Implementation of an Environment to Analyze Object-Oriented Software and Quality Assurance

Devendrasingh Thakore, Akhilesh R Upadhyay

Abstract— Software quality cannot be improved simply by following industry standards which require adaptive/upgrading of standards or models very frequently. Quality Assurance (QA) at the design phase, based on typical design artifacts, reduces the efforts to fix the vulnerabilities which affect the cost of product. For this different design metrics are available, based on its result design artifacts can be modified. But to modify or make changes in artifacts is not an easy task because these artifacts are designed by rigorous study of requirements.

The purpose of this research work is to automatically find out software artifacts for the system from natural language requirement specification as forward engineering and from source code as reengineering, to generate formal models specification in exportable form that can be used by UML compliment tool to visually represent the model of system. This research work also assess these design models artifacts for quality assurance and suggest alternate designs options based on primary constraints given in requirement specification.

Following problems are resolved in this research work

1. Automatic generation of design phase class model from natural language input

2. Automatic generation of design phase class model from already developed source code

3. Generation of secure validated deign from above generated class models with different level of security as high low and medium with the help of different software metrics

To resolve these problem there is need of automated environment which will assess generated design artifacts from natural language as forward engineering and from source code as reengineering and finally suggest and validates alternate designs options for better quality assurance.

Index Terms— POS Tagging, OOA, UML, Use case, actor, software quality, quality metrics, XMI.

I. BACKGROUND

Many approaches and techniques have been proposed up till now to automate the process of various model generations from natural language requirement specification as well as from source code. However theses approaches are not used in real world system development due to their limitations.

CM-Builder [**2**] aims at supporting the analysis stage of development in an Object-Oriented framework. CM-Builder uses robust Natural Language Processing techniques to analyze software requirements texts written in English and build an integrated discourse model of the processed text, represented in a Semantic Network. This Semantic Network is then used to automatically construct an initial UML Class Model. The initial model can be directly input to a graphical CASE tool for further refinements by a human analyst.

Manuscript received on May, 2013.

Devendrasingh Thakore, Ph.D Scholar, Shri JJT University, Jhunjhunu, Rajasthan, India.

Dr. Akhilesh R Upadhyay, Vice Principal, Sagar Institute of Research and Technology, Bhopal, India.

Linguistic assistant for Domain Analysis (LIDA), provide linguistic assistance in the model development process. It presents a methodology to conceptual modeling through linguistic analysis. Then gives overview of LIDA's functionality and present its technical design and the functionality of its components. Finally, it presents an example of how LIDA is used in a conceptual modeling task **[3].**

NL-OOML **[4]** presents an approach to extract the elements of the required system by subjecting its problem statement to object oriented analysis. This approach starts with assigning the parts of speech tags to each word in the given input document. The text thus tagged is restructured into a normalized subject-verb -object form. Further, to resolve the ambiguity posed by the pronouns, the pronoun resolutions are performed before normalizing the text. Finally the elements of the object-oriented system namely the classes, the attributes, methods and relationships between the classes, the use-cases and actors are identified by mapping the 'parts of speech- tagged' words of the natural language text onto the Object Oriented Modeling Language elements using mapping rules. But approximately 12.4 % of additional classes and 7.4 % of additional methods are identified in all the samples taken each of around 500 words. These additionally identified candidates are those that will usually be removed by human by intuition. Since the system lacks this knowledge, they were also listed as classes. Coverage accuracy is 82%

"A Visual Analysis and Design Tool for Planning Software Reengineering", by Martin Beck, Jonas Trumper, Jurgen Dollner paper presents new visual analysis and design tool to support software architecture during reengineering process in identifying a given software`s design and in visually planning for quality improving changes to its design.

