Metrics for Predicting Run-Time Failures and Maintenance Effort: Four Case Studies

Aaron B. Binkley and Stephen R. Schach
Vanderbilt University

It is important to have the ability to predict which modules in a software product are likely to be fault-prone so that corrective action can be taken. A number of metrics have been put forward to predict faults and failures, including the coupling dependency metric (CDM). In the four case studies presented here, CDM outperformed a wide variety of competing metrics. The case studies were implemented in COBOL, C, C++, and Java and used both classical and object-oriented methods. This article is a summary of research conclusions that can be examined in more detail by consulting the references at the end of this article.

The distribution of faults within any particular software product is not uniform. For example, 47 percent of the faults found by users of OS/370 were associated with only 4 percent of the modules [1]. Therefore, what is needed are metrics that can be applied to detailed designs to predict which modules will be fault-prone. Then, instead of wasting money detecting and correcting faults during implementation and integration, those modules can be redesigned before coding commences.

Metrics of this kind can also be applied to existing software, to predict which modules are likely to contain residual faults when the software is installed. Residual faults can have two major adverse consequences: run-time failures, which prevent users from making optimal use of the software; and time that has to be spent on corrective maintenance. If a fault is detected in an existing module that has been flagged as fault-prone, it may turn out cheaper to discard, redesign, and recode the module than to attempt to fix one fault in a module that probably contains many other faults.

The CDM is a metric that accomplishes this objective. It has outperformed a wide variety of other metrics, both classical and object-oriented, in predicting run-time failures and residual faults [2]. In this article, we compare the performance of CDM with other metrics on four real-world case studies: run-time failure data for a COBOL registration system, maintenance data for a C text-processing utility, maintenance data for a C++ patient collaborative care system, and maintenance data for a Java electronic file transfer facility.

Description of Method

We followed the same procedure in each of the four case studies. We chose a set of metrics that would presumably be good predictors of run-time failures or corrective maintenance for that case study. We then applied each metric in the set to each module in the case study. For example, one of the metrics was lines of code (LOC), so we counted the LOC in each module. We then chose an objective measure of software quality, for example, the number of faults detected in a module after installation. Next, we computed the Spearman rank correlation [2] between the LOC in each module and the number of faults in each module. A high correlation would indicate that modules with many LOC also have many faults, i.e., a high correlation would mean that lines of code is a good predictor of faults.

The Metrics in Our Case Studies

We compared a number of different metrics in each case study, including

This is a brief description of the metrics mentioned in this article. For detailed information on all the metrics in our four case studies, consult the Internet address listed in [3].

Lines of Code. We computed lines of code as the number of noncomment source statements.
Cyclomatic Complexity. The cyclomatic complexity metric measures the number of branches in a module.
Fan-in/fan-out. For any module M, the fan-in of M is the count of the modules that call M plus the number of global data elements. The fan-out of M is the count of modules called by M plus the number of global data elements altered by M.
Ordinal Scale Coupling. Level numbers are assigned to classical coupling categories. Specifically, level numbers 1 through 5 are assigned to data coupling, stamp coupling, control coupling, common coupling, and content coupling, respectively. The level numbers corresponding to each instance of coupling in a given module are then summed.
Response for a Class. The response set of a class is the set of methods that can potentially be executed in response to a message received by an object of that class. The response for a class (RFC) is then the number of methods in its response set.
Coupling Dependency Metric. The coupling dependency metric (CDM) is the sum of three components: a measure of the extent to which a program relies on its declarations remaining unchanged (referential dependency); a measure of the extent to which a program relies on its internal organization remaining unchanged (structural dependency); and a measure of the vulnerability of data elements in one module to change by other modules (data integrity dependency).

widely used quality metrics like cyclomatic complexity, object-oriented metrics like the depth of the inheritance tree (for the two object-oriented case studies), and CDM. The metrics compared in the four case studies are described in detail in [3]. The sidebar on page 21 provides a brief overview of the metrics mentioned in this article.

