ClassIN: A Class Inheritance Metric Tool

B. Ramachandra Reddy and Aparajita Ojha

PDPM Indian Institute of Information Technology, Design and Manufacturing Jabalpur, Dumna Airport Road, P.O. Khamaria, Jabalpur, 482005, India
{brreddy,aojha}@iiitdmj.ac.in

Abstract. This paper presents a comprehensive class inheritance metrics tool called ClassIN. The tool works for Java projects and presents an analysis with twenty different inheritance metrics including metrics at class level as well as at class hierarchy level. This also helps in identifying class hierarchies that may be more complex from the point of view of software maintenance. Graphical visualization of three important metrics, namely AID, specialization ratio and reuse ratio are also provided for an insight on structure of class hierarchies of a given Java project. This would help developers in identifying classes that may be more prone to faults or high maintenance costs.

Keywords: Class Inheritance Hierarchies, Inheritance Metrics, Software Maintainability.

1 Introduction

Inheritance patterns in object oriented software systems greatly affect the overall performance and maintainability of systems. Numerous metrics have been proposed by researchers and developers that measure different aspects of inheritance present in a software. Depth of inheritance (DIT), number of children (NOC) [1], average inheritance depth (AID), specialization ratio (SR) and reuse ratio (R) [2] metrics are some of the most important metrics in quantifying the effect of inheritance. Use of inheritance metrics helps in identifying possible attributes that may reduce maintenance efforts and enhance the reliability, Maintainability [27],[29],[30]and Reusability [3], Fault prediction, Defect Prediction [4],[5],[6],[28],Testability [7],[8].

This paper presents a comprehensive class inheritance metrics tool called ClassIN. The tool works for Java projects and presents an analysis with twenty different inheritance metrics including metrics at class level as well as at class hierarchy level. This also helps in identifying class hierarchies that may be more complex from the point of view of software maintenance. The main features of ClassIN are as follows –

- It provides inheritance metrics of the project at class level and also at class hierarchy level.
- It provides some insight into the depth and breadth of class hierarchies.
- It exports the values of inheritance metrics of a Java project into a spreadsheet (Excel-sheet) for analysis of results.

Á. Rocha et al. (eds.), *New Perspectives in Information Systems and Technologies, Volume 2*, Advances in Intelligent Systems and Computing 276,
DOI: 10.1007/978-3-319-05948-8_11, © Springer International Publishing Switzerland 2014

- It displays all the class inheritance hierarchies of the project.
- Provides a 3D visualization of three important class hierarchy metrics for a Java project, namely AID, Specialization ratio and Reuse Ratio.
- Provides an insight into the maintainability (Modifiability and Understandability) of class hierarchies.

A comparative analysis of the proposed tool with the existing tools is also presented here. Our tool is available online with a demonstration.

2 Inheritance Metrics

Inheritance metrics are used to measure the depth, width and relative inheritance values reflecting the inheritance patterns in an object oriented system. Inheritance metrics are broadly classified into two types - class level inheritance metrics and class hierarchy metrics. The class level inheritance metrics represent the inheritance values of individual classes, whereas the class hierarchy metrics represent inheritance hierarchal structures of the related classes. Table 1 lists metrics that are commonly used for determining class level inheritance and class hierarchy level inheritance. In addition to inheritance metrics the ClassIN tool also provides values of maintainability metrics such as Average Modifiability (AM) and average understandability (AU) of class hierarchies [3].

Table 1. Inheritance Metrics

Class level metrics	Class hierarchy metrics
Depth of Inheritance (DIT) [1]	Maximum DIT (MaxDIT) [11]
Number of Children(NOC) [1]	Average Inheritance Depth (AID) [2]
Total Progeny count (TPC) [10]	Number of children for a component (NOCC) [11]
Total Ascendancy count (TAC) [10]	Total length of inheritance chain (TLI) [10]
Class-to-leaf depth (CLD) [13]	Specialization Ratio (S) [2]
Number of Ancestor classes (NAC) [14]	Reuse Ratio (U) [2]
Number of Descendent classes (NDC) [14]	Attribute Inheritance Factor (AIF) [9]
Number of Overridden Methods (NORM) [12]	Method Inheritance Factor (MIF) [9]
Number of Attributes Inherited(NAI) [12]	Specialization Index(SIX) [12]
Number of Methods Inherited(NMI) [12]	
Coupling Through Inheritance(CTI) [15]	

3 Analysis of Software Metrics Tools

Several commercial as well as open-source OO metric tools exist today. We have analyzed CKJM [16], Analyst4J [17], Eclipse plug-in 1.3.6 [18], JMT [19], VizzAnalyzer [20], Dependency Finder [21], OOMeter [22], SD metrics [23]. Table 2 shows comparative analysis of various existing tools.

