World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:3, 2008

Dynamic Metrics for Polymorphism in Object Oriented Systems

Parvinder Singh Sandhu, and Gurdev Singh

***Abstract*—**Metrics is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules. Software metrics are instruments or ways to measuring all the aspect of software product. These metrics are used throughout a software project to assist in estimation, quality control, productivity assessment, and project control. Object oriented software metrics focus on measurements that are applied to the class and other characteristics. These measurements convey the software engineer to the behavior of the software and how changes can be made that will reduce complexity and improve the continuing capability of the software. Object oriented software metric can be classified in two types static and dynamic. Static metrics are concerned with all the aspects of measuring by static analysis of software and dynamic metrics are concerned with all the measuring aspect of the software at run time. Major work done before, was focusing on static metric. Also some work has been done in the field of dynamic nature of the software measurements. But research in this area is demanding for more work. In this paper we give a set of dynamic metrics specifically for polymorphism in object oriented system.

***Keywords*—**Metrics, Software, Quality, Object oriented system, Polymorphism.

I. Introduction

SOFTWARE metrics are an integral part of the state-of-the-practice in software engineering. Software metrics defines as the continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information [4] , together with the use of those techniques to improve that process and its products [3]. If the metric is to provide useful information, everyone involved in designing, implementing, collecting data for and utilizing a software metrics must understand its definition and purpose.

Basic problems encountered when trying to accurately and reasonably measure dynamic properties of a program are determining and assessing specifications, desirable metric qualities, technical limitations on data collection etc. Software metrics [8][9] measure different aspects of software product and therefore play an important role in analyzing and improving software quality. Most of the metrics are based on criteria such as number of classes, number of links, number of inheritance and composition relationships, ratios attributes and operations in each class, depth of inheritance hierarchies etc.

Parvinder S. Sandhu is with Computer Science & Engineering Department, Rayat & Bahra Institute of Engineering & Bio-Technology, Sahauran, Distt. Mohali, Punjab, 140104, India (phone: +91-98555-32004; e-mail: parvinder.sandhu@gmail.com).

Gurdev Singh is Lecturer with Rayat & Bahra Group of Institutes, Rail Majra (Punjab)-India.

The field of metric in the object oriented system has many characteristics. In [1] had already 100 metrics are given to find out the complexity in software code. And in the field of object oriented system, in [2] there were more than 150 proposed metrics are given. But most of the metrics are based on the individual model and also of static nature.

In this paper we focus on polymorphism in object oriented system. And give a set of 11 dynamic metrics for polymorphism in object oriented system.

This paper is organized as follows:

Section 2 defines the various steps to design a metric, Section 3 defines the properties on which metrics are designed, Section 4 define the way to classifying the metrics, section 5 defines the set of polymorphic metrics to measure dynamic nature, Section 6 defines the experimental details, Section 7 illustrate the analysis of results and finally conclusion and references are given in Section 8 and Section 9 respectively.

II. Designing Steps

This section will discuss some steps to documenting the design of object oriented software metrics in order to insure understanding:

a) Objective Statement: The objective for each metric can be formally defined in terms of one of the following functions, the attribute of the entity being measured and the goal for the measurement.

- Understand: Metrics can help us to understand more about our software products, processes and services.
- Evaluate: Metrics can be used to evaluate our software products, processes and services against established standards and goals.
- Control: Metrics can provide the information that we need to Control resources and processes used to produce our software.
- Predict: Metrics can be used to predict attributes of software entities in the future.

b) Clear Definitions: The second step in designing a metric is to agree to a standard definition for the entities and their attributes being measured. When we use terms like defect, problem report, size and even project, other people will interpret these words in their own context with meanings that may differ from our intended definition. These interpretation differences increase when more ambiguous terms like quality, maintainability and user-friendliness are used.

c) Define the Model: The model defines how we are going to calculate the metric. Some metrics called metric primitives that are measured directly and their model typically consists of a single variable. Other more complex metrics are modeled using mathematical combinations of metrics primitives or other complex metrics. Most modeling includes an element of simplification. When we create a software measurement model we need to be pragmatic. If we try to include all of the elements that affect the attribute or characterize the entity our model can become so complicated that it's useless. Being pragmatic means not trying to create the perfect model. Pick the aspects that are the most important. Remember that the model can always be modified to include additional levels of detail in the future.

