International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2015): 6.391

Tool for Converting Source Code to UML Diagrams & Measuring Object Oriented Metrics in OO Java Software

Shubhangi Sakore[1], Ravina Kudale[2]

Department of Computer Engineering, Bharati Vidyapeeth"s College of Engineering for Women, Pune, India

Abstract: ***According to the Computer World, reverse engineering is commonly used as a way to make new compatible products that are cheaper than what is currently on the market. Software Engineering research and industry recognize the need for practical tools to support reverse engineering activities. The most commonly used standard today is Unified Modeling Language to depict the architecture and design of an application. Software couplings are based on object-oriented relationships between classes, specifically focusing on types of couplings that are not available until after the implementation is completed, and presents a static analysis tool that measures couplings among classes in Java packages. So, our project idea is to develop an application which would convert the given code into UML diagrams (Class Diagram and Package Diagram) and also measure Coupling in Java Object Oriented Software.***

Keywords*:* Software Engineering, Object oriented programming, quality analysis and evaluation, Coupling, Reverse Engineering.

1. Introduction

In the software engineering domain, UML has grown to a widely known and readily used graphical standard for representing various aspects of an object oriented software system. Reverse engineering is first step towards software Architecture recovery. Software reverse engineering tools help in software architecture recovery.

Since source code is in text form, it is complex and is hard for human to read or analyze, especially when the logic is complicated and involves a large number of classes."A picture is worth a thousand words", by visualizing source code with diagram, you can easily realize the classes involve as well as their relationship in run time. This is very beneficial for both analysis and communication.

A UML diagram describes the architecture of object oriented programs. Most of the object oriented systems do not have reliable system"s architecture or designed without software architecture design phase.

Software coupling has long been used to evaluate software. Specific ways to measure coupling have long been known for procedural software, but most measures for object-oriented software have been based on design artifacts such as class diagrams. System statically analyses source code to measure coupling in object-oriented software.

2. Literature Survey

A. Reverse Engineering

In the world of computing applications approximately 30-35% of the overall total lifecycle costs are devoted to helping the programmer understand the functionality of existing code. This is a necessary task, in order to correctly make required changes in response to new requirements, to resolve errors, or perform other changes. Reverse engineering, analyses a system's code, documentation and behavior to create system abstractions and design information. This information is then used to gain more knowledge about the design, the structure, system code, and functionality.

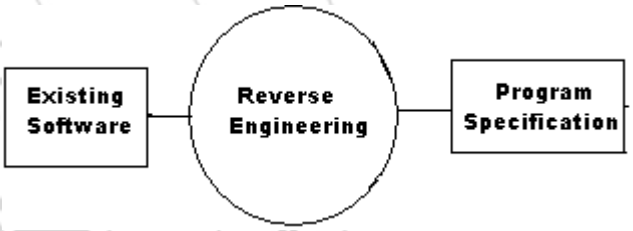


Figure 1

Reverse engineering, also called back engineering, is the processes of extracting knowledge or design information from anything man-made and re-producing it or reproducing anything based on the extracted information. It is a process to achieve system specification by thoroughly analyzing, understanding the existing system. This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.

An existing system is previously implemented design about which we know nothing. Designers then do reverse engineering by looking at the code and try to get the design. With design in hand, they try to conclude the specifications. Thus, we are going in reverse from code to system specification.

B. Object-Oriented Metrics

Object-oriented design and development is becoming very popular in today's software development environment. Object-oriented development requires not only a different approach to design and implementation but it requires a different approach to software metrics. Since object oriented technology uses objects and not algorithms as its fundamental building blocks, the approach to software metrics for object oriented programs must be different from

the standard metrics set. Metrics, such as Lines of code and Cyclomatic complexity, have become accepted as standard for traditional functional or procedural programs and were used to evaluate object-oriented environments at the beginning of the object oriented design revolution. However, raditional metrics for procedural approaches are not adequate for evaluating object oriented software, primarily because they are not designed to measure basic elements like classes, objects, polymorphism, and message-passing.

3. Coupling and Accessibility Matrics

A. Coupling in Object Oriented Software

Coupling is the principle of "separation of concerns". This means that one object doesn't directly change or modify the state or behavior of another object.

