Object-Oriented Static and Dynamic Software Metrics for Design and Complexity

Submitted in fulfillment of the requirement of the degree of

DOCTOR OF PHILOSOPHY

by

Varun Gupta
Reg. No. 2K06 NITK Ph.D 1129 CO

Under the supervision of

Dr. Jitender Kumar Chhabra
Associate Professor

Department of Computer Engineering
National Institute of Technology
Kurukshetra-136119 India

June 2010
I consider myself fortunate and privileged to have Dr. Jitender Kumar Chhabra as my supervisor. I am deeply indebted to him for shaping my path to research by guiding me with his extensive knowledge and his insightful discussions. I thank him for his constant support and encouragement throughout the course of this research work.

I would like to thank Departmental Research Committee members Dr. Mayank Dave (HOD), Dr. A. K. Singh, and Dr. S. K. Jain who gave me suggestions on improving this work.

The research reported in this thesis has benefited from anonymous reviewers. I would like to thank all those reviewers, who through their incisive and useful critic have contributed significantly to improving the submitted papers and consequently the work presented here.

I thank all others who have contributed to my work with material, data, knowledge, viewpoints, advice, comments, suggestions, ideas and other things. I would like to thank my colleagues and friends for inspiring me to focus on my research work.

Lastly and by no means least, I thank my family for their support, encouragement, and understanding during the research work.

VARUN GUPTA
LIST OF PUBLICATIONS OUT OF THE THESIS

Papers Published/Accepted in International Journals

5. “A Survey of Dynamic Software Metrics”, Accepted for publication in Journal of Computer Science and Technology (Springer).

Papers Published in International Conferences


Papers Communicated in International Journals


Paper to be communicated in International Journal

SYNOPSIS

Software metrics are used to measure software engineering products (design, source code etc.), processes (analysis, design, coding, testing etc.) and professionals (efficiency or productivity of an individual designer). If used properly, software engineering metrics allow us to quantitatively define the degree of success or failure of a product, process, or person. These can also be used to take meaningful and useful managerial and technical decisions related to cost, effort, time, quality etc. Thus, incorporating metrics into software development process is a valuable step towards creating better systems. The most common approach used for software development is the object-oriented approach and mainly two kinds of software metrics exist for object-oriented software - static software metrics and dynamic software metrics. The static software metrics are obtained from static analysis of the software, whereas dynamic software metrics are computed on the basis of data collected during execution of the software. This thesis presents some new static and dynamic software metrics for the measurement of design and complexity of object-oriented software.

The research work reported in this thesis is grouped into three categories:- (i) package level metrics, (ii) dynamic metrics, and (iii) cognitive complexity metrics.

Packages are re-usable components for modern object-oriented systems. A package usually consists of classes, interfaces and sub-packages e.g. Java packages. To promote reuse in object-oriented systems and to make deployment and maintenance tasks easy, packages in object-oriented systems should follow the basic principles of design i.e. maximum cohesion and minimum coupling. Very few metrics exist in literature to measure these design characteristics at package level and are primitive in nature. In the thesis, new metrics for the measurement of package cohesion and package coupling are proposed.

The cohesion of a package can be measured in terms of the degree of intra-package dependencies among its elements (classes, interfaces and sub-packages) present at the same hierarchical level. An attempt has been made in the thesis to measure cohesion of packages in object-oriented software on the basis of relations present between the pairs of package elements. The cohesion of a package is measured as the actual number of relations between all ordered and unique pairs of
the package elements, divided by the maximum number of relations possible between ordered pairs of the package elements. The reason of considering ordered pairs is to take care of the direction of relations between package elements. If, there are n elements in a package, then each element of the package may be at the most connected to all other n-1 elements of the package. Thus the maximum possible number of relations among n elements is n*(n-1), which represents the count of maximum possible intra-package dependency for a package. The ratio of the actual number of relations and the maximum possible number of relations provides a normalized value for the proposed measure. The hierarchical structure of packages has also been taken into account during the measurement of package cohesion as the dependency between a class element and a subpackage element of a package is defined in terms of relations between the class element and classes present in the subpackage element of the package at the subsequent hierarchical levels. The proposed cohesion metric has been firstly validated theoretically using standard axiomatic framework given by Briand et al. and then empirically using twenty-five packages taken from six open source software projects developed in Java. The computed values of the proposed cohesion metric for these packages have been found to be strongly correlated with external quality factors such as reusability of the packages.

