Design Generic Architecture for software Engineering "Extractor" Tool

Rashmi Yadav, Abhay Kothari, Ravindra Patel

Abstract— Software architectures capture the most significant properties and design constraints of software systems. In our research work we intended to propose architecture for a reengineering tool. So, this paper will present a survey of techniques that have been proposed a comprehensive technique for design reengineering tool, here emphasize only how basic steps and architecture (extractor) for comprehensive reengineering tool on the bases of previous study, we analyze of different reengineering tool (Dali, GUPRO, DEFCTO, IMAGIX-4D, COLUMBS) and their comparative strength, weakness and suitable environment and architecture thereafter we proposed our holistic archite

Index Terms— Minimum 7 keywords are mandatory, Keywords should closely reflect the topic and should optimally characterize the paper. Use about four key words or phrases in alphabetical order, separated by commas.

1. INTRODUCTION

Propose objective to design the architecture of reengineering tool. We studied- the available reengineering tools to analyzes the pros and corns and develop requirement set classify into functional and nonfunctional requirements. Prepare a functional requirement set for proposed reengineering tool. Analyze the existing tools find their merits and demerits. The functional requirements are extractor, repository, analyzer, visualize. Nonfunctional Requirements[a] are related with the quality Interoperability, Traceability, Compatibility, Scalability, reusability, Interconnectivity, Reliability, Modifiability, Efficiency, and Understandability. We proposed our research objective: To provide a graphical representation for architectural models using Object-Oriented models. And we try to offer reverse engineering facilities, producing Object-Oriented models out of source code**.** To be capable of handling incomplete and inconsistent information .To analyzes models for completeness and consistency. Elaboration and more strict representation of the presented feature models. Scaling the process for large systems by decision support by introducing metrics. Dealing with the cases of exiguous domain knowledge. Recovery of the execution view of the architecture. Development of a mechanism for the maintenance of the collected information during the architecture recovery process. In this paper, we will discuss reengineering Extractor Architecture and basic rules.

2. Literature Survey and Review

According to kinnle [1], extractors extract all interested fact from artifacts and srored in the repository. There are various artifacts software in binary form, database schemas and data, user interfaces, reports, documentation, source code .Merits: In this paper author are choose the source code as an artifact because it is most important and the up-to-date artifacts in compare to other artifacts According to H.J.S. Basten [6], the evaluation of extractor based on the regular expression.the limitation of regular is that are language dependent, it cannot deal with the nested language. It also needs more maintenance. It is cumbersome when more complex context information example scope information, variable qualification. Another extractor is Compiler instrumentation; it removes the drawback of regular expression by supporting a number of languages. The extracted facts in the form of tuples. Grammar based approach is more general it is a mechanism the grammar of a language of interest with fact extraction directives and to automatically generate fact extractor. Its fast extraction can be seen as a very light-weight attribute grammar system that only uses synthesized attributes. In attributing grammar systems the further processing of facts is done using attribute equations that define the values of synthesized and inherited attributes. Basic facts can be described by synthesized attributes and are propagated through the syntax tree using inherited attributes. Analysis results are ultimately obtained as synthesized attributes of the root of the syntax tree. In our case, the further processing of elementary facts is done by using relational techniques. Queries and Relations the query imposes on the syntax tree. The problem with this method is less expressiveness of SQL, it is very slow. Rigi it applies the relational algebra is used in GROK in relation manipulation Language and relational partition algebra to represent basic facts about software system.