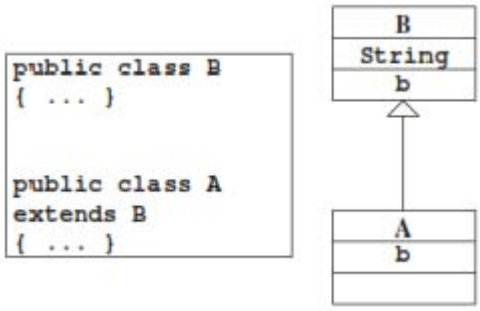
Coupling looks at the relationship between objects and how closely connected they are. A Relations Diagram is a great way to visualize the connections between objects. In such a diagram, boxes represent objects and arrows represent a connection between two objects where one object can directly affect another object.

a. Parameter Coupling:
In Java, parameter coupling occurs through only method and constructor calls. This research refines the generic definition of parameter coupling to be the occurrence of an invocation of a call to a method or constructor through an object or class. Java allows two explicit types of method calls, instance and static, and one implicit type, through a constructor. If a method in class A explicitly calls method m () in class B through an object instance (b.m()), this represents parameter coupling between A and B. An explicit static call occurs when class A calls a public static method m () in class B (that is, B.m()). An implicit constructor call is made when a variable of type B is defined and instantiated in class A, for example, B b = new B (). All three of these types are considered to be parameter coupling in this research, even if no actual parameter value is passed in and no value is returned.

The three types of parameter coupling are summarized as:
1. B b = new B (); // implicit, constructor
2. b.m(); // explicit, through an object reference
3. B.m(); // explicit, static

b. Inheritance coupling
Inheritance coupling refers to the coupling that is related to the inheritance between pairs of classes. The coupling takes place through attributes and methods that are inherited and used by a subclass but that are not re-defined. If a subclass does not actually use anything from its superclass, or if it re-defines everything it uses, this is not considered to be inheritance coupling.

Inheritance coupling occurs when one class is a subclass or descendant of another. The coupling is made through inherited but not re-defined data members of a superclass by its subclass.



Above figure shows a class B and its subclass A. B defines a data member b, and A inherits b from B. Accordingly, classes B and A are inheritance coupled through b.

c. Global Coupling
Global coupling is a kind of inter-class coupling that refers to the coupling that takes place through variables that are defined in one class and used in others. These variables will typically have public or protected/package access specifiers. Public variables represent a traditional, or true global coupling. If the variable is static, otherwise it is a global coupling with an object reference. All of these variations must be detected.

d. Data abstraction coupling
Data Abstraction Coupling (DAC), given by Li and Henry is defined as the number of abstract data types (ADTs) defined in a class. The number of variables having an ADT type may indicate the number of data structures dependent on the definitions of other classes. „T (a)“ is the type of attribute and „a“ is the attribute and C is the set of all classes in the system. This metric confines itself to class level, hence cannot be applied at subsystem level and thus is costly for large scale system.

e. Import coupling
Import coupling measures the degree to which an element has knowledge of, uses, or depends on other design elements. High import coupling can have the following effects:

Decreased maintainability: changes to the supplier may necessitate follow-up changes (ripple effects) to the client. The stability of the supplier is a factor to consider here. High coupling to elements that are not likely to change is less harmful than coupling to "hot spots".

Decreased understandability, increased fault-proneness: Elements with high import coupling operate in large context, developers need to know all the services the element relies on, and how to use them.

Decreased reusability: To reuse a class or package with high import coupling in a new context, all the required services must also be made available in the new context.

f. Export coupling
Export coupling measures the degree to which an element is used by, depended upon, by other design elements. High export coupling is often observed for general utility classes (e.g., for string handling or logging services) that are used

pervasively across all layers of the system. Thus, high export coupling is not necessarily indicative of bad design.