Table 2. Tools and inheritance metrics used in evaluation

Metrics	TOOLS							
	CKJM	**Analyst4J**	**Eclipse plug-in 1.3.6**	**JMT**	**Vizz-Analyzer**	**Dependency Finder**	**OO-Meter**	**SD Metrics**
DIT	√	√	√	√	√	√	√	√
NOC	√	√	√	√	√	√	√	√
AID	×	√	√	×	×	×	×	×
AIF	×	×	×	√	×	×	×	×
MIF	×	×	×	√	×	×	×	×
MaxDIT	×	√	√	×	×	×	×	×
CLD	×	×	×	×	×	×	×	×
TLI	×	×	×	×	×	×	×	√
TPC	×	×	×	×	×	×	×	√
TAC	×	×	×	×	×	×	×	√
NORM	×	×	√	×	×	×	×	×
NAI	×	×	×	√	×	×	×	√
NMI	×	×	×	√	×	×	×	√
SIX	×	×	√	×	×	×	×	×
NOCC	×	×	×	×	×	×	×	×
S	×	×	×	×	×	×	×	×
U	×	×	×	×	×	×	×	×
CTI	×	×	×	×	×	×	×	×
Class hierarchies	×	×	×	×	×	×	×	×

The above table gives a clear picture of various metrics covered in some of the standard tools. One can see that DIT and NOC are covered by all the tools. Further, JMT tool also provides AIF, MIF metrics at project level whereas Analyst4J, Eclipse plug-in cover AID, MaxDIT metrics also. Whereas Analyst4J provides metrics at project level, Eclipse plug-in provides metrics at package level. SD Metrics covers maximum number of class-level metrics whereas JMT provides a mix of class level as well as class hierarchy level metrics. So there is no tool covering metrics at class hierarchy level. A comprehensive study of metrics at class hierarchy level helps determine factors that would result in better maintainability and reusability of the software. In view of this, we have developed a comprehensive tool ClassIN that covers all the metrics listed in Table 1 along with two more metrics useful to predict the maintainability through modifiability and understandability [3].

In general, designers prefer to keep the depth of inheritance low in class hierarchies, in order to improve their understandability and reusability [25]. ClassIN tool is useful for measuring depth and breadth of the class inheritance hierarchies. As

it is well known, DIT, NOC metrics are two most useful measures for prediction of fault proneness and software reusability [26]. As reported in [24], higher DIT value indicates higher maintenance cost [24]. Especially for DIT values ≥ 5, software becomes highly complex from maintenance point of view. Prechelt et al. [26] have also analyzed maintainability with respect to DIT values and have concluded that modules with low inheritance depth are easier to maintain. The DIT metric gives individual class depth in the hierarchy. The NAI, NMI, DIT, NOC metrics are useful for finding the number of test cases required for determining the correctness of a software system [7], [8]. Harrison et al. [9] had concluded that MOOD metrics such as AIF, MIF provide an overall quality assessment of systems. In addition to AIF and MIF metrics, ClassIN provides two more measures, namely Specialization ratio (S) and Reuse ratio (U) to help developers assess more effectively a given software project with respect to reusability and testability. In essence, ClassIN tool provides a variety of metrics that may be used for different purposes. Metrics may be selectively used for analyzing the software performance, fault-proneness, reusability and maintainability.

4 ClassIN Tool

In this section, we shall describe the basic functionalities of ClassIN applied to a usage scenario. Initially ClassIN takes a Java project as input and after analyzing various metrics it displays the highest value of metrics in a mainframe window. The tool also generates all the class hierarchies available in the project. The tool helps in identifying various attributes of a software project. These attributes in turn help in determining if a given project needs a design review. Figure 1 displays a snapshot of ClassIN showing the results for system level inheritance metrics. After generating the metric data, the tool exports all the metrics in an Excel sheet. Next, the tool displays all the class hierarchies in the project. A snapshot of hierarchies displayed by the tool is presented in Figure 2.

In order to show the functionality of the tool and its usefulness, we considered the problem of lack of discrimination in class hierarchies. As shown in Table 1, numerous metrics have been proposed by researchers to depict and analyze the inheritance structure in class hierarchies. However, there is no standard set of metrics that helps

