Correlating Dimensions of Inheritance Hierarchy with Complexity & Reuse

Nasib S. Gill
Professor & Head, Department of Computer Science & Applications
Maharshi Dayanand University (India)
nasibsgill@gmail.com

Sunil Sikka
Research Scholar, Department of Computer Science & Applications
Maharshi Dayanand University (India)
sunil.sikka@yahoo.com

***Abstract*— Inheritance is the vital feature of any object oriented software which provides reuse of exiting classes for designing new classes. Higher reuse provides higher productivity and greater quality. Inheritance hierarchy is one of the very important artifacts targeted for measurement of reuse and reusability. Reuse through inheritance hierarchy can be measured from two dimensions- Depth and Breadth. Higher depth and breadth may increase complexity of software which makes the software difficult to understand and maintain. This paper aimed to correlate the depth and breadth of inheritance hierarchy with reuse and complexity of inheritance hierarchy using design oriented metrics.**

Keywords- Object Oriented Metrics, Reuse, Inheritance and Complexity

I. INTRODUCTION

Extensive use of object oriented paradigm in software development is due to its ability of developing more maintainable and reusable products. The major feature of object oriented software development is development with reuse. Large and complex software can be developed easily and quickly using object oriented paradigm as compare to traditional procedure oriented paradigm. Inheritance is the major feature of object oriented languages to implement reuse. Most of the coding in object oriented software is involved in form of inheritance. At design level reuse is modeled by inheritance hierarchy/tree. During designing of object oriented software lots of efforts are made to adjust inheritance hierarchy and relationships between classes to increase the reuse and reusability. Inheritance hierarchy is the major design artifact used for measuring the reuse and reusability in object oriented systems. It provides early opportunity of measuring reuse and reusability during software development. It can be measured in two dimensions i.e. depth and breadth. Chidamber and Kemerer proposed metrics[1]-Depth of Inheritance Tree (DIT) and Number of Children (NOC) for measuring depth and breadth of inheritance hierarchy. DIT of an inheritance hierarchy is the maximum path length from a class to root node or DIT of a class is number of ancestor class that can affect that class[2]. NOC of a class in inheritance hierarchy is the number of immediate sub classes of that class. Since the proposal of DIT and NOC, these have been widely validated by many researchers. It has been validated in literature that DIT and NOC are correlated with reuse, complexity, maintainability, fault-proneness and change proneness [3]-[9]. Authors also claim that deeper tree constitute higher complexity due to involvement of more methods and classes and greater breadth is indicator of improper sub-classing and more testing efforts[1]. Higher reuse may affect complexity of software in terms of difficulty in understanding the classes. Higher reuse is always desirable where as higher complexity is always undesirable for good quality software. DIT and NOC both measures the same feature of object-oriented software but from two different perspectives. DIT and NOC do not measures the amount of reuse but only provides the depth and breadth of inheritance hierarchy respectively. Various other metrics such as Method Inheritance Factor (MIF) and Attribute Inheritance Factor (AIF) proposed by Abreu et al.[10] are available which are based on actual amount of method and attribute reuse. The feedbacks of these metrics are very useful for improving the software design.

Daly et al. [5] evaluates the effect of DIT on program maintenance and concludes that software having DIT equals to 3 are quicker to maintain as compare to software having no inheritance. Maintenance of software having DIT equals to 5 is slightly slower as compare to software having no inheritance. Unger et al.[4] conducts the similar study with more complex and large software. Results suggest that higher DIT may complicate program understanding but DIT is not only factor for that, many other factors like complexity of program and type of maintenance tasks also affect on maintenance time. Authors found high correlation between maintenance time and the number of methods trace to gain program understanding. Gatrell et al.[3] conducts a empirical study using

C# programs and concludes that class within specific range of DIT are more prone to changes and faults. Classes within specific range of NOC are more prone to changes while classes with a higher NOC are more fault-prone than those classes in a simpler inheritance hierarchy.

Objective of this paper is to correlate the DIT and NOC with complexity and reuse. To measure method reuse and attribute reuse MIF and AIF metrics are used respectively. To measure complexity Response For a Class (RFC) is used.

Rest of the paper is organized into three sections. Section II discusses the metrics and hypothesis used. In section III metrics are applied on various programs written in Java language having DIT and NOC varies from 0 to 6 and results are presented in tabular form. Various observations from result are also discussed in section III. Finally conclusion of paper and future directions are given in section IV.

