Object-Oriented Software Design Metrics

Fernando Brito e Abreu

INESC, Rua Alves Redol, 9, Apartado 13069, 1000 Lisboa, PORTUGAL
(phone: +351-1-3100226 / fax: +351-1-525843 / email: fba@inesc.pt)

1. INTRODUCTION

Object-orientation is not a panacea for successful system development as noted in [Jacobson92]. The shift from craftsmanship to industrialism must come on a more fundamental level that also includes the organization of the complete development process. Object-Oriented Software Engineering (OOSE) has, accordingly, received an inflated attention. Its main objective is to make the OO software development process an engineering activity, either by adapting "traditional" software engineering techniques, or by proposing its own. OOSE deals with technical and managerial issues, the former having received much more attention in the past few years.

Several problem areas in OOSE are, among others, the adoption of the OO technology itself (i.e. paradigm shift), the lack of adequate life-cycle models that support reusability, the ability to assess the quality of the development process and resulting products, and the capability of evaluating the productivity of development teams. These last two issues are fundamental in order for managers to control, steer and follow up software development efforts.

Quality and productivity are indeed the two most important input parameters for controlling any industrial process. Successful control requires some means of measurement. The need for software metrics is now fully recognized by the software engineering community and included in standards like the [ISO9000-3]. The reasons for using metrics in software development are mainly independent of the adopted paradigm, although the latter deeply influences the set of metrics to choose, as its concepts and corresponding abstractions are either disjoint or implemented differently [Abreu93].

This paper aims at achieving some advances towards a solution to the problem of assessing quality and productivity on OO systems and is organized as follows: the next section introduces the specific goals of the current research work in quantitative methods applied to Object-Orientation from which this paper originated. Section 3 presents a set of criteria for choosing a suitable metrics set. Section 4, the core of this paper, includes the detailed proposal of a metrics set named MOOD, for guiding and assessing OO design quality and potential productivity gains. The possible shapes of expected design recommendations based on the MOOD set are introduced in section 5. Some complementary research topics that deserve further effort are identified in section 6. Some concluding remarks are presented in section 7.

2. QUALITY AND OBJECT-ORIENTATION

Quality of software systems can be characterized by the presence of a certain number of external[1] attributes like functionality, reliability, usability, efficiency, maintainability and portability [ISO9126] which can be further detailed. However, due to the evident hardness and cost[2] of evaluating those attributes, most efforts on the software quality field have focused on defining and evaluating suitable development processes. It is believed that a defined and controlled process will lead to the production of quality software products and with fewer costs. Let us call this approach "outside-in". Object-Orientation came to strengthen the

[1] - Perceptible to purchasers, subcontractors and end users.
[2] - In industries other than software, quality assessment is often done by evaluating samples of massively produced products, thus allowing diluting the corresponding effort (i.e. scale economy benefit).

complementary "inside-out" approach where the quality of internal structure is supposed to be the key for ensuring that (external) quality and increased productivity are achieved.

Object-Orientation is well suited for what is called "seamless development", which basically stands for using the same formalism and corresponding notation throughout the life-cycle, by means of stepwise refinement. The traditional barriers between analysis and design and particularly between design and coding, characterized by formalism shifts with corresponding translation rules, are bound to diminish. Therefore, analysis and design play an even more important role than ever. Coding, for instance, can be considered just as a "fill-in-the-design-blanks" activity. Better internal quality is due to new abstractions brought by this paradigm such as classes, methods, inheritance, polymorphism, encapsulation or messages and a corresponding increased emphasis on reuse. However, the use of those abstractions can be varied, depending mainly on the designer ability, so we can expect rather different quality products to emerge, as well as different productivity gains.

The aim of the research going on in our team is twofold. First, we want to be able to identify quality OO designs by means of quantitative evaluation (i.e. using metrics) of the use of the paradigm abstractions that are supposed to be responsible for internal quality. Second, we want to express some of the external quality attributes and productivity advances as a function of those metrics. Our first step was to develop some metrics for OO designs, from a set of criteria presented in next section, that among other things, is expected to allow doing comparative evaluation throughout the OO community, and eventually help training new OO software practitioners by setting design standards that traduce best practice.

