

OO Metrics in Practice

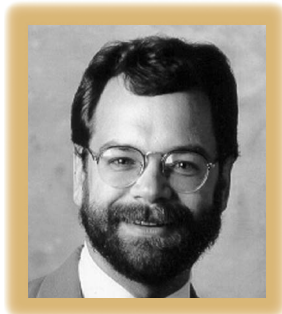
David P. Darcy and Chris F. Kemerer

While it has long been recognized that software process improvement requires measuring both the process and its performance, experience has also shown that few universal metrics exist. The most effective measurement tools are specialized to some aspect of the task or domain being measured.

This condition motivated our initial academic work with Shyam Chidamber in object-oriented (OO) metrics, which we released in

size and individual developer effects. The three systems made very little use of inheritance.

Since this initial work, researchers and practitioners around the world have used our metrics suite and its variants in a wide variety of OO software settings. In addition, many commercial tools are now available to automatically collect some or all of these metrics (see the sidebar, “Commercial Tools for OO Metrics Collection”). A variety of studies have documented relationships between OO metrics and managerial-performance variables including effort, reusability, defects and faults, maintainability, and cost savings (see the References).



Applying OO metrics to commercial systems

A typical application of the CK metrics is identifying trouble spots in the code that might require restructuring or reorganization. Karin Erni and Claus Lewerentz at ABB Corporate Research in Germany investigated SemLib, a semantic graphics library that is a framework for graphical applications development.⁴ The authors examined 31 classes common across three SemLib releases (1.0, 2.1, and 2.2) using multiple measures for each of three factors: size (seven measures), coupling (four measures), and cohesion (five measures). They applied judgment to the measures to identify problem areas and give hints for improvement. They used the results in design reviews to quantify the effect of reorganization efforts.

While reusability is an important consideration of OO systems, it is particularly important for frameworks, given their extensive reuse throughout the family of applications

1991¹ and enhanced and formalized as a suite of six metrics in 1994 (hereafter, the CK metrics).² Our 1998 study explored the relationships between metrics and various outcomes of practical importance in a new data set containing the details from three projects involving support systems for financial traders in a European bank.³ Each project covered a different phase in the development process: design, development, or reuse. The data set showed a consistent effect for coupling between objects (CBO) and lack of cohesion of methods (LCOM) on various factors of managerial interest, such as productivity, reuse effort, and design effort—an impact over and above that of

that use a specific framework. Thus, controlling or reducing complexity is even more pertinent. Erni and Lewerentz used the CK metrics in a decision-support sense; they highlighted framework parts that lie outside acceptable thresholds or that trended abnormally, and a class browser was available to explore reasons for the exceptions. Comments from the developer generally validated the highlighted problem areas and even surfaced a previously unknown problem area.

The metrics as measures of code have often been related to external factors, such as software quality in the sense of defects. Ramanath Subramanyam and Mayuram Krishnan from the US investigated the relationship among three of the CK metrics—CBO, weighted methods per class (WMC), and depth of the inheritance tree (DIT)—and defects for a business-to-consumer e-commerce application.⁵ The authors analyzed 405 C++ classes and 301 Java classes and abstracted defects from a configuration management system after about three months of release. For the C++ classes, WMC, DIT, and CBO*DIT were all significant predictors of defects above size. For the Java classes, only CBO*DIT was a significant predictor above size. As earlier work suggested,^{2,3} DIT was relatively invariant, suggesting that inheritance was not being used.

Similarly, Lionel Briand and Jürgen Wüst studied a small system (approximately 17,000 SLOC, 103 classes, and seven developers) originally developed in Italy for editing music scores.⁶ They analyzed the effort for developing a class using Poisson regression and regression trees. Briand and Wüst found that design measures such as coupling and cohesion added value in increasing models' accuracy above that of size alone. Their paper tested various ways of developing prediction systems for effort from design measures.

