IJFANS INTERNATIONAL JOURNAL OF FOOD AND NUTRITIONAL SCIENCES

ISSN PRINT 2319 1775 Online 2320 7876

Research paper **© 2012 IJFANS. All Rights Reserved**, **UGC CARE Listed (Group -I) Journal Volume 11, Iss 12, 2022**

The Influence of Dynamic Coupling on Object-Oriented Software and Its Implementation

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

***Abstract*— Software quality is important to developers, businesses, and governments. As is well-known, software maintainability determines quality. Coupling measurement is one way to benchmark an application's maintainability. Static code analysis is used in traditional measurement techniques during system testing or trial operations. When we were using COBOL, FORTRAN, Pascal, and C, static analysis was helpful. Dynamic coupling may be used to assess systems and applications with the emergence of object-oriented (OO) languages like C++, Visual C, and Java. Dynamic coupling matches the reality of the tested application since it can uncover problems and shortages that are not predicted with static coupling. Dynamic coupling, on the other hand, requires a lot longer to quality-test, particularly if the subject application is substantial.**

Maintenance is a follow-up activity in software development, and maintainability is a post-delivery quality attribute. In addition, realizing maintenance costs for a target software system couldn't be decided off-hand for maintainers on both sides (i.e., system supplier and developer on one side and customer on the other). The author assumes coupling measures as a proactive action for the quality of the target application and cost-benefit considerations for the stakeholders. OO modeling results in a more manageable and reusable system. This research examines the link between coupling measures and object-oriented system quality and maintainability. This study covers dynamic coupling measurements, data collection, conceptual coupling, static coupling advantages, conclusions, and references.

Keywords — dynamic coupling measurements, static coupling, Software quality, software development and maintainability, coupling measures.

I. INTRODUCTION

Research on quality models in the context of object-oriented systems has mostly focused on developing structural metrics (such as class coupling) and examining their correlations with external quality features (such as class fault-proneness). The majority of the research in this field has mostly focused on this path of investigation. The ultimate objective is to create predictive models that may be used for decision-making in some form, such as choosing which classes should undergo more thorough verification and validation. Most metrics have been established and gathered up to this point based on a static study of the design or code. This is true regardless of the structural attribute that is being taken into consideration. In several situations, they have proven time and time again to be reliable predictors of external quality qualities including fault-proneness, ripple effects after modifications, and changeability. On the other hand, the majority of the systems that have been investigated display a low level of inheritance and, as a consequence, make only a limited use of polymorphism and dynamic binding. We have noticed that inheritance and polymorphism are being used more frequently in the industry as object-oriented design and programming are increasingly used. This is being done to improve the level of internal reuse within a system and to make system maintenance easier. This is evident when looking at the growing number of open source projects, application frameworks, and library collections, despite the fact that there has been no formal survey conducted on the subject.

The issue is that when more intense usage of inheritance and dynamic binding occurs, the static and coupling measures — which constitute the key indications of the great majority of the quality models that have been reported—lose accuracy. It is projected that as a result, the prediction accuracy of the quality models that depend on static coupling measurement would decline.

The evidence that we currently have suggests that dynamic coupling might be of significant interest. According to the findings of a preliminary empirical study conducted on a Small Talk system, there appears to be a significant link between dynamic coupling and change susceptibility. Furthermore, the results of a controlled experiment raise the possibility that static coupling measurements may not always be adequate to account for variations in changeability (such as change effort) for object-oriented systems. According to follow-up research, when trying to understand object-oriented software, professional developers frequently trace the actual flow of messages that occur between objects at runtime. This is done in an effort to gain a better understanding of how object-oriented software works.

As a result of these findings, it appears as though dynamic coupling measures might be of some use such as factors that may be used to determine how cognitively difficult object-oriented software is. Last but not least, dynamic coupling is more accurate than static coupling for analyzing systems that contain dead code, which is code that is not being used and is therefore irrelevant to the analysis. This can seriously skew results. Some of these may be assessed within the framework of object-oriented designs, whereas others need a dynamic study of the code.