According to Rick Kazman [2], there is no single tool are sufficient for the architectural extraction. Architectural construction there is a variance in language, differenr architectural style, implementation and convention. For this Dali proposed an open, lightweight workbench that aids an analyst in extracting, manipulating, and interpreting architectural information. Discover the relationship between "as-implemented" and "as designed "architectures, static extraction made by two types. (i) Based on parsing (ii) Based

on Lexical technique. Lexical techniques are more versatile & lightweight then parse based techniques but Lexical techniques typically achieve lower accuracy. So there is no one tool will successfully extract the complete source code. If choose one method the extractor may be imperfect so the problem if one function call or relation missed that affects the architectural model. Author presents the solution of this problem the composition of multiple extractions Techniques will alleviate these problems by providing a concrete model of higher accuracy than any individual technique. If an extractor missed one function call disturb the high-level model, however, this is a dangerous assumption as it will not be a single function call that is missing, but more likely a whole class of related elements or relations. This deficiency will likely affect the architectural model. We analyze this model and found, Merits: (i) openness New elements must be easy to integrate into the workbench, and such integration should not unnecessarily impact other elements of the workbench dependencies are lightweight. (ii) Dali allows an analyst to interact with the recovered information by assessing the results of the reconstruction effort to see whether composite elements demonstrate functional consistency, and by seeing places where the as-built architecture differs from the as-designed architecture. No extraction technique is useful or complete without user interaction. Demerits: It is not fully automated human involvement necessary for the good architecture.

Johannes Martin [3] provides tool writers with the ability to query the compilers internal data structures for information on the programs being compiled. It gives guideline how these features can be used to write data extractor for supplying data to common reverse engineering tools. C++ Programs are combination of many files. Header files that contain the declaration of data type's functions and global variables and implementation files that contain the definitions. These files depend on each other and implementation file that defines the instantiations of a variable of a certain data type depends on the header file with the declaration of that data type. In a traditional programming environment programmer has to keep track of the dependencies of these files usually using tools such as make utility. A traditional written parser has to process the subject program source code sequentially breaking it up to into lexigraphic token and building Abstract Syntax trees (AST). Once this is done the AST can be traversed for the obtain information from the code. After analyzing their research work we found, Merits: (i) Reducing the complexity of data extractors. (ii) Less time needed to develop these as compared to using traditional approaches and programming tools.Its limitation is based on Structured Query language that is limited in nature. Jürgen Ebert Bernt et al., proposed GUPRO (Generic Understanding of Programs) [4] is an integrated workbench to support program understanding of heterogeneous software systems on arbitrary levels of granularity GUPRO is based on graph. Source code is extracted into a graph repository the abstraction is done by graph queries and graph algorithms. This can be viewed by an integrated querying and browsing facilities. For C-like languages. this paper summarizes the work done on GUPRO during the last seven years The objective of GUPRO is to provide an integrated reverse engineering workbench supporting multiple program analysis techniques. After analyzing their research work we found, Merits: GUPRO uses a schema-independent querying mechanism. Demerits: Due to large software system all facts are source cannot fill at once due to Limited repository size. Fact extractors for multi-languages systems follow a four step parsing approach. Step 1checks if the document is already represented in the repository in a former version. If so, its facts are removed. The document itself is then parsed in a second step, in third step the extracted facts are integrated into the existing repository, in the fourth step ensures further integrity constraints.

Rudolf Ferenc et al. [5] a Framework "Columbus" Reverse engineering tool" extractor is one of the components of this tool like other re-engineering tool. The extraction process is based on a Columbus project. A project stores the input files (and their settings: precompiled header, preprocessing, output directories, etc.) displayed in a tree-view, which represents a real software system. The project can simultaneously contain source files of different programming languages. The process is very similar to a compiler system. The first stage of the extraction process is data extraction. Columbus takes the input files one by one and passes them to the appropriate extractor, which then creates the corresponding internal representation file. Columbus may contribute as a flexible, easily extensible tool architecture, a data exchange model (C/C++ schema) and as a source code analysis process. After analyzing their research work we found, Merits: (i) Architecture of Columbus easily extendable. (ii)It provides re-usability. DEFCTO for fact extraction it is based on language parametric & fact parametric. It amounts to annotating the context-free grammar of a language of interest with fact annotation it describes how to extract the elementary facts from the language element .Then it uses the Relational techniques to further enrich them and to perform the actual software analysis. After analyzing their research work we found, Merits: (i) DEFACTO method is considerably smaller than other competing fact extractor method. (ii) Arbitrary factual annotations can be added to the grammar; it is independent from any preconceived analysis model and is fully general. The method is succinct and its notational efficiency has been demonstrated by comparison with other methods. Demerits: This technique does not rely on a specified grammar formalism or parser. Also, for the processing of the extracted facts, other methods could be used as well, ranging from Prolog to Java.

