A Framework 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. Different design metrics are available, based on their results design artifacts can be modified. Modifying or making changes in artifacts is not an easy task as 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. To analyze, extract and transform the hidden facts in natural language to some formal model has many challenges and obstacles. To overcome some of these obstacles in software analysis there should be some mean or a technique which aims to generate software artifacts to build the formal models such as UML class diagrams. Initially, the proposed technique converts the NL business requirements into a formal intermediate representation to increase the accuracy of the generated artifacts and models. Next, it focuses on identifying the various software artifacts to generate the analysis phase models. Finally it provides output in the format understood by model visualizing tool. The re-engineering process to find out design level artifacts and model information about the previous version of software system from available source code with easy layout is a very difficult task. Performing this task manually has many problems as the ability of human brains to deal with the complexity and security of large software systems is limited. To overcome this difficulty 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— ***actor, OOA, POS Tagging, quality metrics, software quality, UML, Use case, , XMI.***

I. INTRODUCTION

The main purpose of Object Oriented Analysis (OOA) is to capture a Complete, Definite and Consistent picture of the requirement of system and what system must do to satisfy the user requirement and needs. **[1]** This is accomplished by constructing several models of system. In this process user's needs are transformed in to set of problem statement and requirements specification called as Software Requirement Specification (SRS) in natural language (NL).

After that this natural language (Such as English) SRS are translated to the formal specifications such as UML models. This translation consists of generation of structural model of the system such as identifying the actors and related use cases, identify classes, their attributes, methods, and relationships among them.

However requirement (SRS) described in NL can often ambiguous, incomplete, and in consistent. In addition the interpretation and understanding of anything described in natural language has potential of being influenced by geographical, psychological and sociological factors. It is usually job of requirement analyst to detect and fix potential ambiguities, inconsistencies and incompleteness in SRS. But human reviewers can overlook defects in NL requirements which can lead to multiple interpretations and difficulties in recovering implicit requirements if analysts do not have enough knowledge. Also error caused during this stage of software development can be quite expensive to fix later on. Thus analyzing requirements and generating the software artifacts to build analysis model are bulky and complicated task which need automated support. Second thing if software product or system and their design and code grow larger and become more complex, then their quality level needs to remain high. Assessing the software product for quality is very complicated task. If source code of software is less then it is relatively less complex but source code of software is large then it is very complicated by manually and time and budget consuming due to ability of human being to deals with the complexity of large software system is limited. Different analyses and estimation of software quality metrics support this process. Thus recovering this type of quality metrics is the first step towards reengineering a software system. Software quality metrics plays the vital role in the process of assessing software as in qualitative and quantitative terms. Basically, software quality metrics determines specific but some important properties, attributes or characteristics of software in terms of numbers, values or some symbols. This type of assessment should be occurred according the some well-defined measurement rules. Software quality metrics are not only the static measurement state of project but also it will help in assessing the behavior, size, quality and complexity etc, of software. Evaluation of software quality metrics is used to predict the fault-prone area and components of software in early stage of reengineering process of existing software as quality indicators.

Manuscript published on 30 April 2013.
***Correspondence Author(s)**

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.

These software quality metrics helps to identify the problem from software in early stages of the reengineering of existing software.

Also many different metrics have been developed for software quality attributes of object-oriented designs such as performance, reusability, and reliability. However, metrics which measure the quality attribute of information security have received little attention as security is non-functional quality attribute. Moreover, existing security metrics measures the system at high level i.e. the whole system's level or at a low level i.e. the program code's level. These approaches are tough and costly to determine and fix weaknesses caused by software design errors.Final thing is that there should be some mean or a technique which aims at to generate software artifacts from natural language SRS. Initially such technique should convert the NL requirements in to some formal intermediate representation to garneted increase in accuracy of generated artifacts and models. Then it focuses on identifying the various software artifacts and models. Also it should find out model for the system from source code in reverse direction which determines various software quality metrics specification. It should determine the different software quality metrics, design view and attributes of objects from source code. Hence these metrics measures the complexity, effectiveness, efficiency of software, and extracted design view represents the relationship between different entities of software. These metrics should be save cost as well as time of software analyst, developer, tester etc., while reengineering the existing software with less effort.

After finding the design models in forward and reengineering way in proposed work design metrics have been developed which measures security at design level. This system then applies these design level metrics after applying Genetic Algorithms (GA) and gives secure design. The advantage of this technique is the cost and efforts needed to solve the problems after implementation get reduced as it discovers errors at early stage and after implementing that secure design it can be validated by different code level metrics.

With consideration of importance of software quality to reduce the total cost of development and time to market proposed research work spotlighting on the analyzing the object oriented systems in forward or reengineering whatever may be the case for better quality assurance of final product according to users need.

II. 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.

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].** This tool identifies model elements through assisted text analysis and validates by refining the text descriptions of the developing model. LIDA needs extensive user interaction while generating models because it identifies only a list of candidate nouns, verbs and adjectives, which need to be categorized into classes, attributes or operations based on user's domain knowledge.

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.

III. 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.

There are different tools to measure the quality of software at design level as well as code level. Each tool includes different set of metrics to measure different quality attributes. When user uses any tool at design level to validate the design with the help of included set of metrics then there is need to assess the implemented code with same set of metrics. Proposed tool facilitate user to measure security by applying tool at design level as well as code with same set of metrics. Proposed system is involves three modules, which works in manner as below.