Again, an important issue to consider here is stability. High export coupling elements that are likely to change in the future can have a large impact on the system if the change affects the interface. Therefore, high export classes should be reviewed for anticipated changes, to ensure that these changes can implemented with minimal impact.

h. External Coupling

External coupling is defined as access to an external device by two or more classes. In other words, external coupling happens when two classes share something that is outside the application that owns the classes. External resources can include files on a hard disk, printers, or other shared devices. The challenge in designing an algorithm to analyse coupling is to find out the unique interfaces between these resources and the application. Specifically, different classes or applications may use the same resource, but refer to them with different names. Binding to a physical device may be done at the OS level, not the program. This is necessary for symbolically linked files and devices with multiple names. If there is no unique interface, then all interfaces must be enumerated.

B. Accessibility Metrics

Metrics under this category aim to measure the accessibility level of attributes and methods in a particular class from an access modifier perspective. Access modifiers are associated with each class, method, and attribute to control their accessibility. These modifiers include: public, protected, and private. Maruyama et al. have investigated how changes to access modifiers could change the security characteristics of a given program. Their work shows which refactoring rules could change a class‟s accessibility level and therefore changes its security level. However, this approach doesn‟t quantify the impact of these changes on the security level of a given program. Our accessibility metrics are similar to the one used by Bansiya to measure the encapsulation property of a class, called the Data Access Metric (DAM). DAM is measured as the ratio of the number of private (protected) attributes to the total number of attributes in a declared class Bansiya also included another metric to measure the accessibility of methods, called the Operation Access Metric (OAM). OAM is defined as the ratio of the number of public methods to the total number of methods in a class. Our security accessibility metrics statically measure the potential flow of information from an accessibility perspective for an individual object-oriented class. These metrics only consider attributes and methods declared as classified since they are the ones which need to be kept secret. We divide these metrics for individual classes into three kinds of accessibility: instance attributes; class attributes; and methods.

a. Classified Instance Data Accessibility (CIDA)

This metric measure the direct accessibility of classified instance attributes of a particular class. It helps to protect the classified internal representations of a class, i.e. instance attributes, from direct access. It is defined as “The ratio of the number of classified instance public attributes to the number of classified attributes in a class”. Therefore, it is calculated by dividing the number of public classified instance attributes in a class to its total number of classified attributes. This gives us the ratio of classified instance attributes which have direct access from outside the class. Higher values indicate higher accessibility to these classified attributes and hence a larger „attack surface“. This means a higher possibility for confidential data to be exposed to unauthorized parties. Aiming for lower values of this metric adheres to the security principle of reducing the attack surface

b. Classified Class Data Accessibility (CCDA)

This metric measure the direct accessibility of classified class attributes of a particular class. This metric aims to protect the classified internal representations of a class. i.e. class attributes, from direct access. It is defined as follows: “The ratio of the number of classified class public attributes to the number of classified attributes in a class”. This metric is calculated by dividing the number of public classified class attributes of a given class by its total number of classified attributes. The result shows the ratio of classified class attributes which are directly accessible from outside its class. Higher values mean that confidential data of that class has a higher chance of being exposed to unauthorized parties. This metric contributes towards measuring the attack surface size of a given program‟s classified class attributes. Thus, lower values of this metric enforce the security principle of reducing the attack surface.

c. Classified Operation Accessibility (COA)

This metric is the ratio of the accessibility of public classified methods of a particular class. We define it as: “The ratio of the number of classified public methods to the number of classified methods in a class”. It is calculated by dividing the number of classified methods which are declared as public in a given class by its total number of classified methods. This value also indicates the size of the attack surface of a given class. It aims to protect the internal operations of a class which interact with classified attributes from direct access. Lower values of this metric would reduce potential information flow of classified data which could be caused by calling public methods. This metric measures the potential attack surface size exposed by classified methods.

