Dynamic Approach for Measuring Internal Attributes of Object-Oriented System

[1] Amar Pal Yadav [2] C. S. Yadav

Abstract: Automated tool for measuring internal attributes of object oriented system has been developed to calculate the total lines of code, number of classes, number of methods, number of constructors, number of fields. With the help of these internal attributes, we measure the function point and use case point of the system. Again using these we estimate the effort and schedule of development effort. Results obtained from this tool is quite encouraging than the existing tools.

Keywords: lines of code; class; method; constructor; field; object-oriented system.

1. INTRODUCTION

It is useful tool for professional programmer to learn and apply effectively. The basic feature of object oriented programming is to learn and implement system than traditional programming language more easily. There are different types of languages which use the object oriented concept. These are Java, C++, similar etc.

Object-Oriented: It allows for designing and writing any program freely anywhere. The object oriented programming language provides clean, usable, pragmatic approach to objects. Borrowing liberally from many seminal object-software environments of the last few decades, Java manages to strike a balance between the "everything is an object" paradigm and the pragmatist's "stay out of my way" model. The object model in Java is simple and easy to extend, while simple types, such as integers, are kept as high-performance non objects.

Manuscript received May 2014

***AMAR PAL YADAV** M.Tech. UPTU / Noida Institute of Engineering and Technology Gr. Noida. Mob-9953148371 India.*
***C. S. YADAV** Associate Professor and Head, Computer Science & Engineering Department at Noida Institute of Engineering & Technology (NIET), Greater Noida India.*
Mob No-09313419956

Reflection is the part of object oriented system which is commonly used by the programs for acquire the ability to modify the runtime behavior of applications running in the JVM.

The concept of reflection is often mixed with introspection.

1. Introspection is the ability of a program to examine the type or properties of an object at runtime.
2. Reflection is the ability of a program to examine and modify the structure and behavior of an object at runtime.

By this definition, introspection is a subset of reflection. Some languages support introspection, but do not support reflection, e.g., C++.

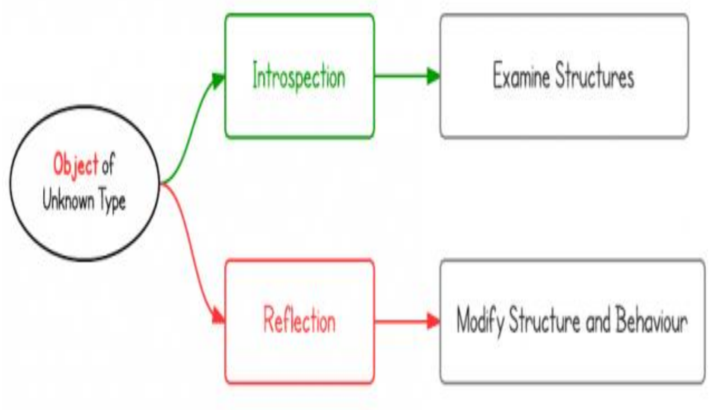

Figure 1: Relation of Introspection and Reflection

Through the example we can understand the introspection-

```
if (obj instanceof Dog)
            {
```

```
        Dog d = (Dog)obj;
        d.bark();
        }
```

The instance of operator determines whether an object belongs to a particular class.

For Reflection we can take example-
The Class.forName() method returns the Class object associated with the class/interface with the given name(a string and full qualified name). The forName method causes the class with the name to be initialized.

```
// with reflection
Class<?> c = Class.forName("classpath.and.classname");
Object dog = c.newInstance();
Method m = c.getDeclaredMethod("bark", new Class<?>[0]);
m.invoke(dog);
```

In Java, reflection is more about introspection, because you cannot change structure of an object. There are some APIs to change accessibilities of methods and fields, but not structures.

(i.) How to find the internal attributes those used for various levels in a program. How to find the complexity of a program to require count information about the number of statements, compound statements, keywords, literals etc in a large program for cost estimation.
(ii.) Existing System and tools work only with find the number of classes, methods and no of lines in large software. But our approach work with finding all attribute used within a program. Our methodology work very efficiently with accuracy to count the number of statements, classes, data types, literals, variables etc.

2. RELATED WORK

We chose those subject programs because their source code is easily accessible and because all projects come with a decent set of test cases. As explained earlier, test cases are necessary for our approach. For each project, we first used the RefaFlex Eclipse plugins to execute the respective test case, yielding a log file filled with information on all reflective accesses occurring on those test runs. The projects and test cases are available on our website, along with our implementation (in source) and detailed documentation on how to produce our results [1, 2]. The large explosion of business demands and enterprise wide applications has created the need for different approaches to software development in order to facilitate business collaboration and growth. This has led to the adoption of Service-Oriented Architecture (SOA) for building highly distributed and integrated enterprise-wide software systems that use web services as its building blocks. A web service can be thought of as an autonomous software component that implements specific business rules and logic to perform a particular functionality. The ability to encapsulate business rules and logic into web services that can be accessed by applications or other web services provides a high level of separation of concerns and greater opportunities of reusability [2, 3]. Over the past three decades, several software development methodologies have appeared. Such methodologies address some or all phases of the software life cycle ranging from requirements to maintenance. These methodologies have often been developed in response to new ideas about how to cope with the inherent complexity of software systems. Due to the increasing popularity of object-oriented programming, in the last twenty years, research on object-oriented methodologies has become a growing field of interest [4]. The fact that the software industry is nowadays confronted with a big number of large-scale legacy systems written in an object-oriented language, which are monolithic, inflexible and hard to maintain shows that just knowing the syntax elements of an object-oriented language or the concepts involved in the object-oriented technology is far from being sufficient to produce good software [5, 6, 7].

3. METHODOLOGY

For every loaded class, the Java Runtime Environment (JRE) maintains an associated Class objects (i) The

Class object “reflects” the class it represents (ii) Can use the Class object to discover information about a loaded class-

- Name
- Modifiers (public, abstract, final)
- Super classes
- implemented interfaces
- Fields
- Methods
- Constructors

```
Import java.lang.reflect.*;
Public class ReflectionDemo1
{
     Public static void main (String args[ ])
        {
        Try
                {
          Class c= Class.for Name (“java. awt.Dimension”);
          System.out.println (“Constructors:”);
          Constructor constructors [ ] =c.getConstructors ();
          for (int i = 0; i < constructos.length; i++);
          System.out.println (“ ” + constructors [i] );
           }
        System.out.println (“Fields:”);
        Fields fields [ ] = c.getFields ( );
        for (int i = 0; i < fields.length; i++)
                {
                  System.out.println (“ ” + fields [i] );
                }
             System.out.println (“Methods :”);
             Method methods [ ] = c.getMethods ( );
             for (int i =0; i < methods.length; i++);
                  {
                   System.out.println (“ ” +methods [i]);
                  }
             }
           catch ( Exception e)
                {
                  System.out.println (“Exception:” +e);
                }
        }
  }
```

4. RESULT AND DISCUSSION

As the reader may have noticed, the possible effects of refactoring on Java programs are anything but trivial to foresee. Many code transformations may need to be disallowed because this model calls would cause these transformations to be not behavior preserving [8]. In some cases, however, one can design additional code transformations on the code itself that, when combined with the refactoring transformation, will cause the combined transformation to preserve the program's behavior after all. In the remainder of this paper we propose a solution that (a) records information about how a program uses at runtime, (b) uses refactoring constraints based on this information to prevent semantics-changing code transformations and in some cases (c) allows the rewriting of code that uses the API such that the rewritten code will exhibit the same behavior as the original program for the given test cases [9, 10, 11].

In the previous sections we claimed that C++ would give programmers the guarantee that estimation of program executed before refactoring will still pass after refactoring, even if is involved. To validate this claim, checked that all program variants produced by applying our methodology still compile, and runtime produce total quantity of used attribute with in a class [12, 13, 14].

We test our approaches large and small programs of java for cost estimation of the project.

We checked our methodology and produce result-

Table 1:

Project ID	*NOP	*NOC	*NOM	*NOV	*NOS	*NOCS
P1_Code	01	01	02	02	32	03
P2_Code	03	02	04	06	46	07
P3_Code	02	01	09	12	57	06
P4_Code	04	02	03	09	102	11
P5_Code	07	03	13	11	193	10

Figure 2: Different cost estimation of the project

*(NOP: No. of Package, NOC: No. of Classes, NOM: No. of Methods, NOV: No. of Variables, NOS: No. of Statements, NOCS: No. of Compound Statements)

Here in figure 2 we have a table which shows different project P1_Code, P2_Code, P3_Code, P4_Code, P5_Code and calculate the different internal attributes of the project. Such as Project P1_Code project have number of package (NOP) 01,number of class (NOC) 01, number of method (NOM) 02, number of variable (NOV) 02, number of statement (NOS) 32, number of compound statement (NOCS) 03.

In Project P2_Code project have number of package (NOP) 03,number of class (NOC) 02, number of method (NOM) 04, number of variable (NOV) 06, number of statement (NOS) 46, number of compound statement (NOCS) 07.

