements into a meaningful group as per the semantic rules. It organizes different symbols into a meaningful entity as a part of pragmatics.
- *Pragmatics:* Pragmatics is the architectonic way of representing the perspective view of the information system.
- *Semiotics:* It's a trinity of syntactic, semantics and pragmatics. Each language (diagrammatic, spoken or programming) has well defined syntactic, semantics & pragmatics.
- *Class:* is a collection of cohesive set of attributes of an entity along with its behaviour in the form of objects methods that define the subset of its attributes. It comprises highly cohesive set of attributes of an entity along with methods that defines subset of attributes of the class structure which are very loosely coupled. Thus class satisfies good database design and good software engineering principals.
- *Object method*: It's a behavior thread of the object class in which subset of attributes of the class is defined.
- *Referenced item:* A noun/noun phrase is said to be referenced in the statement if it is explicitly present in the statement and its value remains unaltered by the realization of the statement.
- *Defined item:* A noun or noun-phrase defined in a program line if it is explicitly present in the statement with changed value after the realization of the statement. A noun/noun phrase is said to be preserved in the statement if its value is implicitly preserved in the statement.

III. PROPOSED METHODOLOGY

The au-courant software testing is correct, complete, robust, efficient and efficacy. Here, correctness of the programming system is any statement should be in one of the path from start to end and no data should be referenced without defining. Completeness of programming system is all the perspective views viz., usecase, work and work process in the business process are completely utilized and there should not be any gap between the usecases, work processes and works of the business process and also there should not be overlap between the different work processes and works but there may be overlap in usecase because usecase involves usecase hierarchies. Robustness is achieved through mutation testing let D is the number of mutants that is distinguished, E be the equivalent mutants and N is the number of mutants. i.e. $D/(N-E)\approx 1$ i.e. continue testing until the mutation score is nearly equal to 1 by random inter change of operators and test cases. Efficacy is for all user perspective views whether the software will satisfy. The difference between the program and software is nearly equal to zero. Efficiency is the structure and modular of the programming system. In this paper we use UML constellation property to achieve the software testing for correctness, completeness, robustness, efficiency and efficacy and the procedure is discussed below

Procedure for top down and bottom up approaches

i. Design control flow graph (CFG) and data flow graph (DFG) for entire programming system and store it in the table as shown below in table1.

Table 1: Control flow table

Start	End	Jump	Alternate jump
<stmt no>	<stmt no>	<stmt no>	<stmt no >

ii. Design higher level diagram with the following procedure

Input: Classing technique [3], slicing technique [4]
Tools used: CFG and DFG tool

- Identify class and class attributes using classing technique and identifies object methods of the class using the slices of the class. Then identify interrelationship through parameter passing from one object method to another object method. Then using this components design class diagram for programming system.

iii. Design middle level diagram viz., object diagram with the following procedure

Input: class diagram using constellation property

- There are two types of boundary object class. One boundary object class transforms interface attribute into system class attributes called source boundary object class and other transforms system class attributes to actor interface attributes called destination object class.

  If

  $$def(\cup \text{object methods of the class}) - \text{class attributes} \subseteq \text{actor interface attributes}$$

  then boundary object class is destination boundary object class

  If

  $$def(\cup \text{object methods of the class} - \text{class attributes} = \emptyset$$

  and if

  $$ref(\cup \text{object methods of the class}) - ref(\cup \text{object methods of the other classes}) \subseteq \text{actor interface attributes}$$

  The boundary object class is a source boundary object class.
- Take the boundary object class if it is destination boundary object class takes that boundary object class search for the object methods of other class where the referenced attributes of this boundary object class are defined. The referenced attribute may be carried out 0, 1, 2. . . n for object methods of other classes continuing this procedure compute the closure as shown in figure 1 the closure may end up in other boundary object class or boundary object class of itself. Specialty of object diagram is that it contains single object method of other class. Each path from destination boundary objects class to source boundary object class.
- If source boundary object class and destination boundary object classes are same then it is cyclic in such case unique usecase exist for an actor. If source boundary object class and boundary object class different then it is acyclic in such case that use case needed two actors.
- Cyclic contains synchronous and simple message.
- Redundancy identified by usecase hierarchy. Usecase hierarchies (levels) identified by figure 1 with the following procedure
  a. Group: If in addition to boundary object class definition, two or more such boundary object classes together define the interfaces of single actor then earlier usecases are grouped for single usecase for a new 'single actor'.

     $$def(\cup \text{object methods of the class}) \subset def(\cup \text{object methods of all boundary object classe})$$

  b. Extends: If boundary object class connect to another boundary object class with sequence of methods embedded between two actors then the usecase formed by the additional activities to form extends.
  c. Uses: If boundary object class of a usecase is extended on both sides with other activities marked by additional boundary object class then the enhanced use case uses previous usecase.
- If compare the object diagram to the higher level class diagram we will identify the completeness of object diagram. And shown in figure 1.

iv. Design lower level diagrams viz. sequence diagram

Procedure to design sequence diagram as follows

Input: Object diagrams constellation property

- Identify boundary object class with following procedure

  Boundary object class can be identified from two classes A is the class and A ≠B ∈ B is any other class. Then for all B

  $$\left[\text{Defined attribute of the class A}\right] \cap \left[\text{Referenced attribute of a class to be searched from DFT slice of the class}\right] = \emptyset \quad \forall \text{ DFT slice of different classes except A}$$

  Then A is destination boundary object class (BOC)

  $$\left[\text{Referenced attribute of each method of class A to be obtained from DFT slice of the class}\right] \cap \left[\text{Class attributes from all classes}\right] = \emptyset \quad \forall \text{ DFT slice of different classes except A}$$

Then B is the source boundary object class.