Ghulam Rasool at el. [7] ,presents a simple and lightweight pattern extraction technique to extract different artifacts from legacy systems using regular expression pattern specifications with multiple language support and own custom-built tool DRT to recover artifacts from existing system at different levels of abstractions. Sanatanu Paul et al.[13], presents a framework in which pattern language is used to

specify interesting code features. The pattern language is derived by extending the source the source programming language with pattern matching symbol. It describes SCRUPLE a finite state machine based source code search tool that efficiently implements this framework. The SCRUPLE Run Time system searches the source code for matches. The program source code is transformed by a source parser into a data structure called the AST (Attributed Syntax Tree) which is based on the attributed dependency graph model. The pattern or query specified by the user is transformed by a parser pattern into automation called code pattern automation (CPA), CPA are non determents finite state automata. Alexandru Tele el at. [8], presented tool SolidFX an integrated reverse-engineering environment (IRE) for C and C++. SolidFX was specifically designed to support code parsing, fact extraction, metric computation, and interactive visual analysis of the results in much the same way IDEs and design tools offer for the forward Engineering pipeline. In the design of SolidFX, adapted and extended several existing code analysis and data visualization techniques to render them scalable for handling code bases of millions of lines. In this paper, the author gave the detail several design decisions taken to construct SolidFX. It also illustrates the application of our tool and our lessons learnt in using it in several types of analyses of real-world industrial code bases, including maintainability and modularity assessments, detection of coding patterns, and complexity analyses. It is proposed architecture based on Elsa but remove the limitation is it require the preprocessed input no preprocessor facts are extracted, it can't parse the incorrect code. The output cannot be filtered or queried for the specific facts. After analyzing their research work we found, Merits: (i) It doesn't require preprocessed input. (ii) Good in error recovery. (iii) Output can be filtered or queried for specific facts.Demerits:The SolidFX need for Refined static information can be extracted from the basic facts, such as control flow and data dependency graphs, leading to more complex and useful s safety anal. According to storey Rigi [9]give two types of representation multiple windows, ShriMP (Simple hierarchical Multiperspective) Rigi gives high level abstraction .Its three main components parsing subsystem-Parsing subsystem contains C,C++,COBOL languages repository - Repository stores the result of parsing step. Interactive graph editor-Graph editor called "rigiedit" gives browsing, editing, manipulating, exploring and managing capabilities. Merits- Extensibility, Customization, representation, it provides metrics for cohesion and coupling, adapt different programming language, easy to use interface. Demerits: Not user friendly. No integrated source code editor .The parser only supports parsing functions and structure data types so it only generates the structure of the software system in functional views (call graph) and cannot generate other views such as class diagrams and control flow. Provides limited amount of software metrics. Unavailability of dynamic views. Reveal [10] describe, Rational Rose and ArgoUml does not define binary class relationship they distinguish graphically association aggregation and composition but produce incomplete relationship. In Rose aggregation is not detected .Reveal can be useful for reverse engineering class diagrams however it does not provide round-trip engineering. Reveal is a more precise tool. Other tools do not use a full parse to guide the reverse engineering process but rather use a fuzzy parse that scans the code for keywords. It provides complete modeling of the application,it is closer to UML standards. This is a commercial tool released by Imagix Corporation [11].The architecture contains three main layers that is View, Exploration engine, Database (Imagix4D).This is a program understanding tool for C and C++ programs. It gives information in a 3D-graphical format. Merits: It is having a good user interface and it is easy to use. It automatically generates documentation from source-code. Demerits of this tool are: (i) This tool cannot be extended. (ii) Its can't generate the graphical views that can be useful in printed form.