This visual analysis and design tool is applied to an existing software system to modify its representation without changing its behavior **[5].**

The main thing related this paper is a new concept for visual analysis and design that support to evaluation of design and identification of transformation. In the research paper Security metrics for object-oriented class designs", Alshammari, Bandar and Fidge, Colin J. and Corney, Diane show that security metric is not considered as much as other quality attributes such as complexity metrics. Also, most security studies concentrate on the level of individual program statements. Such type of approach makes it hard and expensive to discover and fix vulnerabilities caused by design errors in the existing system **[6].**

In future work, let focus on the security design of an existing object oriented application and define security metrics. These metrics allow designers (developer or system analyzer) to find out and fix security vulnerabilities at an early stage of the re-engineering process and it helps to designer review the security metrics to make particular decision about security into re-engineering approach. **[7].**

II. PROPOSED SYSTEM METHODOLOGY

The purpose of this thesis is to do the analysis of software design models generated from natural language software requirement specification in forward direction of software development lifecycle and from source code in case of software reengineering process to assure better quality of final product or system.

This research proposes approach to analyze, extract, transform and generate software artifacts from natural language business model and source code also to build the formal semantic models of the system. After that it will analyse these models and produce alternative design using genetic algorithms for further implementation. Proposed work then validate the chosen design option at code level also.

Proposed system is involves three modules, which works in manner as below.

MODEL I: - Extraction of Models from Natural Language Requirement

This model is used to identify the artifacts which are used to generate the models at analysis phase from natural language. This methodology consist of automatic conversion of natural language software requirement specification conversion to controlled intermediate SBVR format and secondly to identification of software artifacts and model generation, finally visualization of generated models. Used methodology works in different phases organized in pipelined fashion as follows.

1. Pre-process Analysis: This phase stars with the by reading the given English input and tokenizing the whole input in to individual tokens.
2. Tagging: This processed text is further given as input to Part Of Speech (POS) tagger to identify the basic POS tags.
3. Morphological Analysis: To remove the suffixes attached to noun phrases and verb phrases this type of analysis is performed on the tagged output from pervious phase. [7].
4. Parse Tree Generation: Stanford Parser is used to generate parse tree from pos tagged output for each requirements.
5. Role Labeling and Element/Concept Identification: In this phase role labels are identified from pre-processed text such as performer, co actors, events, objects and receiver in the sentences. Also in this phase SBVR concept identification is done according to some identification constraints such as all proper nouns are identified to individual concepts, all common nouns are identified as noun concepts or object type, all action verbs are identified as verb concepts, all auxiliary verbs are identified as fact types, possessed nouns are identified as characteristics or attributes, indistinct articles, plural nouns and cardinal numbers are identified as quantification.
6. Rule Generation: To generate the SBVR rule we have to first produce fact types, in the form of sentences which represents some relationships between the concepts identified in the previous phase. For that purpose use the template such as noun-verb-noun to establish the relationship between two concepts. Thus a fact type is created by combining the noun concepts and verb concepts from pervious phase array list. Generated fact type is used to create the SBVR rule by applying various logical formulations.
7. Artifacts Extraction: In this phase produced SBVR vocabulary and rules are further processed to extract the basic building blocks or artifacts of models such as use case and class diagram etc.
8. XMI Generation and Model Visualization: Finally in this phase the output of above phases are generated in the form of XML Meta data interchange (XMI) file format. Such file is further given as input to UML modeling tool having XMI import feature to visualize the generated models.

Model II: - Re-Engineering and Extraction of Software Quality Metrics

Here take the input from user (developer), a document which contains source code. Convert this source code document into .class file. With the help of BCEL library features, parse the .class file and identify the candidate 1) classes with their inheritance definition, 2) methods with called and calling nature and attribute with their access. After this identification of candidates then applies our rules on the basis of which Security Accessibility and complexity metrics are extracted.

Also recognize and pull out the **design view (Core Prototype Model)** of the source code of software system. Design view (class diagram) specifies the entities and relations that can and should be extracted immediately from source code. Then process this output for Complete Model as input specifies Object, Property, Entity and Association are made available to handle the extensibility requirement.

Step 1: Pre-processing

Java source code files with .java extension are converted firstly into .class file representation by compiling the .java extension source code file. This .class file contains the java type; it may be a class or an interface. Here .class file is used for parsing, analyzing and extraction purpose because it's independent nature.

Step 2: Identify Candidate Classes

After compilation of java source file with extension .java and .class file of source code is generated. Parse the .class file, which is represented in byte codes format, identification of candidate classes with their object, entities and their interrelationship i.e. determine the inheritance object-oriented paradigm definition of classes. Parse the .class file with the help of BCEL (Byte Code Engineering Library) [8] ClassVisitor java package which reads the content of .class file and find out the no. of classes, objects of that classes and inheritance definition of them. Identification of such candidate classes are used into evaluation and extraction of software quality metrics. Structure of ClassVisitor is as follows

Public class ClassVisitor extends
org.apache.bcel.classfile.EmptyVisitor

Step 3: Identify Candidate Methods

Parse the .class file, which is represented in byte codes format, identification of candidate methods with their visibility private, public and protected along with find out which method calling and which method called. This identification helps for counting no. of public methods into the source code of software.

Step 4: Identify Candidate Attributes

Parse the .class file, which is represented in byte codes format, identification of candidate attributes with their visibility private, public and protected along with find out which method access identified attributes. This identification helps for counting no. of attributes accessed by method of another class into the source code of software. Parse the .class file with the help of BCEL (Byte Code Engineering Library) MethodVisitor java package which reads the content of .class file and find out the no. of methods.

Step 5: Extract the software quality metrics from above 3 identified candidates classes, methods and attributes

This section describes the evaluation and extraction of software quality metrics of proposes methodology. According to this methodology the most enduring metrics of performance that have been applied to information extraction are termed as **Data Access Metric (DAM)**, **Operation Access Metric (OPM), Coupling** and **Cohesion**.

These metrics may be viewed as judging effectiveness from the application user's perspective, since they measure the direct accessibility of classified instance attributes of a particular class. It helps to protect the internal representations of a class, i.e. instance attributes, from direct access (DAM), statically measure the potential flow of information from an accessibility perspective for an individual object-oriented class (OPM) and measure independency of each class as complexity. In the case of information extraction:

DAM = No. of private (protected) attributes/total no. of attributes in class

OPM = No. of public methods/ total no. of methods in a class

Cohesion = No. of methods interactions with attributes in the program code / maximum no. of methods interactions with attributes

Coupling = Access frequency of attributes of one class/sum of frequency of all attributes

Step 6: Extract the design view (class diagram) of source code from above 3 identified candidates classes, methods and attributes

Model III: - Identification of validated Secure Design for Quality Assurance

This Model is involves three steps, which works in sequential manner as below

Step 1- Applying Genetic Algorithm (GA) [9]

This step accepts UML class diagram as a input and applies Genetic algorithm on it. Here GA is used to obtain more than one design which fulfills different levels of fitness function. These alternate designs are of three levels of security i.e. High secure, medium secure, low secure.

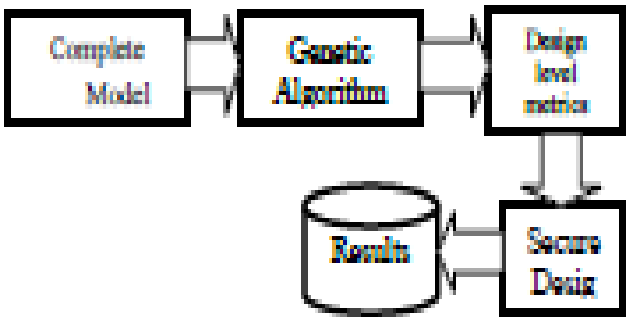

Fig 1: Application of GA and Generation of Secure Designs

Step 2- Code Evaluation

In second module code is implemented for highly secure or medium or low secure design .Then this implemented code get evaluated based on the security related same coupling metrics. Means it checks coupling aspect at code level to provide security.

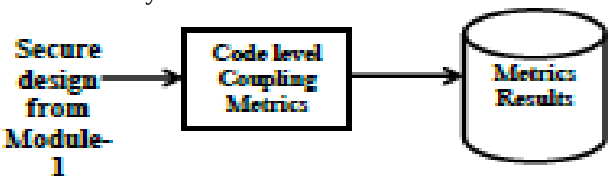

Fig 2: code evaluation by applying coupling metrics

- USE OF COUPLING IN SECURITY METRICS

Many studies have been shown that coupling is closely associated with the Security of software. The reason behind this is when information passed among different components there is more probabilities that it is exposed. It makes sense to assume that coupling is important factor that affects security of software. Following are coupling metrics that can be devised in proposed system.

RFCSec (Response for a Class), WMC (Weighted Methods per Class), CBO Sec (Coupling between Object Classes), Depth of Inheritance Tree (DIT), Number of Children (NOC), COF Sec (Coupling factor), DAC Sec (Data Abstraction Coupling), CaSec (Afferent couplings) Export, CeSec (Efferent couplings) Import, MPCSec (Message Passing coupling), MICSec (Method Invocation) [10]

Step 3- Validation

Third module is nothing but the validation of all designs. It compares metrics results computed at design level and code level to generate the graph. Spider chart and line graph gives the difference between both the results to validate the design .It means tool facilitate user to check the selected design gives appropriate results at design level and at code level or not.

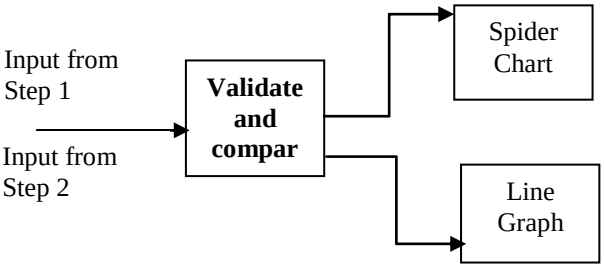

Fig 3: Validation of Metrics

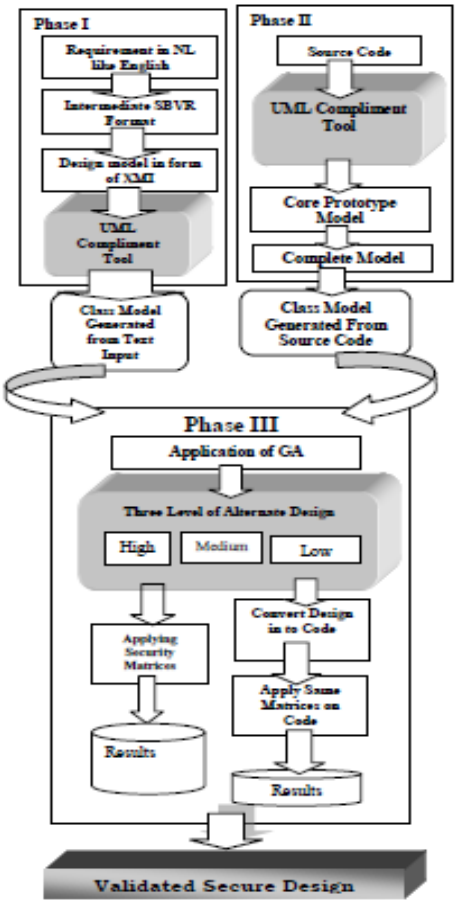

Fig 4: Combine Architectural Diagram of Proposed Environment

III. IMPLEMENTATION DETAILS

Implementation screens of this framework to produce secure design explained below along with example.

Step 1: The text file contains requirements in natural language is as shown in figure 5 and is given as input to the module I

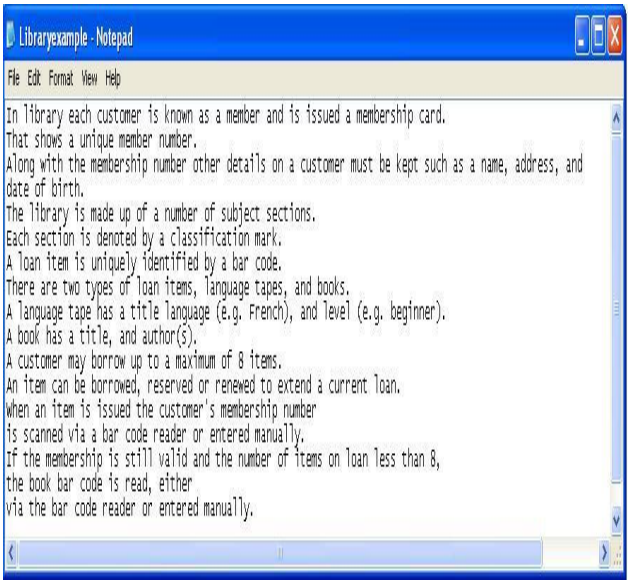

Fig 5: Input to model I

Once the input is accepted by the system, it converts the given input requirements in to SBVR format for better accuracy. It is shown in figure 6 as below.