The coupling between packages is the degree of interdependence between them. Theoretically every package is a stand-alone unit, but in reality packages may depend on each other as either they require services from other packages or provide services to other packages. Thus, coupling between packages cannot be completely avoided but can only be controlled. Coupling between packages is an important factor that affects external quality attributes of software e.g. understandability, maintainability, reusability etc. But still, only a few quantitative studies of the actual use of coupling have been conducted at the package level. In this thesis, new metrics are proposed for the measurement of package level coupling based on formal definitions and properties of the packages in order to achieve good quality software systems. The coupling between two packages is defined as the total number of directed connections between all pairs of their elements, which can be classes or sub-packages. The total coupling of a package at a hierarchical level can be defined as the summation of couplings of the package with all other packages present at the same hierarchical level. The proposed package coupling metrics have been validated
theoretically using standard theoretical framework given by Briand et al. for coupling measures. Then, these metrics have been validated empirically using eighteen packages taken from two open source software systems. The results obtained from this study show strong correlation between package coupling and the effort required to understand a package, which suggest that the proposed package coupling metrics could be used to predict external software quality factors such as understandability.

The conventional static metrics have been found to be inadequate for modern object-oriented software due to improper/insufficient contribution of object-oriented features such as polymorphism, dynamic binding, inheritance and unused code towards their measurement. This fact motivates to focus on dynamic metrics in addition to the traditional static metrics. In this thesis, a few new dynamic metrics are proposed for the measurement of cohesion and coupling at run-time. Moreover, different approaches for the dynamic analysis of programs required for collection of run-time data for the measurement of dynamic metrics are compared. Dynamic analysis of software can be performed in many ways: - using profilers, from dynamic models and using aspect-oriented programming (AOP). Some other less popular techniques like AST rewriting based approach, pre-processor based approach, method-wrappers based approach and hybrid approach can also be used for this purpose. From this study, it is found that AOP approach provides a balanced method for the dynamic analysis of programs. In addition, this approach is easier to implement and at the same time an efficient technique for dynamic analysis without any side effects. In the thesis, AOP approach has been used for the computation of dynamic cohesion and coupling metrics.

Most of the class cohesion measures proposed in literature are static in nature and very few attempts have been made to measure cohesion of a class at run-time. Moreover, the already existing run-time cohesion metrics are nothing but direct extensions of the existing static cohesion metrics. In the thesis, new measures have been defined for dynamic cohesion measurement for objects as well as classes and the proposed dynamic cohesion measures take into consideration the key features of object-orientation like inheritance, polymorphism and dynamic binding. The necessity of dynamic metrics for these features can be understood from the fact that the actual polymorphic invocations can only be determined at run-time. The inclusion of effect of these features in computation of dynamic cohesion makes them more accurate than
the existing static cohesion measures. The method to measure dynamic cohesion is to instrument the source code to log all occurrences of interactions among object-members at run-time, while the application is being executed. The dynamic cohesion of an object is measured on the basis of four types of relationships among elements of an object: - (a) the write dependence of attributes on methods, (b) the read dependence of methods on attributes, (c) the call dependence between methods, and (d) the reference dependence between attributes. In the definitions of the measures, the elements (i.e. attributes and methods) as well as the relationships among the elements of a class are precisely defined. The methods such as constructors, access methods, delegation methods and destructors, which do not contribute to the cohesiveness of objects and classes, are excluded from the cohesion measurement. The dynamic cohesion has been measured first at the object level and then at the class level. Class level cohesion is obtained by aggregating cohesion values of all objects belonging to that class. The proposed measures have been demonstrated to satisfy the four cohesion properties defined by Briand et al. A dynamic analyzer tool has been developed using aspect-oriented programming (AspectJ) to perform dynamic analysis of Java applications for the purpose of collecting run-time data needed for the proposed dynamic cohesion metrics. Using this tool, the empirical values of the proposed metrics have been gathered for source code APIs of Java Development Kit (JDK) and their correlation with the change-proneness of the classes has been investigated in the thesis over different versions of JDK. The proposed metrics are found to be better indicators of change-proneness of classes than the existing cohesion metrics. Moreover, the proposed dynamic cohesion measures are more accurate than their static counterparts as they are defined at run-time and take into consideration the actual interactions taking place rather than the potential interactions. The working of the metrics is such that the scope of the measurement can be limited to a single object or even a scenario. This precision of measurement might be useful in testing and impact analysis whereas other existing cohesion metrics cannot provide such usage, as they measure cohesion up to the class level only.