II. METRICS AND HYPOTHESIS

Five design oriented metrics are used in this study. To measure the dimension of inheritance hierarchy DIT and NOC metrics are used. To measure method reuse MIF metric is used which can be calculated as follows

MIF = (total number of methods inherited) / (total number of methods declared and inherited)

To measure attribute reuse AIF metric is used which can be calculated as follows

AIF = (total number of attributes inherited) / (total number of attributes declared and inherited)

For measuring complexity of a class many metrics are available which measures the complexity from different perspectives such as size, relations, method complexity etc. Complex class is difficult to understand, test and maintain. This paper uses the Response For a Class (RFC) metric to compute the complexity of a class. RFC of a class is number of methods that can be potentially executed in response to a message received by an object of that class[1]. Higher RFC indicates testing and debugging of the class becomes more complicated since it requires a greater level of understanding on the part of the tester[1]. Therefore, higher RFC is analogs to higher complexity of the class. Originally RFC considers only methods of class and methods that can be directly invoked by methods of class. However this paper considers methods that can also be indirectly called by methods of class while computing RFC of a class which is denoted by RFC'. Different classes can have different RFC' in inheritance hierarchy, in this paper maximum of RFC' is considered as RFC' of whole inheritance hierarchy.

Following four hypothesis (H1-H4) are considered

H1: An inheritance hierarchy having more DIT than its peers is more complex as compared to them.

H2: An inheritance hierarchy having more DIT than its peers provides more method and attribute reuse as compare to them.

H3: An inheritance hierarchy having more NOC than its peers is more complex as compared to them.

H4: An Inheritance hierarchy having more NOC than its peers provides more method and attribute reuse as compare to them.

III. EXPERIMENTAL RESULTS

Metrics are applied on two kinds of programs written in Java language. First category of program includes programs having maximum NOC=1 but DIT varies from 0 to 6. Second category of program includes programs having maximum DIT =1 but NOC varies from 0 to 6. For each program value of maximum RFC', MIF and AIF is computed. Results of metrics are presented in Table I to Table IV.

TABLE I. RESULTS OF METRICS AT MAXIMUM NOC=1

Metric	DIT						
	0	***1***	***2***	***3***	***4***	***5***	***6***
Max(RFC')	2	5	7	9	9	12	17
MIF	0	0.29	0.36	0.44	0.57	0.6	0.22
AIF	0	0.14	0.4	0.21	0	0.25	0.25

TABLE II. RESULTS OF METRICS AT MAXIMUM DIT=1

Metric	NOC						
	0	***1***	***2***	***3***	***4***	***5***	***6***
Max(RFC')	2	5	5	4	4	3	4
MIF	0	0.29	0.36	0.43	0.53	0.59	0.61
AIF	0	0.14	0.22	0.25	0.44	.45	0.59

TABLE III. CORRELATION OF DIT WITH REUSE AND COMPLEXITY METRICS

Metric	Correlation Coefficient
RFC'	0.97
MIF	0.55
AIF	0.30

TABLE IV. CORRELATION OF NOC WITH REUSE AND COMPLEXITY METRICS

Metric	Correlation Coefficient
RFC'	0.07
MIF	0.94
AIF	0.98

Following observations are made from results obtained.

- The value of Max(RFC') is increasing with the increase in DIT but value of Max(RFC') may increase or decrease with the increase of NOC. Therefore increase in DIT makes the inheritance hierarchy complicated but it is not same for NOC.
- Increase in DIT increases the value of MIF. This indicates higher value of DIT means higher method reuse. Increase in NOC also increases the value of MIF which indicates higher NOC cause higher method reuse.
- Increase in DIT need not necessary increase the value of AIF. With the increase of DIT value of AIF may increase or decrease. But results show that higher value of NOC yields higher AIF.
- In most of the cases MIF is higher than AIF because generally methods are freer to access as compare to attributes. Attributes are generally restrictive to access.

Correlation of DIT with Complexity

Correlation coefficient (0.97) shows that DIT is strongly positively correlated with maximum RFC' i.e. higher DIT causes higher complexity. Therefore hypothesis H1 is accepted.

Correlation of DIT with Method and Attribute Reuse

Correlation coefficient between DIT and MIF (0.55) and correlation coefficient between DIT and AIF (0.22) shows that increase in DIT also increase method and attribute reuse. Therefore hypothesis H2 is also accepted. However DIT is not strongly correlated with AIF.

Correlation of NOC with Complexity

Correlation coefficient (0.07) shows that NOC is very poorly correlated with RFC'. It shows higher NOC does not mean higher complexity. Therefore hypothesis H3 is rejected.

Correlation of NOC with Method and Attribute Reuse

Correlation coefficient between NOC and MIF (0.94) and correlation coefficient between NOC and AIF (0.98) shows that NOC is strongly positively correlated with method and attribute reuse. Higher NOC means higher method and attribute reuse. Therefore hypothesis H4 is accepted.

IV. CONCLUSION AND FUTURE WORK

This paper correlates the DIT and NOC of inheritance hierarchy with reuse and complexity. Four hypotheses (H1-H4) were derived before collecting the data. On the basis of results obtained hypothesis H3 is rejected and all other hypothesis are accepted. Higher DIT is good indicator of higher complexity where as higher NOC is good indicator of higher reuse. The limitation of this study is that data used to generalize the result is small and derived from academic level programs. Further validations are needed using large industrial projects. However obtained results are quite encouraging and provide guidance for future research on the impact of DIT and NOC on complexity and reuse.

REFERENCES

[1] S.R. Chidamber, C.F. Kemerer "A Metrics Suite for Object Oriented Design" IEEE Transactions on Software Engineering, Vol. 20, No.6, pp. 476-492, 1994.

[2] Frederick T.Sheldon, Kshamta Jerath and Hong Chung "Metrics for Maintainability of Class Inheritance Hierarchies", Journal of Software Maintenance and Evolution: Research and Practice, Vol. 14, pp. 1-14, 2002.

[3] M. Gatrell and S. Counsell. "Size, Inheritance, Change and Fault-Proneness in C# Software",Journal of Object Technology, Vol. 9, No. 5, pp. 29-54, 2010.

[4] Barbara Unger, Lutz Prechelt "The Impact of Inheritance Depth on Maintenance Tasks – Detailed Description and Evaluation of Two Experiment Replications" , Technical Report July 1998.

[5] Daly, J., Brooks, A., Miller, J., Roper, M. and Wood, M. "Evaluating Inheritance Depth on the Maintainability of Object-Oriented Software" Empirical Software Engineering, Vol. 1, No. 2, pp. 109-132, 1996.

[6] K.K Aggarwal, Yogesh Singh, Arvinder Kaur, Ruchika Malhotra, "Investigating effect of Design Metrics on Fault Proneness in Object-Oriented Systems", Journal of Object Technology, Vol. 6, No. 10, pp. 127-141, November-December 2007.

[7] Lionel C. Briand, , Christian Bunse, and John W. Daly "A Controlled Experiment for Evaluating Quality Guidelines on the Maintainability of Object-Oriented Designs" IEEE Transactions on Software Engineering, Vol. 27, No. 6, June 2001.

[8] Lionel C. Briand, Walcelio L. Melo, Jurgen Wust "Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects" ISERN Report No. ISERN-00-06.

[9] Lionel C. Briand , Jurgen Wust, John W. Daly, D. Victor Porter "Exploring the Relationships Between Design Measures and Software Quality in Object-Oriented Systems" Journal of Systems and Software, Volume 51 Issue 3, pp. 245-273, 2000.

[10] Abreu, F. Brito and Carapuca R. "Object-Oriented Software Engineering: Measuring and Controlling the Development Process", Proceedings of the 4th International Conference on Software Quality, ASQC, McLean, VA, USA, October 1994.

AUTHORS PROFILE

Dr. Nasib S. Gill is currently working as Professor & Head, Department of Computer Science & Applications, Maharshi Dayanand University, Rohtak, Haryana (India). He has more than 20 years of experience in teaching and research. He is the recipient of Commonwealth Fellowship Award availed at Brunel University, West London (United Kingdom) for the year 2001-2002. His major current research interests include designing and development of Component-Based Metrics and Software Complexity Metrics.

Sunil Sikka is a research scholar with the Department of Computer Science & Applications, Maharshi Dayanand University, Rohtak, Haryana (India). His area of research is Object Oriented Software Measurement. His other area of interest includes Software Engineering, Artificial Intelligence and Object Oriented Programming.