Software Product	Implementation Language	Approximate Size	Data Analyzed	Metrics Compared	Best Three Predictors
Registration system	COBOL	80,000 LOC	Run-time failures	9	1. CDM 2. Ordinal scale coupling 3. Fan-in times fan-out
Text-processing utility	C	3,000 LOC	Corrective maintenance	17	1. Referential dependency 2. CDM 3. Fan-out
Patient collaborative care system	C++	82,000 LOC	Corrective maintenance	11	1. CDM 2. Number of clients 3. Fan-in
Electronic file transfer facility	Java	6,000 LOC	Corrective maintenance	14	1. CDM 2. Fan-in 3. RFC

Table 1. *Results of the case studies showing which metrics were the best predictors of run-time failures and corrective maintenance.*

The OASIS Course Registration System

OASIS [3] is a university course registration system developed by Vanderbilt Administrative Systems. It comprises 290 COBOL modules that total approximately 80,000 lines of code. Running on a VAX series computer, the software allows multiple interactive student registration sessions to execute concurrently. Communication between the sessions is accomplished through 18 shared resources; namely, disk files and shared memory segments.

During registration periods, a run-time log is kept that records all OASIS events, including registration transactions and system-trapped failures. The data available for this case study consisted of the source code for OASIS and the run-time log for the software product, as recorded during a single registration period. For each failure, we noted which of the 18 shared resources was involved. We then attributed the failure to any module with access to that resource. Then, as previously explained, for each metric we computed the Spearman rank correlation between the value of that metric applied to each module and the number of failures attributed to that module. The results are shown in Table 1.

The three metrics most highly correlated with the number of run-time failures (the three best predictors of run-time failures) were CDM, ordinal scale coupling, and fan-in times fan-out. These are all coupling-based metrics; intramodule metrics like lines of code and cyclomatic complexity fared poorly as predictors of run-time failures.

The ffortid Text Formatting Utility

Our second case study is ffortid [3], a text formatting utility for UNIX used to format Arabic, Hebrew, and Persian text. It converts ditroff output so that designated right-to-left fonts are properly reversed, letters are properly stretched, and slantable fonts are printed on a slanted base line. The ffortid utility is written in C. It is composed of nine modules that consist of 34 functions that total approximately 3,000 lines of code.

Our study examined data from a maintenance period in which eight distinct enhancements were made to ffortid. The required changes varied in magnitude from a simple command line option change to a more complex extension that would allow slanted fonts. During the maintenance period, the developer recorded daily effort statistics including time spent working on the maintenance project, descriptions of the faults encountered as a consequence of enhancement activities, time spent correcting these faults, and the functions to which the time and faults should be attributed. From this information we were able to compute six corrective maintenance measures. The results were similar for all six measures; for brevity, we report on just one here; namely, the time spent performing corrective maintenance.

Table 1 shows that the best three predictors were referential dependency (a component of CDM), CDM, and fan-out. Again, coupling-based metrics outperformed intramodule metrics.

The Collaborative Care System

Our third case study is a comprehensive patient care management system developed by the Vanderbilt University Medical Center [3]. Designed using object-oriented methods, it is implemented in C++. The product defines 113 distinct classes and consists of 312 modules that total approximately 82,000 lines of code.

Our study examined data from a maintenance period beginning with the original installation of the product and ending with the product's second release. During this period, little or no enhancement was made to the product. Instead, maintenance consisted mostly of modifications made to correct residual faults. The configuration control tool used in the development and maintenance of the product provides accurate statistics regarding the modifications made to the source files over time. For our study, we were therefore able to compute the total number of changes (additions, deletions, or modifications of lines of code) made to each class as well as the total number of times a class was revised.

The results appear in Table 1. Again, CDM outperformed the other predictors of software quality, followed by number of clients and fan-in. In this case study, coupling-based metrics outperformed both intramodule metrics and inheritance-based (object-oriented) metrics.

The Electronic File Transfer Facility

Our fourth case study was with a program called submit, an electronic file transfer facility used in Vanderbilt University engineering courses to allow students to submit class assignments directly to their instructors for evaluation [3]. Because the hardware platforms used by students vary, Java was chosen as the implementation language for the product. Implemented in a client-server architecture with clients for both UNIX and Windows NT environments, the product defines 29 distinct classes and consists of six packages that total approximately 6,000 lines of code.