Just like cohesion, dynamic coupling measurement has also caught little attention of researchers. Existing dynamic coupling metrics take into account only method-method invocations between objects of classes at run-time. In this thesis, new dynamic coupling metrics have been proposed which take into account all major types of relations: - (a) run-time aggregation relations, (b) run-time inheritance relations, (c)
run-time method-attribute reference relations, and (d) run-time method-method invocation relations, between the objects of different classes during measurement. A dynamic coupling tracer application has been developed in Aspectj for the purpose of computation of the proposed metrics. With the help of this tracer application, the proposed metrics have been validated empirically using six open source projects developed in Java. In the empirical study, it has been found that average dynamic coupling per class (CDC/NOC) is a useful indicator of the degree of dynamic coupling for a project. The study has further shown that there is no correlation between the number of classes (NOC) and amount of dynamic coupling (CDC) in a project. Thus, number of classes in a project cannot predict the amount of dynamic coupling in a project.

In addition to coupling and cohesion, another important dimension of software quality is its complexity. The software can have better testability, readability and maintainability if it possesses low coupling, high cohesion and less complexity. There are many facets of complexity. Some target the control flow complexity, few concentrate on operators and operands and others focus on measuring comprehension efforts on the basis of spatial and cognitive complexity. Spatial complexity metrics indicate the difficulty of understanding the logic of the program in terms of lines of code that the reader is required to traverse to follow control or data dependencies as they build a mental model. Spatial complexity of object-oriented software is the combination of class spatial complexity and object spatial complexity. The spatial complexity metrics are more difficult to trust and validate due to less-understood effect of human factors and the human mind's working towards comprehension of the programs. In order to improve confidence in some of such metrics, there are certain necessary properties and evaluation criteria widely used by many researchers that software complexity metrics should satisfy. In the thesis, Weyuker’s properties and Briand et al. criteria are used for the evaluation of the object-oriented spatial complexity metrics. These metrics are found to satisfy all nine properties given by Weyuker and all five complexity properties required by Briand et al. evaluation criteria. The object-oriented spatial complexity measures proposed in literature were formulated by keeping C++ language in mind, and there were no spatial complexity measures available for the Java language. Keeping in view the increasing popularity of Java, this thesis attempts to define the spatial complexity measures for Java
programs. Although both C++ and Java are object-oriented languages, still in many aspects Java programs completely differ from C++ programs e.g. way of defining a method. The spatial metrics proposed for Java in the thesis have been formulated in a way so as to handle all such differences. As programming in Java revolves around definitions of classes and objects, comprehension of a Java program requires understanding of Java classes as well as objects. Thus, spatial complexity of Java code is unification of spatial complexities of classes and objects. In the thesis, two categories of spatial complexity measures for Java programs are proposed. These metrics are termed as Spatial Complexity of Java Class (SCJC) and Spatial Complexity of Java Object (SCJO). These metrics have been computed for ten different projects developed using Java. The results indicate that the proposed spatial complexity metrics influence the comprehension process of Java programs to a great extent and understandability of Java programs does not depend on their size only but also on the values of SCJC and SCJO. Lower values of Spatial Complexity of Java Class (SCJC) and Spatial Complexity of Java Object (SCJO) are desirable for better understandability of Java programs.

