Static Analyzer in Java Code as a Prototype for Reverse Engineering Technique

Dahlia Binti Din
Centre for Advanced Software Engineering (CASE)
Fakulti Sains Komputer dan Sistem Maklumat
Universiti Teknologi Malaysia, City Campus
54100 Kuala Lumpur
ailhaddin@yahoo.com

ABSTRACT

Software maintenance is most important process in a System Development of Life Cycle (SDLC). Although the system in the deploy stage, maintenance still need to be done time to time for software improvement. As we know, software maintenance is complex and costly because of the difficulties that exist in software code. To easy understand the code structure, static analysis technique will used to capture the artifacts and visualize the dependencies between the artifacts. This paper will discuss about static analyzer prototype, which can assist the maintainer to do the software maintenance in their Java Software.

KEYWORD

Static Analysis, Code Parser, Reverse Engineering Technique

1. Introduction

This research discusses about static reverse engineering technique for modeling the source code written in Java. This technique is a static analysis that produces software artifacts and their relation. Static analysis will combine with Object-oriented methodologies to produce new reverse engineering technique tools. Object-oriented methodologies provide software developers with a powerful means for designing and implementing application. Programmers have adopted the object-oriented approach to take advantage of better data abstraction, improved encapsulation and modularity, polymorphism, and code and design reuse. These characteristics can potentially benefit both software development and software maintenance [1].

In the object-oriented level, source code understanding involve from analysis of the relationship or dependencies between artifacts in the source code. This artifacts obtained from Java source code that already be parsed by Regular Expression from Java library itself. In this case, one repository will produce to keep artifacts such as classes, methods, variables and other object-oriented component. To support the dependencies between artifacts, visualization is good supports tools to help visualize the dependencies of source code.

This paper intends to produce the prototype of Static Analysis for Object-oriented. In this paper, the Java language will use as a model for source code that will apply in the new prototype. The static analyzer also have parser feature to create the artifacts from source code and represent the dependency of artifacts into a simple visualization.

2. Background

Experimental research of this work is comes primarily from the java source code based maintenance of object-oriented software. Other related, but less significant areas such as design recovery and architectural design.

OOAD (Object-Oriented Analysis and Design) is a methodology that design, visualize and document the artifacts in object-oriented software. These methodologies provide the notations and guidance to model both the static structure of the program and the dynamic behavior of the object. In the UML (Unified Modeling Language) [2] the corresponding diagram are called sequence diagram and statechart diagram, respectively. In this research, UML will be use as a guideline for define every object interaction in java source code.
In the OMT (Object Modeling Technique) [3] the diagram call event trace diagram and state diagram. A scenario diagram shows and object interaction arranged in time sequence during a particular execution of the system. Participating objects or classes drew as vertical lines an interaction between them with horizontal arcs.

Shimba [4], a prototype reverse engineering environment, has been built to support of java software. Shimba uses Rigi and SCED (Scenario Diagram) to analyze, visualize, and explore the static and dynamic aspects, respectively of the subject system. The static source code artifacts and relationship are extracted from Java Byte Code and viewed as directed graphs using the Rigi reverse engineering environment.

Sneed and Dombovari [5] were introducing an approach to model the requirement specification and the system implementation of a large, distributed C/C++ system. The approach combines forward and reverse engineering technique and support of software maintenance.

Columbus [6] is a reverse engineering framework application that provides an environment which enables the analyzing, internal representation, filtering and exporting of information extracted from C++ source files into various formats including XML. It supports project handling, data extraction, data representation and data storage. In addition, efficient filtering methods can be utilized to produce comprehensible representations of the extracted information.

Many research or work paper have been embarking on objects interaction in source code, but they work with different concept and perspectives. Some are researching on dynamic analysis level such as behavior of objects. Some are working on static analysis level with the software artifact result. While few researcher are working at both static and dynamic level to work out advance tools of reverse engineering technique.

3. Static Analysis

The predominant technique used for gathering data is static analysis by parsing a system’s source code to construct abstract syntax trees with a large number of fine-grained syntactic artifacts and dependencies. This type of data gathering is essentially the same as running the front end of a compiler. It requires constructing a scanner and using a valid grammar for the implementation language of the system.

In this paper, to implement static analysis, researcher will use static call graph analysis to gather the data from source code or java byte code to represent different kinds of relationship like:

i. Package to package
ii. Class to package
iii. Class to class
iv. Method to class
v. Method invokes method
vi. Method invokes attribute

Many researchers have spent an inordinate amount of time building parsers for various programming languages and dialects. However, mature technology already exists in the compiler arena that will parse source code, perform syntactical analysis, and produce cross-reference and other information that can be used by other tools, such as debuggers. By using the leverage of proven compiler-based technology for data gathering, users of reverse engineering tools will be assured of predictable results.

4. Proposed Static Analyzer Re-Engineering

In this research, researcher will produce a prototype for static analyzer reverse engineering to help other developer to do the software maintenance especially in Java Software.

The steps of static analysis are as follows:

i. Translate the java byte code to produce the java code using JODE or
ii. Parse the java code into database
iii. Define the artifacts in java code
iv. Define the relationship of every artifacts
v. Create simple visualization of object interaction

4.1. Data Analysis

The structured of Java has a one-to-one mapping concept used in design object model. The structural elements in use in an Objectory design object model are (and correspondent in the source code are) [7]:

For example at the class level, researcher can define the relationship between the classes when the data extracted from the source code. It’s mean the data analysis can determine the characteristic or data of the objects. Characteristics are formalized as attributes, and they represent the data of an object. Data analysis is the process of finding objects’ data then finding the real-world relationship between the objects. At the end of the phase, researcher delivers the written description of each object and relationship that exist in the java code.

5. Conclusion

At the end of the research, researcher will be present a prototype of static analyzer to handle static analysis of software system. Researcher will do the simple visualization to present the artifact and the relationship (dependencies) after data gathering to make the developer more understand the software flow. To make the research more valuable, the study should involve some literatures such as the call graph, static analysis techniques and data analysis.

Reference