In this study, we examined data from a maintenance period beginning with the original installation of the product and ending with the installation of the product's second version. During this period, no enhancements were made to the product. Instead, maintenance consisted mostly of modifications to correct residual faults as well as design flaws with regard to the client-server architecture. The source code was meticulously annotated during maintenance to provide accurate statistics regarding the modifications made to the source files.

The results of computing the Spearman rank correlation between the maintenance data and associated metric values are shown in Table 1. CDM again outperformed the other metrics, followed by fan-in and response for a class. Again, the coupling-based metrics were better predictors of maintenance measures than intramodule metrics (like lines of code or cyclomatic complexity) or inheritance-based metrics (like the depth of the inheritance tree).

Coupling, Faults, Failures, and Maintenance

The obvious question is, why are coupling-based metrics in general (and CDM in particular) such excellent predictors of run-time faults and corrective maintenance? We believe that in most software products, a significant impediment to maintenance is the level of interconnection between modules; that is, the *coupling* between modules.

Consider an arbitrary module M. Let *V* denote the value of a coupling-based metric applied to M. Assume that this coupling-based metric incorporates all possible types of coupling between M and the rest of the product. Then, if a change is made outside M, *V* is a measure of the probability that the change outside M will require a corresponding change within M. In some cases, the need for this change within M will be revealed by the compiler or linker. However, other types of changes may be overlooked, especially when a medium- or large-scale product is developed by a team. Unless the required change is made, a run-time failure may eventually result. This is why coupling-based metrics in general are good predictors of run-time failures.

Turning now to maintenance: to fix a run-time failure requires corrective maintenance. Thus, a metric that can predict where a run-time failure is likely to occur will also be a good predictor of module-level corrective maintenance measures like the number of faults or time to repair faults.

On average, corrective maintenance occupies less than 20 percent of the total maintenance effort [1]. However, coupling also is a good predictor of all other forms of maintenance, including perfective and adaptive maintenance. The reason is that during maintenance the code is changed, and coupling is a measure of the likelihood that a change outside M will necessitate a change within M, irrespective of the reason for that change. That is, the value of a coupling-based metric is a measure of the probability that M must be changed as a consequence of any change to the rest of the product.

Conclusion

We have shown in a set of four case studies that coupling-based metrics like the coupling dependency metric (CDM) are powerful tools for measuring the impact of change. That is, coupling-based metrics are a good way to predict run-time failures and maintenance measures. ◆

About the Authors

Aaron B. Binkley is a senior software developer at Volpe, Brown, Whelan & Co., a San Francisco-based investment banking firm that serves companies in the areas of health care and technology. The research described in this article was performed while he was a graduate student at Vanderbilt University, where he obtained a master's degree in computer science. His primary research interests include software quality metrics and database performance tuning.

Volpe, Brown, Whelan & Co.
One Maritime Plaza
San Francisco, CA 94111
Voice: 415-274-7980
Fax: 415-434-4632
E-mail: aaron_binkley@vbwco.com

Stephen R. Schach is an associate professor of computer science at Vanderbilt University. He also is a software engineering consultant with over 25 years of computer experience, working with industry and giving seminars worldwide on the object-oriented paradigm and software metrics. He has published over 90 refereed technical papers. The fourth edition of his book, *Classical and Object-Oriented Software Engineering*, was published by McGraw-Hill in August 1998.

Vanderbilt University
Computer Science Department
Box 1679, Station B
Nashville, TN 37235
Voice: 615-322-2924
Fax: 615-343-5459
E-mail: srs@vuse.vanderbilt.edu
Internet: http://www.vuse.vanderbilt.edu/~srs/

References

1. Schach, Stephen R., *Classical and Object-Oriented Software Engineering*, 4th ed., McGraw-Hill, New York, 1999 (published August 1998 with a 1999 copyright).
2. Binkley, Aaron B. and Stephen R. Schach, "Validation of the Coupling Dependency Metric as a Predictor of Run-Time Failures and Maintenance Measures," *Proceedings of the 22nd International Conference on Software Engineering*, Kyoto, Japan, April 1998, pp. 452-455.
3. Details of all four case studies can be found in Technical Reports 97–03 through 97–06, Computer Science Department, Vanderbilt University, Nashville, Tenn., 1997 (see http://www.vuse.vanderbilt.edu/~srs/cdm).