Model I: - Forward Engineering Case

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. To do so java tokenizes class is used. After tokenizing each token is stored in separate array list.
2. Tagging: This processed text is further given as input to Part Of Speech (POS) tagger to identify the basic POS tags. To do so Standard POS tagger is used which identifies the 44 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. In this type of analysis WordNet is used to convert the plural into singular form also suffixes attached to verb phrases such as “ed” are also removed **[7].**
4. Parse Tree Generation: Stanford Parser is used to generate parse tree from pos tagged output for each requirements. This phase is very useful to find out artifacts such as actors, use cases to model the use case diagram.
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. Output of this phase is stored in an array list.
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. all SBVR noun concepts and object type are tends to be actor for use case model and classes for class model. All verb concepts associated to noun concepts are tend to be use cases of that actor for use case model and methods for the class model. Association between actor and use cases are identified with the help of parse tree generated
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 identify and extract 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. 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. Identification of such candidate methods are used into evaluation and extraction of software quality metrics. Structure of MethodVisitor is as follows MethodVisitor(MethodGen m, ClassVisitor c)
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. Identification of such candidate methods are used into evaluation and extraction of software quality metrics.
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: - 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.

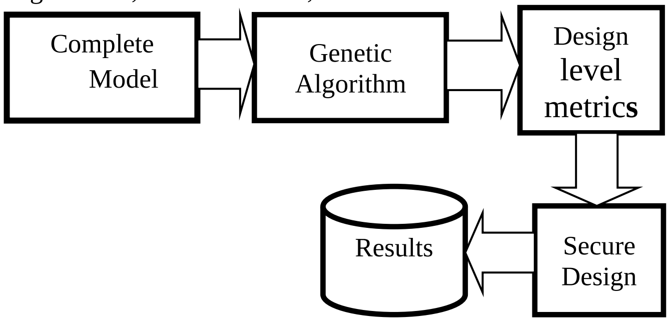

Fig 2: Application of GA and Generation of Secure Designs

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.

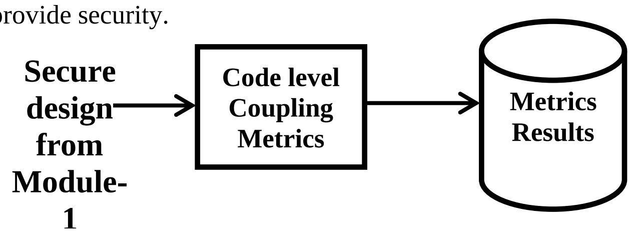

Fig 1: code evaluation by applying coupling metrics

- 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.

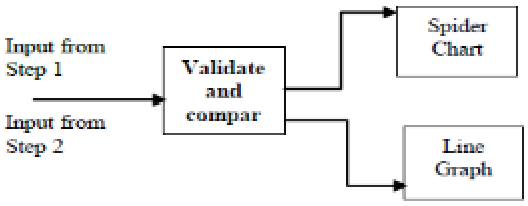

Fig 3: Validation of Metrics

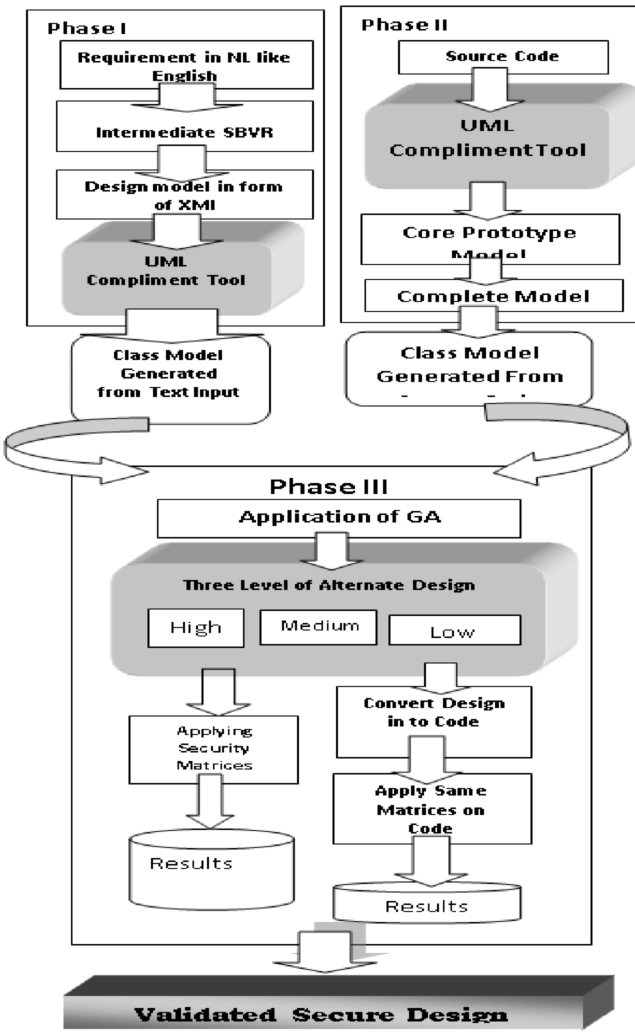
Phase I
Requirement in NL like English
Intermediate SBVR
Design model in form of XMI
UML Compliment Tool
Phase II
Source Code
UML Compliment Tool
Core Prototype Model
Complete Model
Class Model Generated from Text Input
Class Model Generated From
Phase III
Application of GA
Three Level of Alternate Design
High
Medium
Low
Applying Security Matrices
Results
Convert Design in to Code
Apply Same Matrices on Code
Results
Validated Secure Design

Figure 4 Combine Architectural Diagram of Proposed Environment

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,

AUTHOR 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.