3. CRITERIA FOR DESIRED METRICS

The choice of a set of metrics exposes the pitfalls of measuring too much and becoming overwhelmed by a large amount of unmanageable numeric data, or measuring too little and not gaining sufficient insight into the desired objective. After surpassing this problem by deciding to adopt just a few (but not too few) metrics, we need to set some evaluation criteria based on the goals we want to achieve. Without them it is relatively easy to fall in the "YAM"[3] trap or becoming swamped by the myriad of those proposed in the available literature. See [Zuse91], for instance, where a few hundred are mentioned. Next we will derive a set of criteria to help define the MOOD set.

Different people at different times or places should yield to the same values when measuring the same systems. Subjectivity makes metrics comparisons throughout software industry an impossible mission. Subjective ratings (e.g. "Very Low", "Low", "Average", "High", "Very High") are copious in the metrics literature. That is undoubtedly one of the reasons that leads to metrics suspicion among software practitioners and the computer science community in general. One road to achieve this objectivity is:

Criterion 1: metrics should be formally defined

For being useful, metrics must be collected and analyzed throughout time in as many different projects as possible in order to establish comparisons and derive conclusions. However, those projects will surely vary in size. If metrics other than the ones specifically designed to measure size also depend on it, no cumulative knowledge will be achieved. So:

Criterion 2: non-size metrics should be system size independent

Metrics are supposed to represent some product or process attribute. Thus we are faced with the issue of

[3] - Yet Another Metric...

units of measurement. Subjective or "artificial" units inevitably yield to misunderstandings. Remember, for instance, the discussions around different interpretations of LOC (lines of code) [Albrecht83] and Function Points [Symons91][Dreger89]. Then:

Criterion 3: metrics should be dimensionless or expressed in some consistent unit system

The cost of recovering from an error increases exponentially with elapsed project progress since its commitment. Metrics, particularly design ones, are aimed at exposing the defects provoked by those errors and buried in the design. We must be able to collect metrics as soon as a first design is available, if we want to identify the possible flaws, before too much effort is built on top of them. Therefore:

Criterion 4: metrics should be obtainable early in the life-cycle

Real software systems are usually built by a team of people. Often is possible to break down the specification in almost independent modules or subsystems. Each team member or small group of members can be responsible for each of those subsystems. Then, we need metrics applicable not only to the whole system under consideration, but also to each one of its modules or subsystems, thus allowing to pin-point "ill-designed" ones. So:

Criterion 5: metrics should be down-scalable

Metrics collection is a repetitive task, therefore tedious and boring for human beings. The worst is that it takes a lot of time and money! Provided that criterion 1 is met, and that designs are also formally defined, it is possible to build some kind of syntactic analyzer that extracts from them the needed information for computing the metrics. The effort to build such a tool is considerable but it is worth. Then:

Criterion 6: metrics should be easily computable

Many specification and programming languages (either graphical or textual) that support the OO paradigm abstractions are available in the marketplace. Each of them has its own constructs that allow for implementation of those abstractions in more or less detail. Again, the requirement of a common base of understanding for the metrics analysis process, leads us to the need of avoiding the syntactic level. Tools such as those mentioned in the previous paragraph can guarantee this independence. Therefore:

Criterion 7: metrics should be language independent

Several authors have suggested sets of metrics for the OO paradigm; see for instance [Bieman92] [Campanai94] [Chidamber91] [Karunanithi93] or [Yousfi92]. However, most of the proposed metrics do not fulfill all the above criteria, mainly 1, 2 and 3.

4. THE MOOD METRICS SET

4.1 Introduction

The MOOD (Metrics for Object Oriented Development) set is a collection of metrics that were designed with the above defined criteria in mind. The set includes the following metrics:

- Method Inheritance Factor
- Attribute Inheritance Factor
- Coupling Factor

- Polymorphism Factor
- Method Hiding Factor
- Attribute Hiding Factor
- Reuse Factor

Considering that metrics are intended to quantify the presence or absence of a certain property or attribute, we can view them as probabilities. They would then range from 0 (total absence), to 1 (maximum possible presence). This perspective was also used in the ESPRIT REBOOT project [Stalhane92]. This kind of interpretation allows the application of statistical theory to software metrics. For instance, statistical independent metrics can be combined (e.g. multiplied) so that the result can still be interpreted as a probability.

