An Evaluation of the Internal Quality of Business Applications: Does Size Matter?

Bill Curtis
CAST
PO Box 126079
Fort Worth, Texas 76126
+1-817-228-2994
curtis@acm.org

Jay Sappidi
CAST
373 Park Ave. South, 5th Floor
New York, NY 10016
+1-212-871-8330
j.sappidi@castsoftware.com

Jitendra Subramanyam
CAST
373 Park Ave. South, 5th Floor
New York, NY 10016
+1-212-871-8330
j.subramanyam@castsoftware.com

ABSTRACT
This study summarizes results of a study of the internal, structural quality of 288 business applications comprising 108 million lines of code collected from 75 companies in 8 industry segments. These applications were submitted to a static analysis that evaluates quality within and across application components that may be coded in different languages. The analysis consists of evaluating the application against a repository of over 900 rules of good architectural and coding practice. Results are presented for measures of security, performance, and changeability. The effect of size on quality is evaluated, and the ability of modularity to reduce the impact of size is suggested by the results.

Categories and Subject Descriptors
D.2.8 [**Metrics**]: Language Constructs and Features – *complexity measures, Performance measures, Product metrics.*

General Terms
Measurement, Reliability, Security, Verification.

Keywords
Software metrics, Software quality, Benchmarking, Static analysis, Internal quality, Maintainability Performance efficiency.

1. INTRODUCTION
The purpose of this study is to provide an objective, empirical foundation for evaluating the internal, structural quality of application software in both the public and the private sector. Such studies are needed to help IT organizations make visible the costs and risks hidden within their application portfolio, as well as establish a benchmark for making decisions about investments in quality. By internal quality, we are referring to the ISO 9126/25000 use of the term to mean the "static attributes of a software product that satisfy stated and implied needs when the software product is used under specified conditions" (ISO/IEC JTC1/SC7; 2001, 2010).

Internal quality involves the non-functional, internal properties of an application. It evaluates the engineering soundness of an application's architecture and coding, rather than the correctness with which the application implements functional requirements. Internal quality characteristics are critical because they are often difficult to detect through standard testing, yet they are frequent causes of operational problems such as outages, performance degradation, breaches by unauthorized users, and data corruption (Spinellis, 2006). Internal quality metrics have been shown to correlate with criteria such as maintenance effort and defect detection (Curtis, 1979a,b) The first enumeration of such quality characteristics was provided by Boehm and his colleagues at TRW in the 1970s (Boehm, 1976).

In this paper we summarize some of the key findings from our larger study of these applications (Sappidi, 2010). The data summarized here will provide empirical evidence regarding the following questions:

1. Which technology and industry segment scores highest for having secure applications?
2. Which quality characteristic displays the widest variation?
4. Which industry segment has applications that are the most difficult to change and does outsourcing affect these results?
5. Does size affect the quality of an application?
6. What are the some of the most frequent violations of good architectural and coding practice?

2. THE SAMPLE AND DATA
The data in this paper are drawn from 288 business applications, representing 108 M lines of code (3.4 M Backfired Function Points), submitted by 75 organizations for static analysis of their structural quality characteristics. The results of these analyses are captured in AppmarQ, a structural quality benchmarking repository maintained by CAST. These 75 organizations represent 8 industry segments including banking, insurance, telecommunications, energy, manufacturing, IT consulting, software ISVs, and government. These organizations are located

ICSE'11, May 21–28, 2011, Waikiki, Honolulu, HI, USA

primarily in North America, Europe, and India. The languages in which these applications are written include COBOL, Java EE, .NET, C, C++, and ABAP. The applications range from 10,000 to 5 million lines of code (LOC), with a mean of 374,220 LOC. Of these applications, 26% contain less than 50,000 lines of code, 32% fall between 50,000 and 150,000 thousand lines of code. A histogram of application size is displayed in Figure 1.

Since there is no rigorously developed population description of the global trove of business applications, it is impossible to assess the generalizability of these results. Although these results may not characterize the global population of IT business applications, they do emerge from what is believed to be the largest sample of applications ever to be statically analyzed and measured against internal quality characteristics across different technologies. Because of the selection process for submitting applications to deep structural analysis, we believe this sample is biased toward business critical applications.

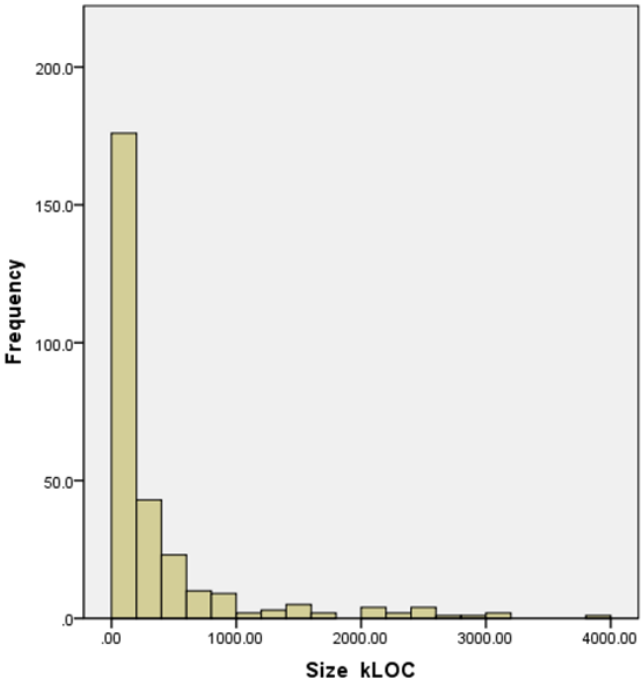

Figure 1. Distribution of the size in thousands of lines of code of business applications in the sample.