Briand, Wüst, and Hakim Lounis replicated earlier results from an academic setting (examining student projects) in an industrial setting and tested the relationship between the CK suite and defects.⁷ They studied a single industrial

system (a C++, open, multiagent system-development environment) with 90 classes and 40,000 SLOC. The findings were similar between the two settings, although enough dissimilarity existed for the authors to suggest that local model calibration would be important.

Long-term studies

Software metrics studies often use single snapshots of a software project. Examining a project over a longer time frame allows consideration of other software quality facets, such as reuse and maintainability. F. George Wilkie and Barbara Kitchenham in the UK examined the relationships between CBO (and various extensions) and the practical performance issues of reusability and maintainability.⁸ In a study of a multimedia conferencing system in production for two and a half years with 114 classes and 25 KSLOC, they were able to show a significant relationship between the metrics and reuse and maintainability.

US researchers Mei-Huei Tang, Ming-Hung Kao, and Mei-Hwa Chen studied three C++ subsystems (20, 45, and 27 classes) of an industrial real-time software system over a three-year period.⁹ They examined relationships between the CK metrics and faults identified in the testing or maintenance phases. As is often the case, DIT and number of children (NOC) were small, tending to zero. However, WMC and

response for a class (RFC) were significantly related to classes with faults across the three subsystems. The authors developed additional metrics based on a CK suite analysis. A model that correctly identified the largest amount of classes with faults involved both the CK suite and the new metrics and could correctly identify more than 90 percent of faulty classes.

Creative uses

In typical software projects' haste to get things done, the value of reflecting on a design using metrics isn't always obvious. Colin Kirsopp, Martin Shepherd, and Steve Webster conducted a lab study in the UK investigating the use of measurement to support OO design evaluation.¹⁰ They asked study participants, including industrial participants, to evaluate a small Java system (15 classes) using an analysis tool that let them apply various metrics and to use design heuristics and the analysis tool to critique the system. The authors found that participants were able to make better claims about the design with the tool's support, though using the tool probably increased the amount of false positives (of design problems). Interestingly, the participants tended to focus on a small set of available metrics and use that set for evaluating the design. The CK suite was the most popular subset, a result possibly explained by the larger amount of explanation available for it compared to the other subsets. This study further emphasized how software measures can fulfill a decision support role, even for relatively inexperienced participants.

Although difficult to reliably quantify, one study provides some insight into actual cost savings that could be achieved by applying metrics as predictors of fault-prone classes and then directing testing efforts to those classes.¹¹ Daniela Glasberg, Khaled El Emam, Walcelio Melo, and Nazim Madhavji studied a commercial XML-document editor developed in Java with 145 classes and 170 post release faults in 34 of those classes. Through an elaborate analysis and using the metrics, they derived a prediction model for

In typical software projects' haste to get things done, the value of reflecting on a design using metrics isn't always obvious.

Commercial Tools for OO Metrics Collection

Many commercial tools enable practitioners and researchers to collect OO metrics. Table A lists seven currently available tools.

Table A

Commercial tools for OO metrics collection

Tool	Vendor	Languages	URL	CK suite coverage*
Krakatau Metrics	Power Software	C++ and Java	www.powersoftware.com	All
JStyle	Codework	Java	www.codework.com	WMC, DIT, RFC, LCOM
Project Analyzer	Aivosto	Visual Basic	www.aivosto.com	All
RSM Metrics	M Squared Technologies	C++ and Java	www.msquaredtechnologies.com	DIT, NOC
SDMetrics	SDMetrics	C++ and Java	www.sdmetrics.com	WMC, DIT, NOC, CBO, RFC
Software Metrics	McCabe & Associates	C++ and Java	www.mccabe.com	WMC, DIT, NOC, RFC, LCOM
Understand	Scientific Toolworks	C++ and Java	www.scitools.com	All

*CBO = coupling between objects, DIT = depth of the inheritance tree, LCOM = lack of cohesion of methods, NOC = number of children, RFC = response for a class, WMC = weighted methods per class

faulty classes. Different levels of cost savings are present depending on the number of classes the authors chose for prerelease testing. In the best case, they conservatively estimated that further testing on 39 percent of the classes predicted to be the most fault prone would save 42 percent of the actual post-release cost of fixing defects. Given the high cost of correcting defects post release, it is clear that some investment in additional testing could save considerable resources that would otherwise be expended later in the project's life.