II. DYNAMIC COUPLING MEASUREMENT

Research on quality models in the context of object-oriented systems has mostly focused on creating structural metrics (such as class coupling) and examining their correlations with external quality criteria. Software architectures such as class fault-proneness and object-oriented systems fall under this category. The ultimate objective is to create predictive models that may be used in decision-making in some form, such as choosing which classes should undergo more thorough verification and validation. Most metrics have been established and gathered up to this point based on a static study of the design or code. Regardless of the structural characteristic being considered, this is accurate. They have proven time and time again to be reliable predictors of characteristics of external quality, such as fault-proneness, ripple effects from modifications, and changeability. However, the bulk of the studied systems only showed a little level of inheritance; hence, only a modest amount of polymorphism and dynamic binding was employed. We have seen that as the usage of object-oriented design and programming advances, inheritance and polymorphism are being employed more commonly in business. This is being done to facilitate easier system maintenance and increase the degree of internal reuse inside a system. Despite the fact that there hasn't been a comprehensive poll on the topic, this is clear from the rise in open-source projects, application frameworks, and library collections.

We shall first make a distinction between the several categories of dynamic coupling metrics. Then, building on this classification, we give both informal and formal explanations of the words, clarifying the underlying ideas through a real-world illustration. We shall then discuss the mathematical characteristics of the suggested measurements. We believe that all five of these features must be present for a coupling measure to be considered well-formed. Our policies were created with that goal in mind. To specify measurements in a form that is independent of the programming language being used, we use reference to a generic data model that has been created using a UML class diagram. The word "system" is often used to refer to the full constellation of hardware, software, and human processes that are being produced when discussing systems and software engineering. Instead, one calls the system's software components "software products."

A. *Classifying the Different Types of Coupling Measures*

Different definitions of dynamic coupling exist, and each definition may be justified in light of the application environment in which such metrics would be applied. The use of three decision criteria forms the basis for the formulation and categorization of dynamic coupling measurements.\

The Measurable Objects: Dynamic coupling may be assessed for a class as a whole or for a specific instance of a class because it depends on dynamic code analysis. Consequently, the thing being measured may be an object or a class.

Granularity: Independent of the object being measured, the measurement of dynamic coupling can be aggregated at a range of various degrees of granularity. It is possible to monitor dynamic object coupling at the object level, but it is also possible to aggregate measurements at the class level. The dynamic coupling of every instance of a class may therefore be combined into a single value. In reality, the class will probably be the lowest degree of granularity, even when assessing the object coupling. This is due to the fact that it is hard to envision any situation in which the coupling measurement of items would be useful. Aggregating all dynamic couplings of objects that are a component of an execution scenario is another option. Additionally, whole use cases (also known as sets of scenarios), complete use cases, and even an entire system (all objects of all use cases) can all have their dynamic object coupling measured. The aggregation scale will be different if the measurement object is a class since we can aggregate the dynamic class coupling over an inheritance structure, a subsystem, a collection of subsystems, or a whole system. The characteristics of the class being measured will determine how this works.

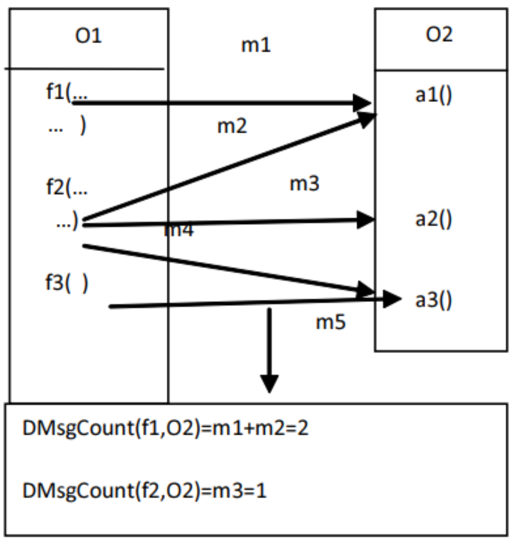
Scope: Another important factor that affects the precision of our estimations of the dynamic coupling between systems is the measurement range. This sets which objects or classes, depending on the entity being tested, are to be taken into consideration while measuring dynamic coupling. For instance, depending on the application environment, we might wish to omit library and framework classes in some situations..