The understandability of object-oriented software has been reported in literature to be dependent on architectural aspects as well along with spatial distances. The architectural aspect of software has been reflected in cognitive complexity with the help of using weights of various types of Basic Control Structures (BCS) present in the source code. The measures of cognitive complexity proposed by various authors considered only these weights, which were reflection of architectural viewpoint only and did not look into the spatial aspect at all. On the other hand, the importance of spatial distance towards complexity has already been established in literature. Thus there is need to combine the impact of architectural as well as spatial aspects of the software to compute the cognitive complexity. This type of software complexity has been termed as cognitive-spatial complexity of the software in the thesis and new cognitive-spatial complexity measures are proposed for object-oriented software. The proposed measures take spatial as well as architectural complexity of the software into account for estimation of the cognitive effort required for the software comprehension process. The spatial complexity has been taken into consideration using the lexical distances (in LOC) between different program elements and architectural complexity of the software has been taken into account using the cognitive weights of various control structures present in the control flow of
the program. The proposed measures have been evaluated against the standard axiomatic frameworks given by Weyuker and Briand et al. to prove their usefulness. Further, the proposed measures have been compared with the existing cognitive and spatial complexity measures for object-oriented software. This comparative study has shown that the proposed measures are better indicators of the cognitive effort required for program comprehension than the corresponding existing cognitive and spatial complexity measures.
CONCLUSIONS AND FUTURE SCOPE

Conclusions

In this thesis, new static and dynamic metrics have been proposed, evaluated and validated for measurement of cohesion, coupling and complexity for object-oriented software. Major contributions of the work reported in this thesis are:

- New metrics for the measurement of package cohesion and package coupling have been proposed and validated.
- Aspect-oriented approach of collection of run-time data has been found to be better than other approaches for this purpose.
- New dynamic cohesion and coupling metrics have been proposed and validated.
- A dynamic analyzer tool has been developed using aspect-oriented programming (Aspectj) to perform dynamic analysis of Java applications.
- New spatial complexity measures for the estimation of comprehensibility of Java programs have been defined.
- New object-oriented cognitive-spatial complexity metrics have been proposed and evaluated.

Future Scope

During working in this area of research, a lot of scope for future work has been observed. There is need of further empirical investigations of the proposed package level metrics in order to establish their relations with other external software quality factors such as maintainability. As dynamic metrics have the advantage of being more precise, but they are difficult to compute in comparison to static ones. Thus, there is clear opportunity for researchers in hybrid approach where the dynamic analysis results can be augmented by static information for collection of metrics data. This hybrid approach can combine static as well as dynamic approaches to make the measurement process more cost-effective. In future work in the area of dynamic cohesion metrics, relationships between the proposed dynamic cohesion metrics and other quality attributes such as fault-proneness could be explored. Moreover, in
future, an alternative approach for selection of the values of the weights assigned to
different relations could be explored to use the dynamic cohesion metrics in predictive
models and estimate coefficients of the predictive models, which could be used as
weights. Further, the impact of the dynamic coupling metrics on the external attributes
of software applications needs to be explored.

The cognitive complexity measures proposed in the thesis can be extended to take
into account the semantic aspect of the code. There is also an opportunity to study and
integrate the impact of special features such as exception handling and dynamic
polymorphism etc. into the cognitive metrics. Even the possibility of proposing
dynamic cognitive metrics can also be explored.
REFERENCES


41. CaffeineMark <http://www.benchmarkhq.ru/cm30/info.html>


63. Element Construction Set (ECS) <http://jakarta.apache.org/ecs/index.html>


96. JPDA <http://java.sun.com/products/jpda>
97. JUnit <http://sourceforge.net/projects/junit/>


171. Tomcat Jasper <http://jakarta.apache.org/tomcat>
172. TulipChain <http://ostermiller.org/tulipchain>
173. Velocity <http://jakarta.apache.org/velocity>
182. XGen Source Code Generator <http://sourceforge.net/projects/xgen/>


188. X. Yang, Research on Class Cohesion Measures, M.S. Thesis, Department of Computer Science & Engineering, Southeast University, April 2002.