Stephane Ducasse et al. [12] ,presents a tool it is a tool environment to reverse engineer or re-engineer a system. It provides an engineering environment that allows tools to collaborate (Moose). Moose give the requirement set for re-engineering tool that is supported for re-engineering tasks, Extensible, Exploratory, and Scalable. Moose uses layered architecture. Its component is an Import/Export Framework, Repository and Model Management, Services, Querying and Navigation, Metrics and other Analysis support, Grouping, Refactoring. Merits: It can store, query and navigate information. Its extensibility propernhnty helps to make it a foundation for another tool. It provides a complete description of the Meta model elements In terms of objects that are easily parameterized, extended or manipulated.

3. Analysis of reengineering tool:

I. DALI TOOL:

S.N.	Merits	Demerits
1	New elements must be easy to integrate into the workbench (openess), and such integration should not unnecessarily impact other elements of the workbench (dependencies are light-weight).	It is not fully automated human involvement necessary for the good information.
2	It allows an analyst to interact with the recovered information by assessing the results of the reconstruction effort to see whether composite elements demonstrate functional consistency, and by seeing places	It is highly complex.

	where the as built architecture differs from the as-designed architecture.	
3	No extraction technique is useful or complete without user interaction.	

II. GUPRO TOOL:

S.no.	Merits	Demerits
1	It uses a schema independent querying mechanism	Due to large software system all facts are source cannot fill at once due to Limited repository size, fact extractors for multi- languages systems follow a four step parsing approach.

III. DEFCTO TOOL

S.no.	Merits	Demerits
1	Arbitrary factual annotation can be added to the grammer; it is independent from any preconceived analysis model and is fully general.	This technique does not rely on a specified grammer formalism or parser.
2	The method is succinct and its notational efficiency has been demonstrated by comparison with other method.	

IV. IMAGIX-4D

S.no.	Merits	Demerits
1	It helps software developers comprehend complex or legacy C, C++ and Java source code.	The hand designed class and function diagrams sometimes does not get match with the tool designed diagrams.
2	By using Imagix 4D to reverse engineer and analyze our code, we are able to speed your development, enhancement, reuse, and testing.	The parser lacks of important information about method/function calls which is due to inability of interpreting template parameters.
3	It eliminates bugs due to faulty understanding.	It is unable to resolve the function to which the invocation resolves during compilation time.
4	It enables us to rapidly check or systematically study your software on any level -- from its high level architecture to the details of its build, class and functional dependencies.	Imagix 4D requires many hours of analysis for larger code-bases
5	We can visually explore a wide range of aspects about your software - control structures, data usage, and inheritance. All based on its precise static analysis of your source code.	Imagix 4D does not produce a full executable slice, since it does not perform analysis of relevant conditions for the identified statements.
6	Using this tool we are able to find and focus on the relevant portions of your source code through its querying capabilities.	
7	Automated analysis, database lookups, and graphical querying all sift through the mountains of data inherent in our source code so we can examine the structural and dependency info we are interested	

	in. Quickly and accurately.	

4. Architecture for reengineering tool:

for any software engineering phase we follows following steps requirement, analysis ,design, and maintenance but here we try to design reengineering phase fig.1 shows overall structure for reengineering process.