III. COLLECTION OF DYNAMIC COUPLING DATA

It is crucial to collect data on dynamic coupling in a method that is both efficient and feasible. The two approaches to the problem are covered in this section. While the second technique calculates the measures based on dynamic UML models, the first method determines the measures by gathering the coupling data from running programmes.

A. *Tool for Gathering Dynamic Coupling Measures During the Running of the Simulation*

We developed the JDissect tool to get information on dynamic coupling from Java programmes. An overview of the architecture is shown in Figure 1 for your viewing enjoyment. The device comprises two stages specifically designated for the collecting and analysis of dynamic coupling data. The first phase entails gathering and storing data produced by an active Java software. This may be done by telling the Java Virtual Machine (JVM) to load the libjdissect.so library of data collecting tools. When specific internal events that have been described occur, these procedures are invoked. The interfaces used for communication between the library and the JVM (Java VM Debugging Interface) go by the titles Java Virtual Machine Profiling Interface (JVMPI) and Java Virtual Machine Data Interface (JVMDI). The great bulk of the data is gathered via the profiling interface that is connected to a method. As a result, the



Method Invocation syntax is connected to both the Class and the Method.

Fig 1: Dynamic Coupling between Two Objects

B. Data Collection Through the Utilization of UML Models

The operation was under the premise that the dynamic coupling data is compiled using the dynamic analysis of the code up to this point. It was also hypothesized that it could be feasible to get information on dynamic coupling by examining dynamic UML models, such as interaction diagrams. One of the proposals presented was this. The ability to use the data from such studies to make quick judgments makes the measurement of coupling on early design artifacts of practical significance. For instance, if the required UML diagrams for a certain design are available, one may create test cases based on the UML diagrams and calculate the dynamic coupling associated with each test case (use case scenario). This would necessitate making the supposition that the required UML diagrams are accessible. For instance, because it is believed that they may find more problems, test cases with a high dynamic coupling could be run first. As a consequence, the test plan would include a sequence for carrying out test cases based on knowledge of the components' dynamic interaction. Interaction diagrams represent the main difficulty when attempting to assess dynamic coupling based on UML models. This would necessitate making the supposition that the required UML diagrams are accessible. For instance, because it is believed that they may find more problems, test cases with a high dynamic coupling could be run first. As a consequence, the test plan would include a sequence for carrying out test cases based on knowledge of the components' dynamic interaction. Interaction diagrams represent the main difficulty when attempting to assess dynamic coupling based on UML models.

IV. THE CONCEPTUAL COUPLING METRIC

In order to complete many duties related to maintenance, developers are required to measure, either directly or indirectly, several aspects and evaluate the qualities of an evolving software system. Researchers have come up with a number of different suggestions for measurements that might help engineers obtain more comprehensive perspectives on the programme. Due to the fact that coupling has a direct impact on maintainability, it is one of the qualities that has the most sway over the maintenance process. The suggested coupling measures are used for a number of tasks, such as impact analysis, class fault-proneness evaluation, fault prediction, re-modularization, software component identification, design pattern evaluation, software quality evaluation, and other related tasks. The lowest possible degree of connection inside an OO system is one of the broad objectives of software designers. The classes in the system that are most likely to be impacted by changes and issues caused by other classes are those that are related tightly; as these classes frequently have a greater architectural importance, they should be taken into account. Coupling metrics are useful in these pursuits, and most of them are based on a type of dependency analysis that makes use of readily available design information or source code. As a result of the fact that many of these measures are founded on similar hypotheses and utilise similar data for calculation, there are more suggested coupling measures than dimensions that are represented by the measures. The measurements only capture a smaller number of dimensions.