Across a wide variety of reported results from using OO metrics in industrial settings and using data from an assortment of countries and applications, we can make several observations:

- OO metrics have been successfully applied in various domains and programming languages in countries worldwide.
- They have consistently demonstrated relationships to quality factors such as cost, defects, reuse, and maintainability—relationships that go above and beyond that of size.
- A metric set consisting of size (measured by SLOC or specific variants of WMC), coupling (measured by

CBO or RFC), and cohesion (measured by LCOM) measures is generally useful in varying prediction models.

- Inheritance (measured by DIT or NOC) is apparently used only sparingly in practical OO applications, and thus its relationship to project outcomes is less certain.
- As would be expected, the exact relationship between the metrics and prediction of outcomes of managerial interest need to be calibrated for local influences.

We expect to see continued use and further development of OO metrics in the years ahead. ☺

References

1. S.R. Chidamber and C.F. Kemerer, "Towards a Metrics Suite for Object Oriented Design," *Proc. ACM Conf. Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA 91), ACM Press, 1991, pp. 197–211.
2. S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Software Eng.*, vol. 20, no. 6, 1994, pp. 476–493.
3. S.R. Chidamber, D.P. Darcy, and C.F. Kemerer, "Managerial Use of Metrics for Object Oriented Software: An Exploratory Analysis," *IEEE Trans. Software Eng.*, vol. 24, no. 8, 1998, pp. 629–639.
4. K. Erni and C. Lewerentz, "Applying Design-Metrics to Object-Oriented Frameworks," *Proc. 3rd Int'l Software Metrics Symp.* (MET-

- RICS 96), IEEE CS Press, 1996, p. 64–74.
5. R. Subramanyam and M.S. Krishnan, "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects," *IEEE Trans. Software Eng.*, vol. 29, no. 4, 2003, pp. 297–310.
6. L.C. Briand and J. Wust, "The Impact of Design Properties on Development Cost in Object-Oriented Systems," *Proc. 7th Int'l Software Metrics Symp.* (METRICS 01), IEEE CS Press, 2001, pp. 260–271.
7. L. Briand, J. Wust, and H. Lounis, "Replicated Case Studies for Investigating Quality Factors on OO Designs," *Empirical Software Eng.*, vol. 6, no. 1, 2001, pp. 11–58.
8. F.G. Wilkie and B.A. Kitchenham, "An Investigation of Coupling, Reuse and Maintenance in a Commercial C++ Application," *Information and Software Technology*, vol. 43, no. 13, 2001, pp. 801–812.
9. M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An Empirical Study on Object-Oriented Metrics," *Proc. 6th Int'l Software Metrics Symp.* (METRICS 99), IEEE CS Press, 1999, pp. 242–249.
10. C. Kirsopp, M. Shepperd, and S. Webster, "An Empirical Study into the Use of Measurement to Support OO Design Evaluation," *Proc. 6th Int'l Software Metrics Symp.* (METRICS 99), IEEE CS Press, 1999, pp. 230–241.
11. D. Glasberg et al., *Validating Object-Oriented Design Metrics on a Commercial Java Application*, NRC-ERB 1080, National Research Council Canada, 2000.

David P. Darcy is an Assistant Professor of information systems in the Department of Decision and Information Technologies, Robert H. Smith School of Business, University of Maryland. Contact him at ddarcy@rhsmith.umd.edu.

Chris F. Kemerer is the David M. Roderick Professor of Information Systems at the Katz Graduate School of Business, University of Pittsburgh, and an adjunct professor of computer science at Carnegie Mellon University. Contact him at ckemerer@katz.pitt.edu.