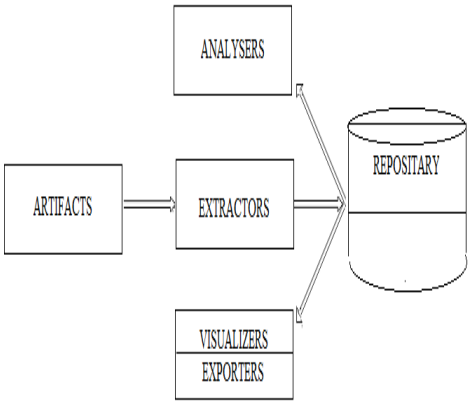

Fig.1. Basic Reengineering Process

5. Architecture for Reengineering Extractor

On the basis of above analysis ,in this research work we proposed a architecture ,for reengineering tool and here we try to describe their functionality .In fig 2 present basic object-oriented diagram for extractor ,diagram consists of eight parts. The Parser & Scanner are abstract classes. The parser request taken from its scanner and, it has dependency on tokens. The Scanner owns the token; it owns the source by the source field. Scanner is one of basic part of our basic reengineering tool; it is dependent on source code, Parser is dependent on scanner. Symbol table is dependent on passer. Scanner is dependent on token. Eof token is a subclass for token subclass. Token is dependent on source code. This diagram show basic object-oriented properties .This object-oriented properties help us realizing overall architecture of any system.

Class Level Diagram for Reengineering Extractor:
In fig 3 further classify our architecture in class level architecture and identify function and its variables which help us for realizing any system at low level design also.

Parser Class:
Parser controls the transaction process it parses the source program and so the passer class has an abstract parser method. It wills repeatedly the entire scanner for next token.

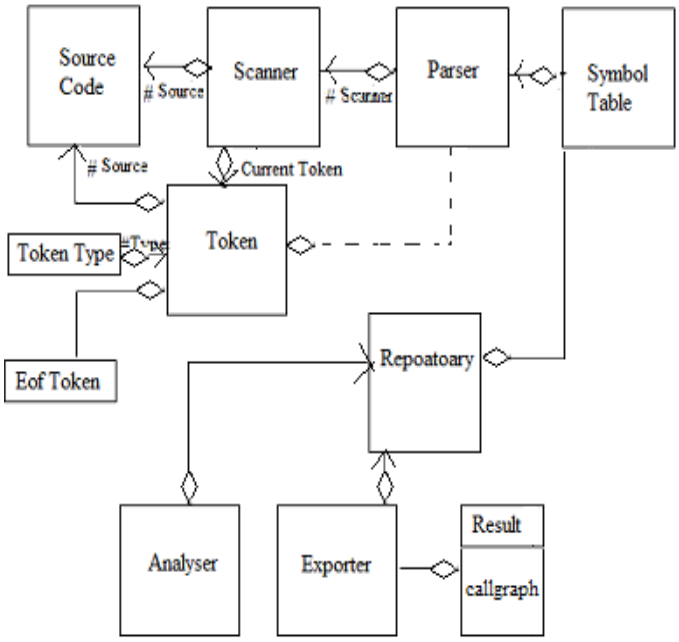

Fig.2. Basic Extractor Diagram

Current Token -: The parser current token.
Next token ( ) - convenience methods in tuner call the scanner's current token & next token ( ) methods.
Geterrer count ( ) - it will return the number of syntax trees.

Scanner Class:
The scanner controls taken extracted from source program.
extract token ( ) - it will read character from source in order to construct the tokens of current char ( ) call the corresponding methods of source class .

Token class:
In Token Class store useful information about a token including type, string value and location (line numbers & position) in the source program. It also has current char ( ) next char ( ) methods that will in turn call the current char ( ) & next char ( ) method of source class. Token type is language specific and Token type interface serves as place holder. Eof token subclass represent the end of source file, using token subclasses will keep the scanner code modular. Dashed arrow is a reference that exists only during a method call. A solid arrow with a closed arrow head point from substitutes to super class a class diagram can include sections for the field and methods field name that are arrow labels do not appear again inside the field section.