Based on the semantic data shared by source code components, we introduced a new set of coupling metrics that formulates and captures additional dimensions of coupling, namely conceptual coupling. The notion that conceptual coupling represents a new dimension of coupling serves as the foundation for these new dimensions of coupling. Our measurements may be thought of as gauging how conceptually similar different method classes are to one another. The measurements are based on the use of information retrieval (IR) methodologies (i.e., through comments and identifiers) to express and examine the semantic information that is incorporated into software. The conceptual link may be used to supplement existing metrics, particularly in activities like impact analysis and change propagation, as current models do not fully account for the cascading consequences of changes made to existing software. They are directly applicable to different types of reverse engineering as well as re-modularization.

V. ADVANTAGES FOR USING DYNAMIC COUPLING OVER STATIC

- A brand new evaluation and analysis tool for Java and C++ software that is based on dynamic metrics and is named Dyna Metrics. This tool is able to evaluate and analyse all of the important dynamic metrics that have been known up to this point. In order to determine the value of each individual statistic, Dyna Metrics further evaluates each by contrasting it with its static equivalent.
- Their capacity to examine the behaviour of software while it is being executed, which is a skill that makes them far more valuable than their static counterparts. Suites developed by Yacoub et al. and Arisholm et al. are examples of some of the most recent dynamic metric suites to be released.
- Dynamic metric tools examine how a piece of software behaves while it is actively being used.
- Regardless of the structural property that is being evaluated at run time, it determines which classes should be subjected to a more rigorous verification and validation process.
- It is accurate predictors of external quality parameters such as the fault-proneness and changeability of the product.
- These dynamic coupling measures differ from one another in terms of the entities that they measure, as well as in terms of their scope and granularity, which both contribute to a rise in the usage of inheritance and polymorphism.

VI. CONCLUSION AND FUTURE WORK

A software performance measurement instrument based on metrics, like dyna metric. The tool is flexible enough to work with a number of static and dynamic metrics even if it is still in the early phases of development. To the best of our knowledge, no other tool has been able to manage the bulk of the dynamic metrics that are included in the product. The instrument's main support system is the Java Virtual Machine Tools Interface (JVMTI). The ability to process C++ is something we are currently working to add to the tool. Incorporating as many static and dynamic measures as is practically possible is Dyna Metrics' long-term goal. As a result, the tool will be able to establish an accurate set of high performance metrics for application software, which will eventually cause it to become an industry standard. As a result, the tool will be able to establish an accurate set of high performance metrics for application software, which will eventually cause it to become an industry standard.

The combined results show that dynamic export coupling metrics are important indicators of change proneness, even if this prior work was unable to compare with static coupling and size measurements.

REFERENCES

[1] Aine Mitchell and James F. Power, Run-time Cohesion Metrics: An Empirical Investigation. In Proceedings of International Conference on Software Engineering Research and Practice (SERP'04), Las Vegas, Nevada, (2004).

[2] Aine Mitchell and James F. Power, Toward a Definition of Run-time Object-oriented Metrics. In Proceedings of 7th ECOOP Workshop on Quantitative Approaches in ObjectOriented Software Engineering (QAOOSE'2003), Darmstadt, Germany, (2003).

[3] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite forObject-Oriented Design," IEEE Transactions on Software Engineering.Dynamic Coupling Measurement for Object-Oriented

[4] Dynamic Coupling Measurement for Object-Oriented Software. Erik Arisholm, Lionel C. Briand and Audun

Føyen.

[5] The Dynamic Function Coupling Metric and Its Use in Software Evolution. A´ rpa´d Besze´des, Tama´s Gergely, Szabolcs Farago´, Tibor Gyimo´thy and Ferenc Fischer University of Szeged, Department of Software Engineering.

[6] The Impact of Static-Dynamic Coupling on Remodularization, Rick Chern Kris De Volder, University of British Columbia, 2366 Main Mall, Vancouver, BC, Canada frchern, kdvolderg@cs.ubc.ca