d) Establish Counting Criteria: The next step in designing a metric is to break the model down into its lowest level metric primitives and define the counting criteria used to measure each primitive. This defines the mapping system for the measurement of each metric primitive.

e) Decide what is Good: The fifth step in designing a metric is defining what is good. Once you have decided what to measure and how to measure it, you have to decide what to do with the results. Is 10 too few or 100 too many? Should the trend be up or down? What do the metrics say about whether or not the product is ready to ship?

f) Metrics Reporting: The next step is to decide how to report the metric. This includes defining the report format, data extraction and reporting cycle, reporting mechanisms and distribution and availability.

g) Additional Qualifiers: The final step in designing a metric is determining the additional metric qualifiers. A good metric is a generic metric. That means that the metric is valid for an entire hierarchy of additional extraction qualifiers. The additional qualifiers provide the demographic information needed for various views of the metric. The main reason that the additional qualifiers need to be defined as part of the metrics design is that they determine the second level of data collection requirements.

III. Designing Properties

Designing new dynamic metrics [7] must ensure that they effectively capture the aspect of software behavior that they are intended to measure. New metrics [5] must also render clear and comparable numbers for any kind of program. Therefore, we discuss some general requirements for dynamic metrics, which address some of the most important factors, which may impact their usefulness. These properties not only helpful in designing the metrics, but can also be used in the evaluation of the applicability of a particular metric to specific purposes. These desirable properties [14] are only presented informally; it may not be possible to realistically achieve all of them for every metric.

a) Dynamic: A metric should measure an aspect of a program that can only be obtained by actually executing it. The dynamic nature of a metric makes it unaffected by the addition of unexecuted code to the program, because code that is never executed will obviously never contribute to the measured value.

b) Robust: A robust metric should not be overly sensitive to the size of a program's input. Using dynamic metrics the measures are heavily influenced by program behavior. A dynamic metric is robust if a "small" change in program behavior results in a correspondingly small change in the measured value.

c) Discriminating: A metric is discriminating if a large change in behavior causes a correspondingly large change in the resulting metric.

d) Unambiguous: It is crucial to provide a clear, precise and unambiguous definition of all dynamic metrics.

e) Platform Independent: Metrics pertain to program behavior, they should not change if the measurement takes place on a different platform. While it may seem like platform-independence is easily achieved in any languages.

IV. Classification

Classification of the metrics into four basic categories is presented here. These categories correspond to the ubiquitous value metrics such as average, hot spot detection metrics and metrics based on discrete categorization.

a) Value Metric: The value metric is the most commonly used kind of dynamic metric and corresponds to typical one value answers. Many data gatherers for instance will present a statistic like average or maximum as a rough indicator of some quantity; the idea being that a single value is sufficiently accurate. Typically this is intended to allow one to easily compare results for different benchmarks, since the values form an intuitive totally ordered set. It may also be used to allow one to observe differences in behavior before and after some transformation.

b) Percentile Metric: Percentile metrics are similar to value metrics but additionally have an associated threshold value which indicates the proportion of the program entities which are to be considered in the computation of the metric i.e. the hotness level that is measured by the metric. A higher threshold is used when looking for more pronounced hotspots, and is thus associated with a higher hotness level.

c) Bin Metric: Compiler optimization is often based on identifying specific categories of measurements, with the goal of applying different optimization strategies to different cases. A call-site optimization, for instance, may use one approach for monomorphic sites, a more complex system for polymorphic sites of degree 2, and may be unable to handle sites with a higher degree of polymorphism. In such a situation single value metrics do not measure the situation well, e.g., computing an average number of types or targets per call site may not give a good impression of the optimization opportunities for de-virtualization. An appropriate metric for this example would be to give a relative or absolute value for each of the categories of interest, namely 1, 2, or _3 target types. These kinds of metrics are referred to as bin metrics, since the measurement task is to appropriately divide elements of the sample space into a few categories or bins.

d) Continuous Metrics: All three kinds of dynamic metrics have continuous analogues, where the calculations are performed at various partial stages of execution rather than once at the end of the execution. Motivation for continuous metrics arises from the inherent inaccuracy of a single summary metric value in many situations.

V. Dynamic Metrics

Dynamic polymorphic metrics measure the various aspect of the polymorphism behavior in the programs.

a) CSPV: Call Site Polymorphic Value metric count total number of different call sites executed. This measurement does not include static invoke instructions, but does count virtual method calls with a single receiver.

b) IDVP: Invoke Density Polymorphic Value metric count number of invoke Virtual and invoke Interface calls per kbc executed. This metric estimates the importance of invoke byte codes relative to other instructions in the program, indicating the relevance of optimizing invokes.

c) RPB: Receiver Polymorphic Bin metric shows the percentage of all call sites that have one, two and more than two different receiver types. The metric is dynamic, since we measure the number of different types that actually occur in the execution of the program

d) RCPB: Receiver Call Polymorphic Bin metric shows the percentage of all calls that occur from a call site with one, two and more than two different receiver types. This metric measures the importance of polymorphic calls

e) RCMRV: Receiver Cache Miss Rate polymorphic Value metric shows as a percentage how often a call site switches between receiver types. This is the most dynamic measurement of receiver polymorphism, and it represents the miss rate of a true inline cache.

f) TPB: Target Polymorphic Bin metric shows the percentage of all call sites that have one, two and more than two different target methods. This metric is dynamic, but does not reflect the run time importance of call sites.

g) TCPB: Target Call Polymorphic Bin metric shows the percentage of all calls that occur from a call site with one, two and more than two different target methods.

h) TCMRV: Target Cache Miss Rate polymorphic Value metric shows as a percentage how often a call site switches between target methods. It represents the miss rate of an idealized branch target buffer. It is always lower than the corresponding inline cache miss rate since targets can be equal for different receiver types. Accordingly, this metric can also be heavily influenced by the order in which target methods occur.

i) DPA: Dynamic polymorphism in ancestors is the sum of number of dynamic polymorphism function members in ancestor that appears in the different classes.

j) DPD: Dynamic Polymorphism in Descendants is the sum of number of dynamic polymorphism function members in descendant that appears in the different classes.

k) ACRV: Average Changing Rate of Virtual methods are used to check the efficiency by using run time method resolution.

VI. Experimental Details

The following five criteria are used for the comparative analysis:

a) Program Selection: Selection of programs was based on the key concept of dynamic polymorphism. We focus on the run time behavior of the program. We choose c++ language for the practical experiments. But the given metrics are not limited to any particular language.

b) Length of Program: Design of the metric are based on the concept of dynamics so the focus of run time properties of the program, not much focus on the program length and size.

c) Complexity: The experimental studies are much influenced by the complexity of the program. The testing of the dynamic metrics are done on the simple to complex program. As the virtual functions increases the complexity of the program in object-oriented system automatically increases.

d) Metrics: The selection of the metrics are done on the basis of dynamic behavior of the program specifically polymorphism in object oriented system. Therefore focuses on the dynamic nature of the metric are considered and calculate the results by running the programs.

VII. Analysis of Results

Polymorphism [11] is a salient feature of object-oriented languages. A polymorphic call in object oriented system [12] takes the form of invoke virtual or invoke interface.

TABLE I
Experimental Results

Metric	**I**	**II**	**III**	**IV**	**V**
CSPV	1	2	3	4	5
IDPV	5	14	11	11	23
RPB	20%	7.14%	9.09%	9.09%	4.35%
RCPB	20%	57.14%	54.55%	63.64%	60.87%
RCMRV	0%	28.57%	27.27%	36.36%	17.39%
TPB	20%	85.71%	75%	100%	76.92%
TCPB	20%	80%	75%	100%	77.78%
TCMRV	0%	16.67%	16.67%	0.14%	0.1%
DPA	1	2	2	1	1
DPD	1	1	1	1	1
ACRV	0.33	0.29	0.5	0.29	0.33

This paper is presenting eleven dynamic metrics as a means of assessing the actual runtime behavior of a program by providing a high-level overview of several of its key aspects. This dynamic information can be more relevant than the more common static measures. These metrics are designed with the goals of being unambiguous, dynamic, robust, discriminating, and platform-independent. These metrics are also classified in four categories value, percentile, and bin and continuous. The utility of the metrics is evaluating by applying them to five specific dynamic polymorphic problems, and determining to which extent they provided useful information for each task. We furthermore want to establish the preciseness of using dynamic metrics as helping tools for exploratory program understanding in object oriented system.

Implementation of dynamic metrics is given in this section on the set of five different polymorphic programs. The results are mentioned in Table I.

VIII. Conclusion

Fuzzy–GA hybrid algorithm is proved to be best as compared to the other algorithms considered in this work. In such data search application the design and developed fuzzy GA code has shown its superiority because it includes the advantages of fuzzy as well as genetic algorithms. Fuzzy provides a robust inference mechanism with no learning and adaptability while on the other hand, the genetic algorithms provide an efficient data modification in the wake of optimization objectives of given application. Neuro-fuzzy algorithm is definitely superior to fuzzy algorithm as it inherits adaptability and learning but seriously lacks optimal nature. From the simulation and the result obtained, it has been shown that the percentage average error is least in the case of fuzzy-GA algorithms and maximum in the case of fuzzy algorithms. Neuro-fuzzy algorithm has yielded accuracy lying between the accuracy levels as in the case of fuzzy and fuzzy-GA algorithms. It is concluded that for non linear and complex engineering applications involving control, inference and analysis by and large fuzzy-GA is an efficient technique.

References

[1] H. Zuse, Software Complexity – Measure and Methods, Berlin: Walter de Gruyter, 1991.
[2] T. Fetcke, Software Metrics in Object Oriented ProgrammingI, in Institute of Methods. Berlin: Technischen University Berlin, 1195, pp 161.
[3] Goodman, Paul. Practical Implementation of Software Metrics. London: McGraw Hill, 1993.
[4] F. B. Abreu, Metrics in Management of Information System Development Projects, Proceeding of 6th Conference on Quality of Software, APQ, Lisbon Portugal, 1992.
[5] M. Lorenz and J. Kidd. Object-Oriented Software Metrics: A Practical Guide. Prentice-Hall, 1994. pp 1-22.
[6] R. Pressman, Software Engineering: A Practitioner's Approach 5th Edition, McGraw-Hill Science / Engineering / Math, 2001.
[7] Thomas Ball. The concept of dynamic analysis. In Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering, Toulouse, France, 1999, pp 216–234.
[8] S. R. Chidamber and C. F. Kemerer. A metric suite for object-oriented design. IEEE Transactions on Software Engineering, 20(6): Jun. 1994, pp. 476–493.
[9] Neville I. Churcher and Martin J. Shepperd. Comments on a metrics suite for object oriented design. IEEE Transactions on Software Engineering, 21(3):, Mar. 1995,pp 263–265.
[10] Bruno Dufour, Christopher Goard, Laurie Hendren, Oege de Moor, Ganesh Sittampalam, and Clark Verbrugge. Measuring the dynamic behaviour of AspectJ programs. In Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA), Victoria, British Columbia, Canada, Oct. 2004.
[11] Karel Driesen. Efficient Polymorphic Calls. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston/Dordrecht/London, 2001.
[12] Michael D. Ernst. Static and dynamic analysis: Synergy and duality. In WODA 2003: ICSE Workshop on Dynamic Analysis, May 9, 2003, pages 24–27.
[13] Norman E. Fenton. Software measurement: a necessary scientific basis. IEEE Transactions on Software Engineering, 20(3): Mar. 1994, pp 199–206.
[14] Norman E. Fenton and Martin Neil. Software metrics: successes, failures and new directions. Journal of Systems and Software, 47(2–3): Jul. 1999, pp 149–157.
[15] Norman E. Fenton and Martin Neil. Software metrics: roadmap. In Proceedings of the Conference on the Future of Software Engineering, Limerick, Ireland, 2000, ACM Press. pages 357–370.
[16] Norman E. Fenton and Shari Lawrence Pfleeger. Software metrics: a rigorous and practical approach. PWS Publishing Company, 1997.
[17] G. Caldiera and V. R. Basili, Identifying and Qualifying Reusable Software Components, IEEE Computer, (1991), pp. 61-70.
[18] Maurice H. Halstead, Elements of Software Science (Elsevier North-Holland, New York, 1977).