The MOOD metrics definitions are based on a group of formally defined functions and on set theory and simple mathematics. This fulfills criteria 1 and 6. All MOOD metrics are expressed as quotients where the numerator is the actual use (in the design under consideration) of a given OO abstraction and the denominator is the maximum achievable possible value for the same abstraction use. As a result, all metrics are size independent and dimensionless and criteria 2 and 3 are met. The MOOD metrics meet criterion 4 because they are applicable as soon as a preliminary system design is available. The proposed metric set can be applied to any existent Class Cluster (as defined in section 4.4), or any combination of them, and so we can say that criterion 5 is also fulfilled. No reference is made to specific language constructs, that is, MOOD metrics refer to OO abstractions, and not to its implementations. Criterion 7 is therefore also accomplished.

5. DESIGN HEURISTICS

By thorough interpretation of data taken from real projects, we believe that we will be able to compute design heuristics. Those can exhibit three shapes: recommended lower limit (LL), recommended interval (INT) and recommended upper limit (UL). Table 1 shows which shape applies for each of the MOOD metrics. The appropriateness of each limit (including the interval ones) is expected to increase as our metrics collection and analysis process proceed.

Table 1: Shape of design heuristics based on MOOD

MOOD METRIC	**LL**	**INT**	**UL**
Method Inheritance Factor		x	
Attribute Inheritance Factor		x	
Coupling Factor			x
Polymorphism Factor			x
Method Hiding Factor	x		
Attribute Hiding Factor	x		
Reuse Factor	x		

According to this framework, expected recommendations will be of the kind:

- "Keep the Method Inheritance Factor between 0.25 and 0.37"
- "Coupling Factor should be below 0.52"
- "Good Reuse Factors are considered to be above 0.43"

Values mentioned above are irrelevant. We expect to disclose some realistic ones in a following paper.

6. FUTURE WORK

We are presently developing a tool for supporting the collection, storage and analysis of the MOOD metrics set. The core of this tool (metrics definition dictionary, metrics storage, human-machine interface) is language independent. Specific implementation language stubs will parse the specification code and determine the base measure function values. The C++ stub is under construction and we plan to develop an Eiffel [Meyer92] one in the near future. When the tool becomes fully operational, we will proceed to an extensive evaluation of available systems and try to derive and refine the values for the limits mentioned in the design recommendations.

The study of correlation between MOOD metrics and quality attributes as those mentioned in [ISO9126] will be one of next steps. We will also investigate the statistical independence of each MOOD metric towards each of the other ones.

We think that the MOOD metrics (except the Reuse Factor) can be combined to obtain a generic OO software system complexity metric. That is one of our future challenges. We will start by evaluating the MOOD metrics against a set of desiderata for software complexity metrics defined in [Weyuker88].

A concurrent effort for developing a resource estimation model named MOORED (Model for Object Oriented Resource Estimated Determination) is under way, and some cross-fertilization is expected in the field of complexity and productivity evaluation.

7. CONCLUSIONS

The adoption of the Object-Oriented paradigm is expected to help produce better and cheaper software. The main concepts of this paradigm, namely, inheritance, encapsulation, information hiding or polymorphism, are the keys to foster reuse and achieve easier maintainability. However, the use of constructs that support those concepts can be more or less intensive, mainly depending on the designer ability. Advances in quality and productivity need to be correlated with the use of those constructs. Therefore, we need to evaluate them quantitatively to guide OO design. The availability of these metrics should allow comparison of different systems or different implementations of the same system, thus helping to derive some design heuristics that could/should be included in design tools. Those heuristics would at least be a valuable help to new staff members.