These business applications were analyzed using CAST's Application Intelligence Platform (AIP) which performs a static analysis of an entire application using over 900 rules to detect violations of good architectural and coding practice. These rules have been drawn from an exhaustive study of software engineering texts, online discussion groups focused on application best practices and defects, and customer experience drawn from defect logs and application architects.

The AIP begins by parsing an entire application at build time to develop a representation of the elements from which the application is built and its data-flows. This analysis is normally performed at during the build in order to analyze the source code at the application level across various language and technical platforms. The AIP includes parsers for the 28 languages listed in Figure 2 and a universal analyzer that provides and 80% parse for languages lacking a dedicated parser. Once parsed, AIP looks for violations of its architectural and coding rules and identifies the number of violations versus the number of opportunities for violations for each rule. The results are aggregated to the application level where each violation is weighted by its severity and summed into both a specific measure for a quality characteristic such as Changeability or Security, and a Total Quality Index that includes all violations. AIP provides a series of management reports and tools that guide developers in drilling down to specific violations that need remediation.

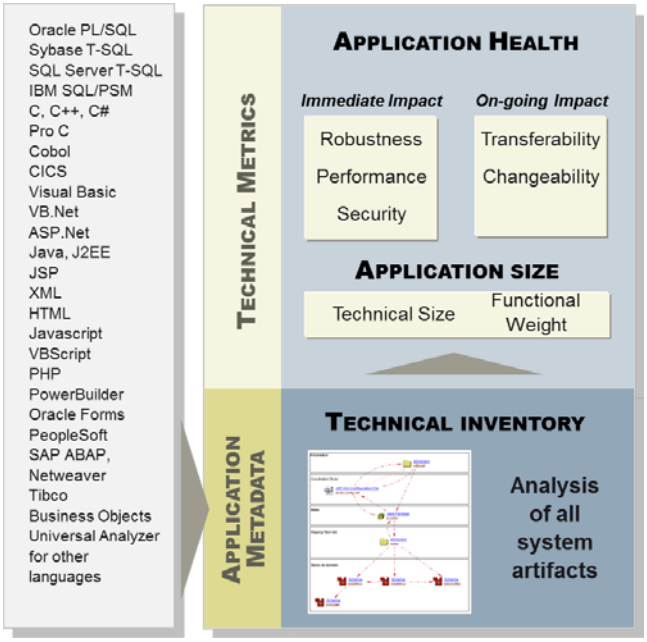

Figure 2. Application Intelligence Platform.

The application health factors in AIP were selected after reviewing ISO/IEC 9126. However, since the quality characteristics in this standard have not been defined down to a level that can be computed from the source code, some health factors names differ from 9126 based on the content analyzed and the meaningfulness of the names to users of the technology. In order to provide more standardization to computable measures of internal quality, the senior author is leading an industry effort called the Consortium for IT Software Quality (2010) sponsored by the Software Engineering Institute at Carnegie Mellon University and the Object Management Group that is developing standard definitions for automatable metrics. CISQ is intends to make these metrics as consistent as possible with the emerging ISO 25000 series of standards which will replace ISO 9126.

This paper will concentrate on results for three internal quality characteristics--security, changeability, and performance. The number of rules evaluated for each of these quality characteristics ranged between 176 and 506. Scores for each of these internal quality characteristics are aggregated from the component to the application level and reported on a scale of 1 (high risk) to 4 (low

risk), using an algorithm that weights the severity of each violation and its relevance to each individual quality characteristic.

3. SECURITY

Security scores evaluate the attributes of an application that affect its ability to prevent unauthorized intrusions and loss of confidential data. The distribution of Security scores across the sample is presented in Figure 3. The bi-modal distribution of Security scores suggests that applications there are group differences in the Security aspects of business applications. The distribution of scores on Security is wider than for any of the other quality characteristics, suggesting strong differences in attention to Security among different types of applications or industry segments.

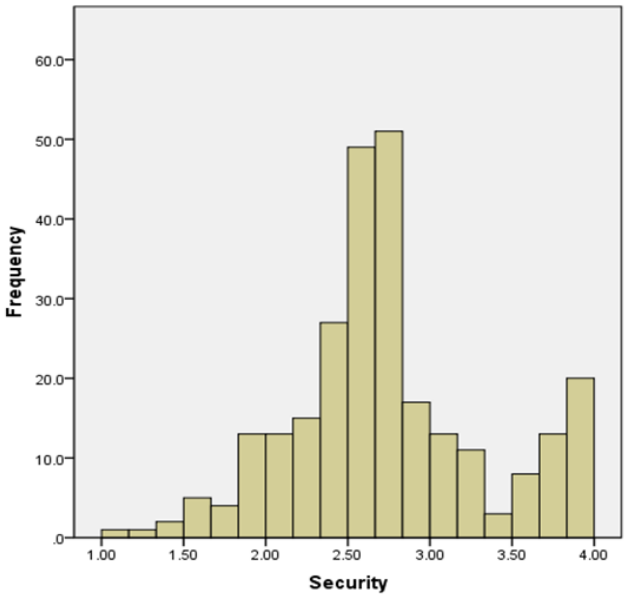

Figure 3. