IJFANS INTERNATIONAL JOURNAL OF FOOD AND NUTRITIONAL SCIENCES
ISSN PRINT 2319 1775 Online 2320 7876
 UGC CARE Listed (Group -I) Journal Volume 11, Iss 10, 2022

Impact analysis in object-oriented systems for Dynamic Coupling Measures

L. K Suresh Kumar
Associate Professor, Department of Computer Science
UCE, Osmania University, suresh.l@uceou.edu

***Abstract*— In recent years, researchers have explored the link between coupling and object-oriented programme quality. Several studies have shown correlations between class-level coupling and fault-proneness. Static code analysis quantifies coupling. Statically analysing code helps monitor security, reliability, performance, and maintainability. Static code analysis exposes structural problems and prevents entire classes of issues when done correctly. Most current work was done on non-object oriented code, and dynamic code analysis is more expensive and difficult. This reliance on static analysis might be troublesome for current software systems, because dynamic binding existed before OO.**

Keywords*— *class-level coupling and fault-proneness. Static code analysis ,security, reliability, performance, and maintainability

I. INTRODUCTION

Coupling, sometimes known as dependence, is a term used in the field of computer science to refer to the degree to which one software module is dependent on all of the other modules. When paired with high cohesiveness, low coupling helps promote the overall goals of high readability and maintainability in a computer system. This is because low coupling is frequently a symptom of a well-structured computer system and a good design.

The fundamental objective of coupling measurements is to quantify the degree to which a class is related to other classes. These metrics are often derived from some type of static code analysis. For instance, class A is considered to be linked to class B if one of A's methods invokes one of B's methods. As a result of the fact that coupling measurements are used to assist software developers, testers, and maintainers in reasoning about the complexity of software and the quality attributes of software, it has significant applications in the software development and maintenance fields. There has been a lot of research done on coupling metrics, and they have been put to use to help maintainers with a variety of tasks, including impact analysis, determining the fault-proneness of classes, fault prediction, ripple effects, and changeability, to name just a few [5, 6, 11, 12, 9]. Therefore, coupling metrics aid developers, testers, and maintainers in thinking about the programme and in projecting the demands for code inspection, testing, and debugging.

II. METHODS FOR MEASURING DYNAMIC COUPLING

The results of dynamic analysis, which are now being used to gather dynamic coupling measures, indicate that these measures are superior both as markers of complexity and as predictors of the quality characteristics they measure.

Only "static" couplings are taken into consideration by the traditional coupling measurements [5,7]. They may considerably underestimate the complexity of the software and misunderstand the necessity for code inspection, testing, and debugging since they do not account for "dynamic" couplings that are the result of polymorphism. Take into consideration the typical bridge construction [8] seen in figure 1. Traditional coupling measures [5,7] would only count a single coupling because of a polymorphic call to imp.DevM() in method A:m, and that coupling would be a coupling of A to Imp. In point of fact, the actual complexity of the call is significantly higher because the polymorphic call can couple "dynamically" each concrete A to each concrete Imp for a total of six couplings: 'A1,Imp1', 'A1,Imp2', 'A1,Imp3', 'A2,Imp1', 'A2,Imp2', and 'A2,Imp3'. In addition, the actual complexity of the call is significantly higher because the poly

Arisholm et al. do in-depth statistical analysis, which demonstrates that these measures are superior than standard coupling measures in terms of their ability to indicate the level of complexity present and to predict the quality of certain qualities.

III. ANALYSIS OF THE STATIC CLASS

However, dynamic analysis has a number of drawbacks, including the following:
i) it is relatively slow, (ii) it requires a complete programme, and (iii) it produces incomplete results as the couplings output by the dynamic analysis are valid for particular inputs and executions of the programme.
The utilisation of static analysis as an alternative to dynamic analysis for the computation of dynamic coupling measures is the purpose of the entire body of work that has been done. It's possible that the static analyses discussed in this work offer numerous benefits that dynamic studies don't have[1]:
I they are applicable in real-world situations, (ii) they are able to operate on incomplete programmes, and (iii) they generate outcomes that are consistent across all programme executions.

A technique known as "programme analysis" examines a program's source code in order to draw conclusions about how it will behave when it is executed. Compiler optimization has historically been accomplished through the use of programme analysis. An example of this would be an analysis that looked at the programme, determined which expressions produced the same result, and then rewrote the code to get rid of any redundant computations; in most cases, this results in the programme running more quickly. In addition, programme analysis may

ISSN PRINT 2319 1775 Online 2320 7876
Research paper UGC CARE Listed (Group -I) Journal Volume 11, Iss 10, 2022

be used in a broad number of applications within the tools used to improve the productivity and quality of software. We present a system for the static analysis of tightly typed programming languages such as Java, which allows for the computation of dynamic coupling measures derived from [3]. This method is successful when used to unfinished programmes. Because it is necessary to be able to conduct individual analyses of software components, this is a trait that is considered to be very significant.

At the lower end of the cost/precision continuum, the framework with Class Hierarchy Analysis (CHA) and Rapid Type Analysis (RTA), two well-known class analyses, is launched. On a number of different aspects, empirical results are now being offered. An evaluation of precision is given, which demonstrates that analyses based on RTA achieve almost perfect precision. This means that it computes almost exactly the same couplings as would have been identified dynamically with the most comprehensive suite of test cases written on top of the component had it been subjected to the same testing. Based on these findings, it appears that dynamic coupling measures may be quantified correctly using a method that is both straightforward and affordable: static analysis. The following are some of the contributions made by this work: For the purpose of computing dynamic coupling measures, we suggest using a framework for static analysis that is parameterized by class analysis. Our analysis framework is built to handle programmes that are only partially complete. We offer an empirical research that examines the effects of two different instantiations of the framework on a number of different Java components.

The Argument in Favor of Using Static Analysis: By utilising dynamic analysis, Arisholm et al. [3] are able to capture dynamic coupling measures; nevertheless, they note that dynamic analysis does have certain limitations. First, because it needs numerous steps, it is a somewhat lengthy process. Second, the process of designing and constructing an instrumentation framework is rather difficult from an engineering standpoint. Third, in order to do a dynamic analysis, a whole programme is always necessary. Fourth, because the results that are acquired are dependent on certain iterations with specific inputs, it is possible that the results that are obtained are insufficient.

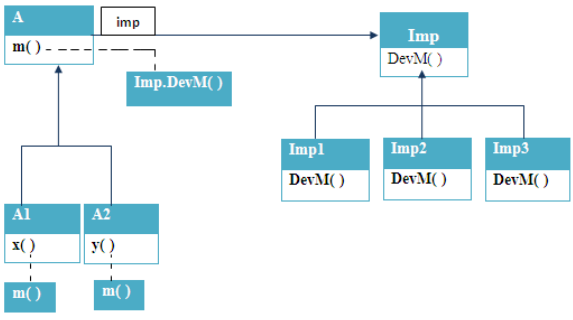

Fig 1 :Bridge Structure

Figure 1 depicts two different ways to instantiate A1: the first uses Imp1, while the second uses Imp2. Due to the execution of the two clients, the dynamic couplings 'A1,Imp1' and 'A1,Imp2' will be counted as being due to the call imp.DevM. (). As a result, the dynamic analysis won't take into account those four legitimate couplings. In order to accurately capture all of the possible dynamic couplings that are a result of this call, one would require at least six clients and six runs.

For the aim of computing dynamic coupling measures, static analysis [14], and in particular class analysis, provides a potential alternative to dynamic analysis as an option to consider. [Case in point:] One may get an approximation of the probable classes of each reference variable and reference object field by using class analysis, and then use that information to reason about the possible dynamic couplings. When it comes to computing dynamic coupling measurements, static analysis provides a number of significant benefits over dynamic analysis. To begin, the dynamic studies that are not included in this article are likely to be substantially less useful and far more expensive than the static analyses that are. Second, these analyses are simple to apply; it is likely that they are used for the sake of important activities such as the development of call graphs since they are straightforward to do. Third, the static analysis that was looked at may be modified to function on incomplete programmes (also known as software components). This is an important advantage. In the fourth place, the static analysis is conservative, which means that any and all feasible run-time couplings will definitely be recorded. This is a promise.

Static analysis, on the other hand, has the potential to be unduly cautious and indicate infeasible couplings. This is because infeasible couplings are those that cannot take place during any execution of the programme.

IV. Problem Statement

Our study is geared toward the development of a static analysis that is capable of accurately computing dynamic coupling measures, and the analysis must be able to function on incomplete programmes. The analysis receives as input a set of interacting Java classes denoted by the notation Cls. The set of accessible classes is a subset of Cls; these are classes that may be accessed by unknown client code from outside of Cls. This subset of Cls is identified as the set of accessible classes. These types of client code are only able to access the fields and methods of Cls that have been defined in some accessible class. The fields and methods that have been declared in accessible classes are referred to as boundary fields and boundary methods.

A. Methods for Measuring Dynamic Coupling

The object coupling metrics that capture polymorphic interactions are the primary focus of this body of work. When the dynamic analysis comes across an execution of method m1 with a receiver of type C1 and, during the course of this execution, call site I in method m1 invokes method m2 with a receiver of class C2, the measure of highest granularity, which is denoted by IC OD in [2], records a tuple that reads "C1,m1,I,C2,m2." Remember the example from Figure 1 and make the assumption that the number 1 is in the call site imp.DevM() in method A:m. The dynamic analysis, assuming that there are tests that assure complete coverage, would report the following IC OD tuples for call site 1:

〈A1,A:m,1,Imp1,Imp1:DevM()〉

〈A1,A:m,1,Imp2,Imp2:DevM()〉

ISSN PRINT 2319 1775 Online 2320 7876
Research paper UGC CARE Listed (Group -I) Journal Volume 11, Iss 10, 2022

〈A1,A:m,1,Imp3,Imp3:DevM()〉

〈A2,A:m,1,Imp1,Imp1:DevM()〉

〈A2,A:m,1,Imp2,Imp2:DevM()〉

〈A2,A:m,1,Imp3,Imp3:DevM()〉 .

The measure of intermediate granularity, which is designated by the symbol IC OM in [3], records tuples with the call site index I removed from the IC OD tuple. These are the tuples C1,m1, and C2,m2. The measure with the smallest amount of granularity, which is designated by the symbol IC OC in [3], keeps track of tuples with the callee method removed from the IC OM tuple. It is very clear that IC OD incorporates IC OM, and IC OM incorporates IC OC.

It is possible to compute a measure for each class by making use of these tuples. For instance, the IC OD measure for class C is equal to the number of tuples for which the caller method m1 is a method declared in C.

B. *Discussion*

In order to meet the requirements of additional problem definitions that call for an analysis of incomplete programmes, we use the following constraint and standard [10]. Only the executions in which the invocation of a boundary method stays within Cls, meaning that all of its transitive callees are likewise located within Cls, are taken into consideration by us. When we take into account the prospect of undiscovered subclasses, it is possible that any instance calls made from Cls will be "redirected" to undiscovered external code, which might have an impact on the coupling inference. Therefore, Cls is expanded to encompass not just the classes that offer functionality for components but also all additional classes that are referenced in a transitive manner. In the tests described in Section 5, we included all classes that were referred to by Cls in a circular fashion. This strategy confines analytical information to the "known world" as it exists at the moment; in other words, the knowledge may become obsolete in the future as a result of the addition of new subclasses to the Cls.

V. AN ANALYSIS OF THE FRAGMENT CLASSES FOR THE DYNAMIC COUPLING MEASURES

Class analysis is a process that identifies the object classes to which a certain reference variable or field may point. This may be done by looking at the values of the variable or field.

When arbitrary client code is written on top of Cls, the knowledge obtained from class analysis is utilised to provide an approximation of the set of IC OD coupling tuples that can occur. In order to accomplish this, the class analysis solution is utilised to generate an approximation of the set of potential classes for the caller as well as the set of possible classes for the callee. In this study, we examine the calculation of dynamic coupling measures based on two common and straightforward class analyses: Class Hierarchy Analysis (CHA) [13] and Rapid Type Analysis (RTA) [13]. Both of these analyses are widely known.

The CHA method is the most basic type of class analysis. The structure of the class hierarchy is investigated by CHA in order to ascertain the many ways in which polymorphic variables and fields may be bound. An implementation of CHA will normally make the assumption that there is an entire programme, which means that there is a main method that serves as the beginning of the execution of the programme; it will begin at main and keep a collection of methods R that are available from there. Whenever CHA discovers a method that can be reached, it investigates the call locations included inside that method. It does this by first approximating the available run-time classes for r based on the hierarchy, and then for each C it determines the suitable run-time target mj based on C and the compile-time target m. This process is repeated for each call site l = r.m(...). R is expanded by one for every one of these mj.

RTA is an additional straightforward method of class analysis. It is an improvement over CHA since it takes into consideration the classes that are actually instantiated in the application. The process begins with an exercise known as Class Hierarchy Analysis, which produces a call graph. It does this by using information about the classes that have been created to further minimise the number of executable virtual functions, which in turn reduces the size of the call graph.

RTA operates on the assumption that there is a whole programme and begins execution with the main method as the first accessible method. RTA will investigate both the call sites and the instantiation sites inside an accessible method whenever one of those methods is discovered. In a nutshell, the RTA solution for a reference variable or a reference object field is the collection of instantiated classes that are compatible with the type that was defined for it.

A. *Analysis of the Fragment Class*

Class analyses are often conceived of as whole-program analyses. This means that they take a full programme as their input and generate a class analysis solution that represents bindings across the entirety of the programme as their output. Nonetheless, the problem that is going to be looked at in this work calls for a class analysis of a portion of the programme. The analysis has to approximate dynamic couplings that may happen across various executions of any client code built on top of the classes that make up the input, and the input itself is a collection of classes called Cls. In order to solve this issue, we have resorted to a method known as fragment analysis, which is an overarching strategy. Instead of working on an entire programme, fragment analysis is performed on a portion of that programme; in our example, that portion is the collection of classes known as Cls.

The fragment analysis will first provide a fake main method that will act as a placeholder for client code that will be put on top of Cls. Intuitively, the fake main acts as a simulation of the probable flow of classes between the client code and Cls. After that, the fragment analysis[32] binds main to Cls and makes use of some whole-program class analysis engine to generate class analysis information. This information summarises the impacts that may be caused by arbitrary clients.

B. *The Calculation of Static and Dynamic Coupling Measures*

Input Methods: the collection of methods that may be accessed

Statements in Methods Set, or Stmt for short.

ISSN PRINT 2319 1775 Online 2320 7876

Cs: Vars → P(Classes) output Dyn: Cls × Methods × I × Cls × Methods

i. foreach virtual call si: r.m (...) s.t.r. equals this

ii. if si is included in the static procedure C1:n, then...

iii. add { 〈C1, n, I C2, target(C2, m)〉 | C2 ∈ Cs(r) ∧ C1 = C2} to Dyn

iv. otherwise if the si variable is included within the n instance method (n can be constructor)

v. add { 〈C1, n, I C2, target(C2, m)〉 | C1 ∈ Cs(thisn) ∧ C2 ∈ Cs(r) ∧ C1 = C2 } to Dyn [6] for each static call, enter C2.m in the si variable (...)

vi. If si is included within the static method C1:n, then C1 Equals C2.

vii. add 〈C1, n, I C2, m〉 to Dyn

viii. otherwise, if si is included within instance method n (n can be constructor)

ix. add { 〈C1, n, I C2, m〉 | C1 ∈ Cs(thisn) ∧ C1 = C2 } to Dyn

The technique depicted is responsible for computing Dyn, which is the collection of tuples C1,n,i,C2,m that provides an approximation of the various dynamic couplings that can occur when arbitrary client code is written on top of Cls. The class analysis variable, Cs, is used to set the parameters for the algorithm. The accuracy of the computation of the dynamic measures is directly proportional to the accuracy of the class analysis that lies behind it. If the class analysis is more exact, then the set of couplings will also be more precise. This is because a more precise class analysis will result in the solution having fewer classes for each reference variable.

VI. RESULTS

RTA is demonstrably superior to CHA in terms of accuracy. These results make the distinction between the granularity levels of the three object coupling measures abundantly clear; for the majority of components, the IC OD is a sizeable amount greater than the IC OM, and the IC OM is a sizeable amount larger than the IC OC. This accurately depicts the organisation of the code. It is common for the same virtual call to appear multiple times within the same method at different call sites. When the call site is dropped from the tuple, previously distinct tuples are treated as one, and as a result, there is a drop from IC OD to IC OM. This is because when the call site is dropped, previously distinct tuples are treated as one. When the callee is removed from the tuple, separate tuples are combined into one, which is what causes the change from IC OM to IC OC. Another common occurrence is when many methods with the same receiver class are called from within the same method. Additionally, we classify the IC OD tuples as either monomorphic or polymorphic depending on their structure. The monomorphic tuples are the outcome of call sites that are resolved in a singular manner by CHA. This means that both the caller class and the callee class are singular in CHA's estimation. There are three types of polymorphic tuples: I those that are polymorphic only in the caller; (ii) those that are polymorphic only in the callee; and (iii) those that are polymorphic both in the caller and in the callee.

According to the findings of the class analysis, there are three distinct categories of imp: Imp1, Imp2, and Imp3. The outcomes of this classification process for the RTA IC OD tuples are presented in Table 2.

Component	#MONO	#POLY			
		Caller	Callee	Both	Total
Gzip	2				
Zip	24	0	0	0	0
checked	4	0	0	0	0
collator	107	0	23	0	23
Date	171	13	161	0	164
number	98	0	15	0	15
boundary	182	0	48	0	48

Table 2: Dynamic tuples category statistic.

Most of the polymorphic tuples are callee-polymorphic; however, caller polymorphic tuples still exist and it is important that an analysis for the computation of dynamic coupling measures considers both polymorphism in the caller and in the callee.

A. Analysis Precision

The issue of analysis precision is important for the static analysis for the computation of dynamic coupling measures. If the analysis is imprecise it may report coupling tuples that cannot happen for any execution of the program.We examined the IC_OD tuples computed by the two instantiations of our analysis and for each tuple we attempted to write client code that would exhibit that tuple. We were able to prove that all monomorphic tuples in our code base are feasible.

Component	#Polymorphic tuples		
	CHA-based	RTA-based	Actual
Gzip	24	6	5
Zip	5	0	0
checked	0	0	0
collator	27	23	23
Date	187	164	164
number	22	15	15
boundary	57	48	48

Table 3 shows the number of polymorphic tuples: the number for the CHA-based analysis, the number for the RTA-based

Statistically inferred polymorphic tuples are shown in Table 3, along with the actual number of tuples retrieved by manual examination. Table 3 demonstrates that the RTA-based analysis attains an almost perfect level of precision since all tuples, with the exception of one, may be exercised by a customer.

ISSN PRINT 2319 1775 Online 2320 7876
Research paper **UGC CARE Listed (Group -I) Journal Volume 11, Iss 10, 2022**

The static analyses that were taken into consideration for this research are straightforward and quick to put into action; they also have a cost that is essentially proportional to the size of the programme. When utilised in practise, these studies have the potential to lead to more efficient computation of dynamic coupling measurements. In addition, the analysis that is based on RTA is capable of virtually perfect accuracy, and as a result, it may be able to offer a feasible, more convenient, and more practically applicable alternative to dynamic analysis for the purpose of computing dynamic coupling measures.

There is a significant amount of work being done on coupling measures[7,5,6,12,11,9]. These coupling measurements are commonly derived by code analysis; however, they do not take polymorphism into account, which results in an underestimate of the complexity of the code. The foundation of our work is also static analysis; however, the metrics that we compute take into account polymorphism in a very direct way.

Polymorphism is taken into account in a group of coupling measures that have been established by Eder et al., however the calculation of these measures is not taken into account. On the other hand, the focus of the work presented in this study is on the computation of coupling measures; it offers a static analysis approach that captures the measurements. It is very clear that the two approaches are complementary to one another.

VII. Conclusion

A novel strategy for computing dynamic coupling measurements is given. This strategy makes use of static code analysis. The following is a list of the primary contributions that our work has made. First, a framework for static analysis that works on unfinished programmes is provided. This framework is intended for the computation of dynamic coupling measures for tightly typed object-oriented languages such as Java. Second, there is a presentation of actual research that demonstrates the accuracy of the analysis to be extremely close to flawless. As a result, a practical alternative to the more time-consuming and difficult dynamic analysis that is often performed for the purpose of computing dynamic coupling measures has been offered.

References

[1] for object-oriented software. In IEEE METRICS, pages 33-42, 2002.

[2] E. Arisholm, L. Briand, and A. Foyen. Dynamic coupling measurement for object-oriented software. IEEE Trans. SoftwareEngineering, 30(8):491-506, 2004.

[3] E. Arisholm, D. Sjoberg, and M. Jorgensen. Assessing the chaneability of two object-oriented design alternatives—a controlled experiment. Emperical Software Engineering, 6(3):231-277, 2001.

[4] M. Berndl, O. Lhotak, F. Qian, L. Hendren, and N. Umanee. Points-to analysis using BDD's. In ACM Conference on Programming Language Design and Implementation, pages 103-114, 2003.

[5] L. C. Briand, P. T. Devanbu, and W. L.Melo. An investigation into coupling measures for C++. In ACM/IEEE International Conference on Software Engineering, pages 412-421, 1997.

[6] L. C. Briand, J. Wust, and H. Lounis.Using coupling measurement for impact analysis in object-oriented systems. In IEEE International Conference on Software Maintenance, pages 475-482, 1999.

[7] S. Chidamber and C. F. Kemerer. Towards a metrics suite for object oriented design.In ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, pages 197-211, 1991.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides.Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[9] T. Gyimoty, R. Ferenc, and I. Siket. Emprirical validation of object-oriented metrics on open source software for fault prediciotn. IEEE Trans. Software Engineering, 31(10):897-910, 2005.

[10] A. Rountev. Precise identification of side effect free methods. In IEEE International Conference on Software Maintenance, pages 82-91, 2004.

[11] F. G. Wilkie and B. Kitchenham. Coupling measures and change ripples in C++ application software. Journal of Systems and Software, 52(2):157-164, 2000.

[12] P. Yu, T. Systa, and H. A. Muller. Predicting fault-proneness using oo metrics:An industrial case study. In European Conference on Software Maintenance and Reengineering, pages 99-107, 2002.

[13] Atanas Rountev , Ana Milanova and Barbara G. Ryder.Fragment Class Analysis for Testing of Polymorphism in Java Software,pages 3-11