System Overview

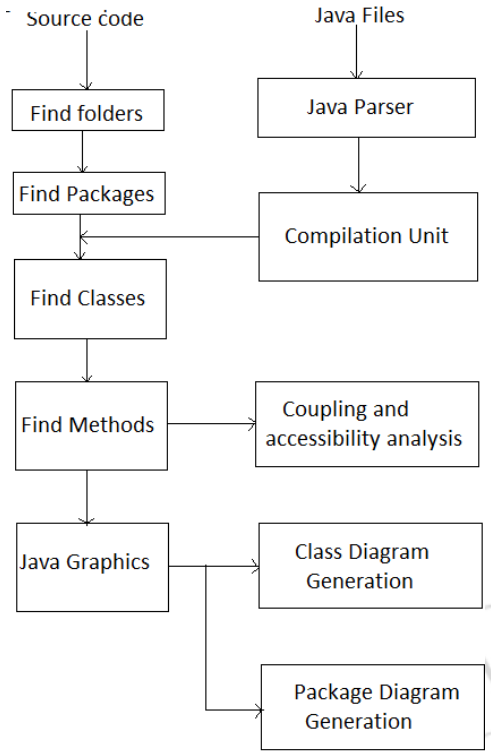


Figure: System Overview

In the proposed architecture, initially the java source code is entered by the user and this source code is then checked for errors by the compiler. If there are errors detected in the source code, then the system will exit.

If source code is compiled successfully, then module will thereby extract the details such as classes, functions, attributes, association rules, inheritance, packages etc. from the given java source code and store them in appropriate form in hash table.

Finally, data is interpreted by drawing a class and package diagram using java graphics.
This system also analyses class files for calculating different coupling and accessibility metrics.

Class diagram generation
The class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application.

1) An inheritance relationship exists within two classes of a given code when we encounter two keywords in the class definition:
   1. Extends 2. Implements
2) An association represents a family of links. Binary associations (with two ends) are normally represented as a line.
3) So our project includes a parser which consists of such regular expressions which identify these key words in the class definition and retrieves the needed data in hash table.
4) The data from hash table is used to provide class name and type, the table Variable provide the list and type of variables used within the class and Function provide the functions included in the class. The parser stores whether a instance variable of any class is defined within a class in the hash table.
5) Finally, data is interpreteded by drawing a class diagram using java graphics.

Algorithm:
1) Provide the parser with the source files whose class diagram is to be created.
2) Store the retrieved data in the necessary format in the hash table.
3) Use the stored data to draw the class diagram.

Package Diagram Generation
A package diagram in the Unified Modeling Language depicts the dependencies between the packages that make up a model.

1) In java programming, package can be created using a package statement along with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package. The package statement should be the first line in the source file.
2) The package can be imported using the import keyword in java. A class file can contain any number of import statements. import package_name;
3) So our project includes a parser which consists of such regular expressions which identify these package names and their relationships with each other in the class definition and retrieves the needed data in hash table.
4) Finally, data is interpreted by drawing a package diagram using java graphics.

Algorithm
1) Provide the parser with the source files whose package diagram is to be created.
2) Store the retrieved data in the necessary format in the hash table.
3) Use the stored data to draw the class diagram.

4. Conclusions

The existing system convert the given source code into UML diagram, but all these tools are paid application and other which are open source lack behind in certain aspects which would be overcome by our proposed system. It is clear that, our system shows the clear conversion of Java code to class diagram and package diagram. In this system, we also have defined a number of coupling and accessibility metrics for object-oriented designs. The metrics not only allow designers to define the most secure design but they can also give indications of where any potential vulnerability occurs.

References

[1] Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software,By Lionel C. Briand, Senior Member, IEEE, YvanLabiche, Member, IEEE, and Johanne Leduc

[2] Extending the Sugiyama Algorithm for Drawing UML Class Diagrams: Towards Automatic Layout of Object Oriented Software Diagrams by: Jochen Seemann
[3] Static Control-Flow Analysis for Reverse Engineering of UML Sequence Diagrams. By: AtanasRountev Ohio State University, Olga Volgin University of Michigan Miriam Reddoch , Hewlett Packard
[4] W. De Pauw, E. Jensen, N. Mitchell, G. Sevitsky, J. Vlissides, and J. Yang, ―Visualizing the Execution of Java Programs,‖ Proc. Software Visualization, pp. 151162, 2002
[5] A. Sachitano, R. O. Chapman, and J. A. Hamilton, "Security in software architecture: a case study," in Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004, pp. 370-376.
[6] P. K. Manadhata, K. M. C. Tan, R. A. Maxion, and J. M. Wing, "An Approach to Measuring a System's Attack Surface," Carnegie Mellon University, Pittsburgh, PA August 2007.