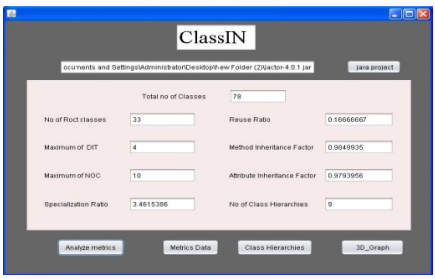

Fig. 1. Snapshot of ClassIN tool

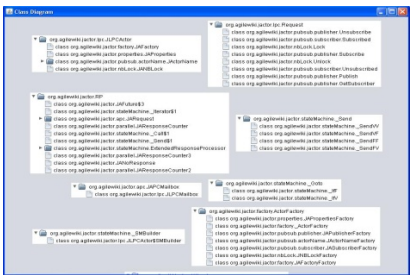

Fig. 2. Snapshot of class hierarchies

in distinguishing between inheritance patterns. In fact, very different hierarchical structures lead to the same values of some standard inheritance metrics, resulting in lack of discrimination anomaly. This prevents the developers in effectively analyzing class hierarchies for maintainability, testability and reusability of class hierarchies. As a case study, we proposed a vector valued measure DIPV= (AID, S, U) for discriminating class hierarchies. Using ClassIN Visualization module, DIPV is plotted for different class hierarchies in Figure 3. AID provides some insight into the inheritance levels in a given class hierarchy, whereas the specialization ratio S and the reuse ratio R give the idea of the breadth of the hierarchy and the depth of reuse. Thus, the triple gives a fair idea of inheritance structure of a class hierarchy. Two DIPV vectors are compared as follows. If $u = (x_1, y_1, z_1)$ and $v = (x_2, y_2, z_2)$ are two vectors then $u > v$ if and only if one of the following conditions hold (i) $x_1 > x_2$ (ii) $x_1 = x_2$ and $y_1 < y_2$ (iii) $x_1 = x_2$, $y_1 = y_2$ and $z_1 > z_2$. Otherwise the two DIPV vectors are equal. Low DIPV value of class hierarchy indicates low maintainability, high reusability and better testability of the software. The tool also helps in suggesting various measures for analyzing modifiability and understandability using its visualization module. Figure 4 shows a snapshot of the output of visuliazation module where modifiability and understanblity metric values of certain class hierarchies are plotted agaisnt the class numbers in the project. The first class hierarchy has highest maintainability vallue among the all the class hierarchies. The graph suggests the designers should reconsider the class hierarchy structure.

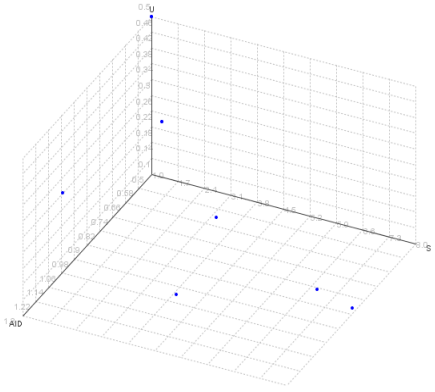

Fig. 3. DIPV values of class hierarchies

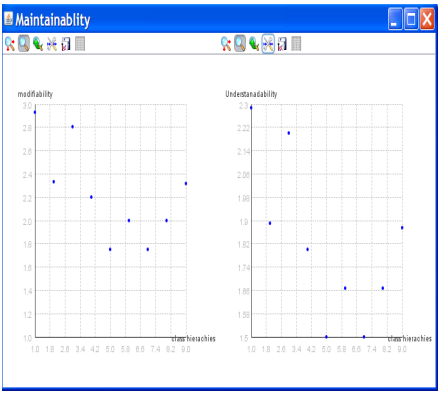

Fig. 4. Maintainability of class hierarchies

5 Tool Architecture

The architecture of ClassIN tool is presented in Figure 5. The ClassIN tool is decomposed into three modules. The first module takes a java project as an input and finds total number of classes in the project using reflection classes. In the second module, tool finds inheritance relations between classes. In the third phase, tool

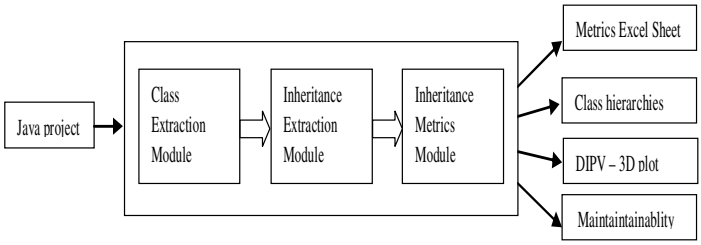

Fig. 5. ClassIN tool architecture

measures all the inheritance metrics for the project at class level as well as class hierarchy level. The tool generates four outputs namely metrics excel file, display of class hierarchies in the project, DIPV plot and graphs for analyzing maintainability of class inheritance hierarchies. Maintainability values for each class hierarchy are also displayed to ascertain class structures with higher maintenance costs. This also helps in discrimination of different class inheritance patterns in the project.