"Blind" choice (or creation) is dangerous, so a set of common requirements for metrics and corresponding rationale was introduced, which includes the need for formal definition, language independence, dimensionlessness, ease of calculation and early obtainability. A suitable metrics set named MOOD was then proposed. We believe that these metrics can help in setting OO design standards at the organization level, helping OO practitioners to guide their development process and, hopefully, leaving them in a cheerful MOOD...

REFERENCES

[Abreu93] Abreu F.B., "Metrics for Object Oriented Software Development", Proceedings of 3rd International Conference on Software Quality, ASQC, Lake Tahoe, USA, October 1993.

[Albrecht83] Albrecht A.J. and Gaffney J.E., "Software Function, Source Lines of Code and Development Effort Prediction", IEEE TSE, vol.9, n.6, pp.639-648, November 1983.

[Bieman92] Bieman J., "Deriving Measures of Software Reuse in Object Oriented Systems", Proceedings of The BCS-FACS Workshop on Formal Aspects of Measurement, Springer-Verlag,LNCS, 1992.

[Campanai94] Campanai M. and Nesi P., "Supporting O-O Design with Metrics", Proceedings of TOOLS Europe'94, France, 1994.

[Chidamber91] Chidamber S. and Kemerer C., "Towards a Metrics Suite for Object Oriented Design", Proceedings of OOPSLA'91, pp.197-211, 1991.

[Dreger89] Dreger J.B., Function Point Analysis, Prentice-Hall, Englewood Cliffs, NJ, ISBN 0-13-332321-8, 1989

[ISO9000-3] ISO/IEC 9000 Part 3, Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software, 1991.

[ISO9126] ISO/IEC 9126, Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for their use, 1991.

[Jacobson92] Jacobson I., Christerson M., Jonsson P. and Övergaard G., Object-Oriented Software Engineering - A Use Case Driven Approach, ACM Press / Addison-Wesley, 1992.

[Karunanithi93] Karunanithi S. and Bieman J., "Candidate Reuse Metrics For Object Oriented and Ada Software," Proceedings of IEEE International Software Metrics Symposium, pp.120-128,Baltimore, May 1993.

[Meyer88] Meyer B., Object-oriented Software Construction, Prentice Hall International, 1988.

[Meyer92] Meyer B., Eiffel: The Language, Prentice Hall International, 1992.

[Stalhane92] Stalhane T. and Coscolluela A., "Final Report on Metrics", Deliverable D1.4.B1, ESPRIT Project 5327 (REBOOT), February 1992.

[Symons91] Symons C.R., Software Sizing and Estimating - Mk II Function Point Analysis, John Wiley & Sons, ISBN 0-471-92985-9, 1991.

[Weyuker88] Weyuker E., "Evaluating Software Complexity Metrics", IEEE TSE, vol.14, n.9, pp.1357-1365, September 1988.

[Yousfi92] Yousfi N., "Measuring Internal Attributes of Object-Oriented Software Products", Proceedings of the 5th Int. Conference on Software Engineering & its Applications, Toulouse, 1992.

[Zuse91] Zuse H., Software Complexity: Measures and Methods, Walter de Gruyer, 1991.

About the Author

Fernando Brito e Abreu has an Electrical Engineering degree and a MSc in Electrical and Computer Engineering, both from IST (Lisbon Technical University, Portugal). He is a lecturer at ISEG (Lisbon Technical University) since 1988. He also conducts training and consulting in Software Engineering Project Management for the Portuguese Software Industry. He is a member of the Software Quality Promotion Group (GDQS) of the Portuguese Quality Assurance Association (APQ), the IT&T Commission of the National Council for Quality (a Portuguese Government initiative), the Euromethod Forum (an European Commission initiative) and of the ISO/IEC JTC1 Sub-Committee 7 in Software Engineering. He leads a research group in Software Engineering Project Management at INESC[4] and is currently pursuing a PhD program. He has participated in several national and international software conferences either presenting papers and tutorials or as a panelist, panel chairman, reviewer and program committee member. His main research interests lie in the areas of Information Systems Modeling and Techniques, Object-Oriented Development, Metrics and Estimation Models.

[4] - INESC is a private non-profit association, dedicated to research, development and training in advanced technological areas. It is an interface between the Portuguese Telecommunications and Information Technology sectors and the University system.