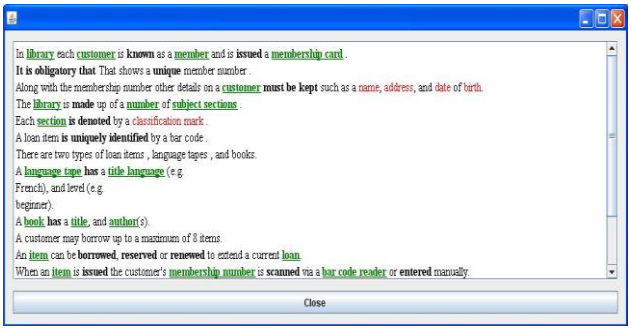

Fig 6: SBVR Format

This SBVR Representation is further scanned, analyzed to extract the software artifacts such as classes, their attributes to generate the class model which is shown in figure 7 as below.

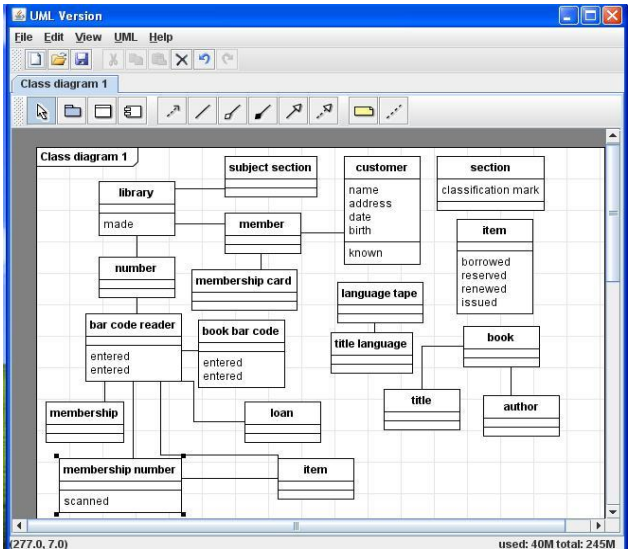

Fig 7: Extraction of the software artifacts

Similarly input is given to module II in the form of already developed and compiled code in the form of .class file and generates the class models as shown below in figure 8

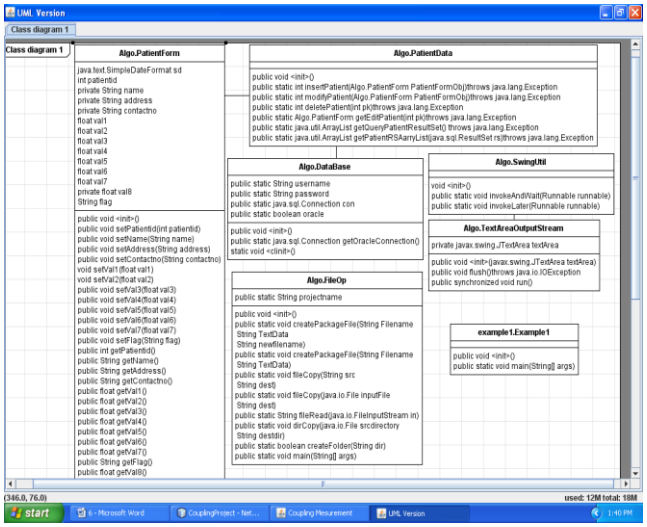

Figure 8: Class model generated from .class file

After this point this framework will work in two execution path options as below

1. **From module one to module three**
   In this execution path it takes natural language as input and converts it into complete class model as the output of module one then this output is given as input to module three which will produce validated secure design options.
2. **From module two to module three**
   In this execution path module three takes the input as .class files of already developed software project and from such input complete class model is extracted as intermediate output. Such type of output produced is given as input to module three which will produce the validated secure design as final output.

IV. CONCLUSION

This framework describes a computerized way to take out the software element at analysis phase. It uses NLP techniques to consider business level software requirements and builds an incorporated analysis level model. This approach can be used for the identification of software elements such as, use cases, classes, their attributes, and the static relationships among them with increase in accuracy due to use of intermediate format.