In Project P3_Code project have number of package (NOP) 02,number of class (NOC) 01, number of method (NOM) 09, number of variable (NOV) 12, number of statement (NOS) 57, number of compound statement (NOCS) 06.

In Project P4_Code project have number of package (NOP) 04, number of class (NOC) 02, number of method (NOM) 03, number of variable (NOV) 09, number of statement (NOS) 102, number of compound statement (NOCS) 11 and in Project P5_Code project have number of package (NOP) 07,number of class (NOC) 03, number of method (NOM) 13, number of variable (NOV) 11, number of statement (NOS) 193, number of compound statement (NOCS) 10.

In Figure 3 have to represent the Table data different cost estimation in 3 Dimension form where all the internal attribute of the Object-Oriented System have show Number of Package (NOP), Number of Classes (NOC), Number of Methods (NOM), Number of Variables (NOV), Number of Statements (NOS), Number. of Compound Statements (NOCS) of the different project P1_Code, P2_Code, P3_Code, P4_Code, P5_Code.

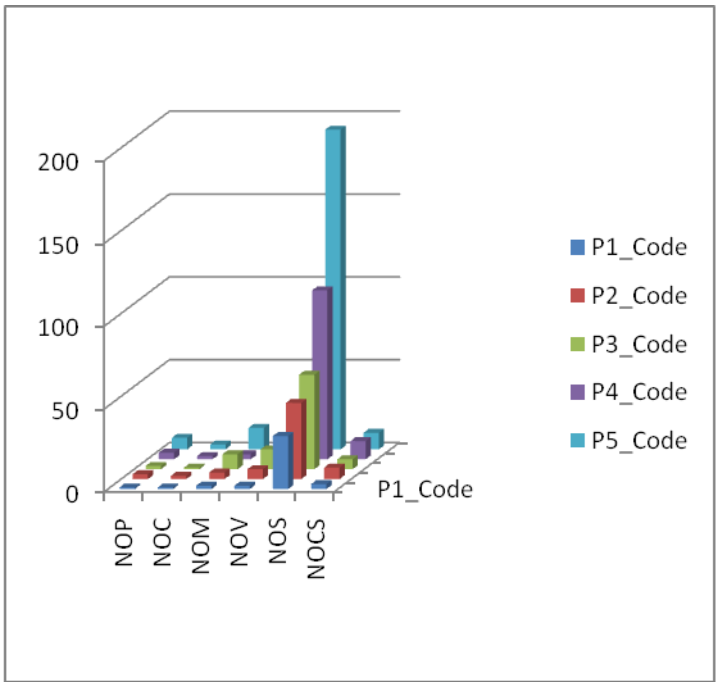

Figure 3: Different cost estimation of the project in 3D

In Figure 4 have to represent the Table data different cost estimation in 2 Dimension form where all the internal attribute of the Object-Oriented System have show Number of Package (NOP), Number of Classes (NOC), Number of Methods (NOM), Number of Variables (NOV), Number of Statements (NOS), Number. of Compound Statements (NOCS) of the different project P1_Code, P2_Code, P3_Code, P4_Code, P5_Code.

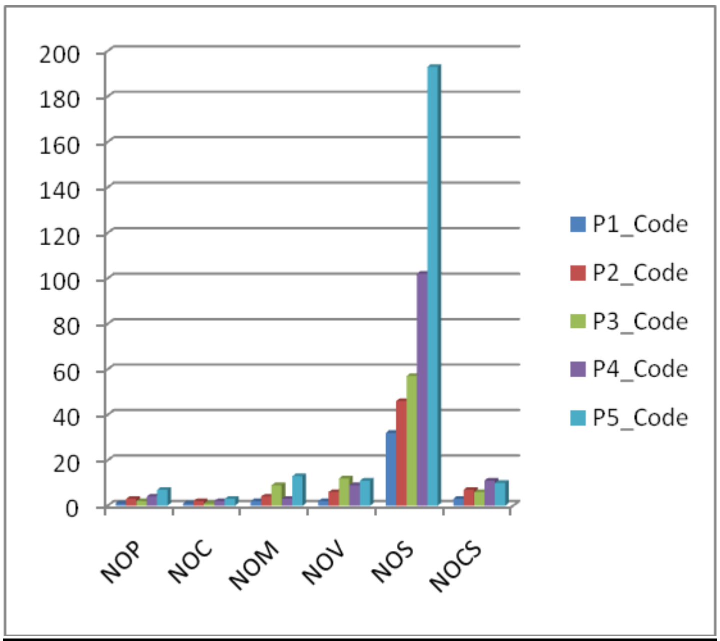

Figure 4: Different cost estimation of the project in 2D

5. CONCLUSION

We have presented a novel approach and tool to providing safer refactoring for Java programs. Our methodology uses runtime information from test runs to generate constraints and then considers these constraints when computing a refactoring preconditions and transformation. It thereby guarantees that the refactoring cannot break the recorded test runs, even if they use Java's API.