+ public ,- private # method – package

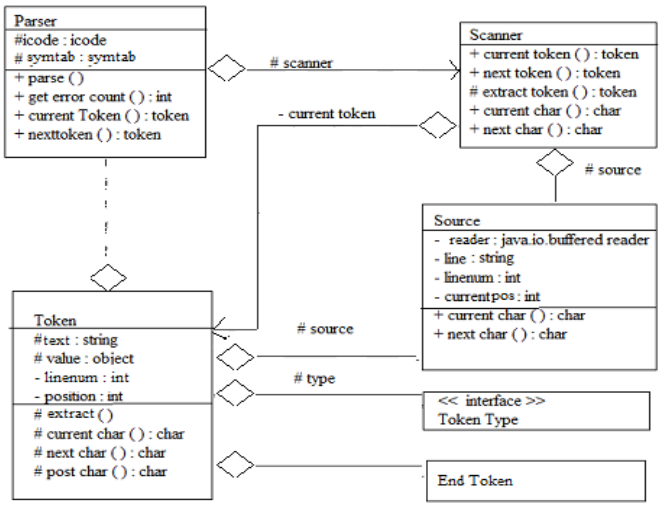

Fig.3. Extended Architecture of Extractors

6. Conclusion

In this research work we try to provide a architecture reengineering extractor which is holistic in manner, pursue the standard of object-oriented notation and realization of architecture on the basis of object oriented properties of software, which help short all limitation of existing tool which mention above table and next section of this research work we developed a software tool on the bases of above limitation. The architecture of Extractor for reengineering tool definitely helps software industry for convert legacy system to newer system and reduces overall reengineering time.

Reference

1. Holger Michael Kienle" Building Reverse Engineering Tools with Software Components, A Dissertation of DOCTOR OF PHILOSOPHY 1996 pages 1-360.

2. Rick Kazman S. Jeromy Carriere October 1997 Playing Detective: Reconstructing Software Architecture from Available Evidence Technical Report CMU/SEI-97-TR-010 ESC-TR-97-010 DTIC QUALITY.

3. Johannes Martin, "Leveraging IBM Visual Age for C++ for Reverse engineering Tasks" Conference of the Centre for Advanced Studies on Collaborative Research (CASCON' 99), pages 83–95, November 1999.

4. Jürgen Ebert Bernt, Kullbach Volker and Riediger Andreas, Winter *GUPRO* Generic Understanding of Programs June 2002.

5. Rudolf Ferenc, Arpad Beszedes, Mikko Tarkiainen, and Tibor Gyimothy. Columbus—reverse engineering tool a schema for C++. *18th IEEE International Conference on Software Maintenance (ICSM'02)*, pages 172–181, October 2002.

6. H.J.S. Basten and P. KLINT DEFACTO, "Language-Parametric fact Extraction from Source Code *SLE*, volume 5452 of Lecture Notes in Computer Science, page 265-284. Springer, (2008)

7. Ghulam Rasool, and Ilka Philippow ,Recovering Artifacts from Legacy Systems using Pattern Matching World Academy of Science, Engineering and Technology 22 2008.

8. Alexandru Tele and Heorhiy Byelas, A Framework for Reverse Engineering Large C++ Code Bases" Elsevier Electronic Notes in Theoretical Computer Science 233 (2009) 143–159.

9. Tung doan, An evaluation of four reverse engineering tools for C++applications, October 2008 University of Tampere, Department of computer sciences M. Sc. Thesis.

10. Sarah Matzko and Jasmes F. Power, Reveal: A tool to Reverse engineer class diagrams, Confrence in Research and Practice in Information Technology, Vol.10.

11. http://www.imagix.com

12. Stéphane Ducasse, Michele Lanza and Sander Tichelaar "The Moose Reengineering Environment"

13. Sanatnu Paul, Atul Prakash "A framework for source code search using program patterns" IEEE Transnaction on software Engineering Vol 20 No. 6 june 1994.

14. Ilian Pashov, Matthias Riebisch, "Using Feature Modeling for Program Comprehension and Software Architecture Recovery "Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04),0-7695-2125-8/04, 2004