The quality of software must be defined in terms that are meaningful to the users. Recovering these types of metric is the reverse engineering which is first step towards the reengineering. These software metrics can be used in a Re-engineering of software development process to improve the design of the implementation. But the quality of software systems heavily depends on their structure, which affects maintainability and readability. However, the ability of humans to deal with the complexity of large software systems is limited. Also it is possible to find out applicable software metrics that reflect the design quality of a software system.

Therefore this research work emphasis on the software quality metrics which are categorized into complexity and security part which are obtained via analyzing .class file which is obtained from source code input and the document specification. The proposed work is fully automated eliminating the manual effort required from the developer and analyzer, further because of the elimination of manual work these system is effective, efficient for the reengineering of the software which already in existence with effective utilization of the key resources . This framework applies metrics at both the levels design as well as code level by considering coupling of classes and objects. Also it generates spider graph to give the clear idea about the class coupling. It can be considered as an opportunity to identify high coupled class which may be vulnerable to attack in future. As it gives solution at design level which is advantage in terms of cost required to fix these flaws in later stage. Again this framework uses genetic algorithm for alternate designs. Benefit to use this algorithm is it is easy to understand and multi object optimizer.This Approach brings us to a very important outcome which allows the business people work independently of IT to analyze, design and build up their system.

REFERENCES

1. Ali Bahrami, Chapter 6, Object Oriented Analysis Process, in Object Oriented System Development.
2. H. M. Harmain and R. Gaizauskas, CM-Builder: An Automated NL Based CASE tool, in IEEE International Conference on automated software engineering (2000)
3. Overmyer, S. P., Benoit, L. and Owen R., Conceptual modeling through linguistic analysis using LIDA. International Conference of Software Engineering (ICSE), (2001)
4. G.S. Anandha Mala, J. Jayaradika, and G. V. Uma, Restructuring Natrual Language Text to Elicit Software Requirements, in proceeding of the International Conference on Cognition and Recognition (2006)
5. A Visual Analysis and Design Tool for Planning Software Reengineering", by Martin Beck, Jonas Trumper, Jurgen Dollner
6. Security metrics for object-oriented class designs", Alshammari, Bandar and Fidge, Colin J. and Corney, Diane
7. WordNet 2.1, last updatehttp://wordnet.princeton.edu/wordnet/, 27th October, 2010
8. "BCEL API documentation" "bcel.sourceforge.net/docs/index.html"
9. Lionel C. Briand Jie Feng Yvan Labiche," Using Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders" SEKE '02, July 15-19, 2002, Ischia, Italy. ACM 1-58113-556-4/02/0700.
10. Chidamber and C. Kemerer, "A metrics suite for object oriented design," IEEE Transactions on Software Engineering, vol. 20, pp. 476–493,

AUTHORS PROFILE

Devendrasingh Thakore, is pursuing Ph.D form JJTU, India. He obtained M.E. (Comp.) degree from the BU, Pune in 2004, M.B.A. (Marketing.) from MITSOM, Pune and B.E. (Comp Engg) from Walchand College of Engg, Sangli [M.S.] in the year 1996 and 1990 respectively. His areas of interest are Computer Network, Software Engineering and Database System. He has seventeen years experience in teaching and research. He has published more than twenty research papers in journals and conferences. He has also guided ten postgraduate students.

Dr. Akhilesh R. Upadhyay, obtained Ph.D. degree from the Swami Ramanand Teerth Marathwada University, Nanded in 2009, M.E. (Hons.) and B.E. (Hons.) in Electronics Engineering from S.G.G.S. Institute of Engineering & Technology, Nanded [M.S.] in year 2004 and 1996 respectively. He is currently working as Vice Principal and Head of Electronics and Communication Engineering Department at Sagar Institute of Research and Technology, Bhopal, India. He has more than 12 years teaching and 3 years of industry experience. He is Associate Editor of Journal of Engineering, Management & Pharmaceutical Sciences, Ex-Editor of International Journal of Computing Science and Communication Technologies and member of editorial boards/review committee of various reputed journals and International conferences. He has more than 50 research publications in various international/national journals and conferences; he also authored more than 16 text/reference books on electronics devices, instrumentation and power electronics. He is recognized Ph.D. Supervisor for various Universities in India and presently guiding 11 Ph.D. scholars.