Our paper also highlights some interesting corner cases in which the semantics of Java's API deviates from the semantics of the Java language. For example, accessibility checks in the API do not at all consider the protected modifier. We show other interesting cases, in which behavior preservation may be unintended. We can use this methodology for cost estimation and testing of large program with help of measuring internal attributes.

REFRENCES

[1] Oscar Callau, Romain Robbes, Eric Tanter and David Rothlisberger. How developers use the dynamic features of programming languages: the case of smalltalk. In MSR, MSR '11, pages 23{32, New York, NY, USA, 2011. ACM.

[2] Assessing Internal Software Quality Attributes of the Object-Oriented and Service-Oriented Software Development Paradigms: A Comparative Study Yaser I. Mansour, Suleiman H. Mustafa Journal of Software Engineering and Applications, 2011, 4, 244-252.

[3] A Brief History of the Object-Oriented Approach Luiz Fernando Capretz University of Western Ontario Department of Electrical & Computer Engineering London, ON, CANADA, N6G 1H1 Software Engineering Notes vol 28 no 2 March 2003

[4] Measurement and Quality in Object-Oriented Design Radu Marinescu LOOSE Research Group "Politehnica" University of Timis¸oara Bvd. V. Pˆarvan 2, 300223 Timis¸oara (Romania).

[5] E. Bodden A. Sewe, J. Sinschek, H. Oueslati, and M. Mezini. Taming reection: Aiding static analysis in the presence of reflection and custom class loaders. In ICSE, pages 241{250, 2011.

[6]E. Bodden, A. Sewe, J. Sinschek, and M. Mezini.Taming reection: Static analysis in the presence of reflection and custom class loaders. Technical Report TUD-CS-2010-0066, CASED, March 2010.

[7]M. Sridharan, S. Artzi, M. Pistoia, S. Guarnieri, O. Tripp, and R. Berg. F4F: Taint analysis of framework-based web applications. In OOPSLA, OOPSLA '11, pages 1053{1068. ACM, 2011.

[8]J. Sawin and A Rountev. Assumption hierarchy for a CHA call graph construction algorithm. In SCAM'11, pages 35{44, 2011.

[9]M. Schafer, J. Dolby, M. Sridharan, E. Torlak, and F. Tip. Correct refactoring of concurrent Java code. In ECOOP, pages 225{249, 2010.

[10]J. Sawin and A. Rountev. Improving static resolution of dynamic class loading in Java using dynamically gathered environment information. International Journal of Automated Software Engineering, 16(2):357{381, June 2009.

[11] M. Schafer, T. Ekman, and O. de Moor. Sound and extensible renaming for Java. In OOPSLA, OOPSLA '08, pages 277{294, New York, NY, USA, 2008. ACM.

[12]M. Hirzel, D. von Dincklage, A. Diwan, and M. Hind.Fast online pointer analysis. TOPLAS, 29(2):11, 2007.

[13]S. Blackburn, R. Garner, C. Ho_mann, A. Khang, K. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, E. Moss, A. Phansalkar, D. Stefanovi_c, T. Van Drunen, D. von Dincklage, and B. Wiedermann. The DaCapo benchmarks: Java benchmarking development and analysis. In OOPSLA, pages 169{190. ACM, 2006.

[14]M. Hirzel, A. Diwan, and M Hind. Pointer analysis in the presence of dynamic class loading. In ECOOP, pages 96{122, 2004.

ABOUT AUTHOR:

1.

AMAR PAL YADAV: M.TECH. Department of Computer Science & Engineering, Noida Institute of Engineering and Technology Greater Noida.

Mob No-09953148371

2.

C. S. YADAV: Prof. Chandra Shekhar Yadav is an Associate Professor and Head, Computer Science & Engineering Department at Noida Institute of Engineering & Technology (NIET), Greater Noida. He has about 17 years of experience in teaching. He received Master of Computer Application degree from Institute of Engineering & Technology (IET), Lucknow in year 1998, M. Tech (Computer Science & Engineering) from JSSATE, Noida in year 2007. Currently he has submitted Ph.D. (Computer Science & Engineering) thesis in U.P. Technical University, Lucknow under the supervision of Prof. (Dr.) Raghuraj Singh, Director KNIT Sultanpur. He has supervised 07 M. Tech. theses.

Mob No-09313419956