Identify sequence of activities using boundary object class for sequence diagram Categorization of message using data flow method in sequence diagram identified with following procedure

a. If consecutive sequence of two methods consists of common subset of attributes then synchronous message exist and acyclic boundary object class
b. If the defined attributes of two object methods is a proper subset of a particular single class and the first one in CF order then subsequent messages are simple message.
c. If attribute defined in one method and referenced in consecutive sequence of methods then subsequent messages are asynchronous messages.

v. Here time order in sequence diagram is compared with time order of object diagram. According to the DFT we identify time order of the messages for sequence diagram. This time order provides the categorization of messages which is used to provide sequence of activities. This used to verify the intercommunication between activities in the object diagram using bottom up approach. If there is any deviation in the messages it identifies the error in the diagram with respective to the program slice.

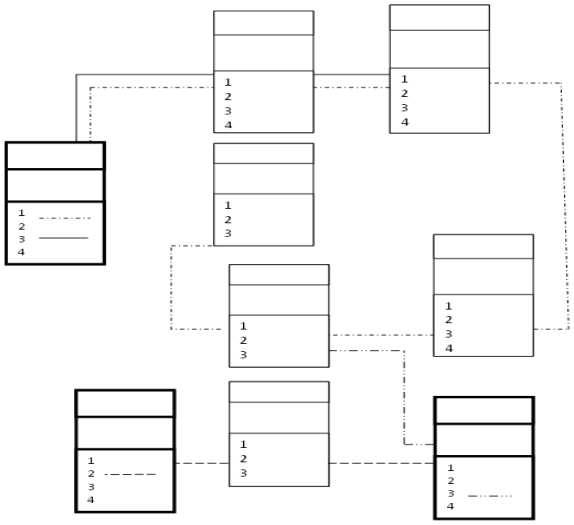

Fig. 1 Object diagram model

vi. Cluster of syntactics and semantics of higher level diagram defines the pragmatics at lower level diagram. Then we say that higher level diagram constellate the lower level diagram. Any aberration between the syntactics, semantics and pragmatics in the lower level diagram can be identified as error.

vii. Top down and bottom up approaches are interleaving verify the correctness and completeness in design diagrams.

viii. If we could not identify error in one diagram then we can observe errors in other diagram. Each method participates once in type of diagram.

ix. In our au-courant testing technique syntactics and semantics of programming language and pragmatics of the programming system are tested in a single phase, testing at design levels. Moreover, errors of the code are tested in the design stage which reduces semiotics of the programming language to the diagramming and the number of errors to be tested by 1/3rd of the original errors.

ACKNOWLEDGMENT

Words are insufficient to express our deep sense of gratitude to Dr. D. B. Phatak, Professor of Dept. of Computer Science & Engineering IIT Bombay, for his inspiration.

CONCLUSION

The need of arbitrary human skills, the acquisition of domain knowledge and sound semiotics of the programming language are necessary to test the 60% of testing viz., correctness, efficacy and robustness. The complete testing needs correctness, efficacy, completeness, efficiency and robustness. The available testing techniques test only 60% features. Thus, the current testing techniques test is trust worthy for less than 60% of testing. The completeness and efficiency have never been tested through the existing testing techniques. In this paper we have developed a methodology that eliminates the need of domain knowledge and the need of human skill and semiotics of the programming languages through reverse engineering the programming system to the multi-level UML. The constellation property of multi-level UML is used as stratificational grammar to test the correctness, efficacy, completeness, efficiency and robustness. In addition to testing the efficiency it also transforms existing programming system into structured modular according to user required work process based, work based and usecases based modulation.

REFERENCES

[1] Grady Booch, James Rumbaugh, Ivar Jacobson, The Unified Modeling Language User Guide (Second Edition) / Edition 2

[2] Pankoj Jalote “An Integrated Approach to Software Engineering” Third Edition, Springer, Narosa Publishing House Narosa: 81-7319-702-4.

[3] S. M. Handigund, Reverse Engineering of Legacy COBOL systems, Ph.D. thesis, Indian Institute of Technology Bombay, 2001.

[4] Shivanand M Handigund, Arunakumari B N, “An Ameliorated Methodology to Abstract Object Oriented Features from Programming System” has been published in proceedings of international conference of Soft Computing and Software Engineering [SCSE'15] to be held at University of California, Berkeley USA during March 5-6 2015.

[5] Mark Weiser Program slicing, in IEEE international conference in software maintains transaction of software engineering 1981.

[6] Shivanand M Handigund, Shakuntala Sajjanar, Arunakumari B N “Resuscitation of Syllogism Within UML Levels through the renovation of object diagram” has been submitted for presentation in the 4th International Conference on Advances in Computing, Communications and Informatics(ICACCI-2015) held during 10-13 August 2015 SCMS Aluva, Kochi, Kerala.