6 Tool Availability

The tool along with user guide and technical documentation, may be freely downloaded from its webpage at

https://sites.google.com/site/brcreddyse/Tools

References

1. Chidamber, S.R., Kemerer, C.F.: A Metrics Suite for Object Oriented Design. J. IEEE Trans. Soft. Eng. 20(6), 476–493 (1994)
2. Henderson-Sellers, B.: Object Oriented Metrics: Measures of Complexity, pp. 130–132. Prentice-Hall (1996)
3. Sheldon, F.T., Jerath, K., Chung, H.: Metrics for Maintainability of Class Inheritance Hierarchies. J. Soft. Main. and Evol. Res. and Pra. 14(3), 147–160 (2002)
4. Basili, V.R., Briand, L.C., Melo, L.W.: A Validation of Object-Oriented Design Metrics as Quality Indicators. J. IEEE Trans. Soft. Eng. 22(10), 751–761 (1996)
5. Subramanyan, R., Krisnan, M.S.: Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects. J. IEEE Trans. Soft. Eng. 29(4), 297–310 (2003)
6. Cartwright, M., Shepperd, M.J.: An Empirical Investigation of an Object-Oriented Software System. J. IEEE Trans. Soft. Eng. 26(8), 786–796 (2000)
7. Bruntink, M., Deursen, A.V.: An Empirical Study into Class Testability. J. Sys. and Soft. 79, 1219–1232 (2006)
8. Baudry, B., Traon, Y.L.: Measuring Design Testability of a UML Class Diagram. J. Info. and Soft. Tech. 47, 859–879 (2005)

9. Harrison, R., Counsell, S.J.: An Evaluation of the Mood set of Object-Oriented Software Metrics. J. IEEE Trans. Soft. Eng. 21(12), 929–944 (1995)
10. Abreu, F.B., Carapuca, R.: Candidate Metrics for Object-Oriented Software within a Taxonomy Framework. J. Sys. and Soft. 26, 87–96 (1994)
11. Vernazza, T., Granatella, G., Succi, G., Benedicenti, L., Mintchev, M.: Defining Metrics for Software Components. In: 5th World Multi-Conference on Systemics, Cybernetics and Informatics, Florida, vol. XI, pp. 16–23 (2000)
12. Lorenz, M., Kidd, J.: Object-Oriented Software Metrics. Prentice Hall (1994)
13. Tegarden, D.P., Sheetz, S.D., Monarchi, D.E.: A Software Complexity Model of Object-Oriented Systems. J. Dec. Sup. Sys. 13, 241–262 (1995)
14. Li, W.: Another Metric Suite for Object-Oriented Programming. J. Sys. and Soft. 44, 155–162 (1998)
15. AlGhamdi, J., Elish, M., Ahemed, M.: A Tool for measuring Inheritance Coupling in Object Oriented Systems. J. Info. Sci. 140, 217–227 (2002)
16. CKJM metric tool, http://www.spinellis.gr/sw/ckjm/
17. Analyst4J metric tool, http://www.codeswat.com/cswat/index.php
18. Eclipse plug-in 1.3.6 tool, http://www.sourceforge.net/projects/metrics/
19. JMT tool, http://www-ivs.cs.uni-magdeburg.de/sw-eng/agruppe/forschung/tools/
20. VizzAnalyzer tool, http://www.arisa.se
21. Dependency Finder tool, http://www.depfind.sourceforge.net/
22. Alghamdi, J., Rufai, R., Khan, S.: OOMeter: A Software Quality Assurance Tool. In: 9th European Conference on Software Maintenance and Reengineering, pp. 190–191. IEEE Computer Society, Manchester (2005)
23. SD Metrics tool, http://www.sdmetrics.com/
24. Daly, J., Brooks, A., Miller, J., Roper, M., Wood, M.: Evaluating Inheritance Depth on the Maintainability of Object-Oriented Software. J. Emp. Soft. Eng. 1, 109–132 (1996)
25. Genero, M., Piattini, M., Calero, C.: A survey of metrics for UML class diagrams. J. Obj. Tech. 4(9), 59–92 (2005)
26. Prechelt, L., Unger, B., Philippsen, M., Tichy, W.: A Controlled Experiment on Inheritance Depth as a Cost Factor for Code Maintenance. J. Sys. and Soft. 65, 115–126 (2003)
27. Harrison, R., Counsell, S.J., Nithi, R.: Experimental Assessment of the Effect of Inheritance on the Maintainability of Object-Oriented Systems. J. Sys. and Soft. 52, 173–179 (2000)
28. Catal, C.: Software Fault Prediction: A literature review and current trends. J. Expert Sys. with App. 38, 4626–4636 (2011)
29. Chen, J., Huang, S.: An Empirical Analysis of the Impact of Software Development Problem Factors on Software Maintainability. J. Sys. and Soft. 82, 981–992 (2009)
30. Genero, M., Manso, E., Visaggio, A., Canfora, G., Piattini, M.: Building Measure-based Prediction Models for UML Class Diagram Maintainability. J. Emp. Soft. Eng. 12, 517–549 (2007)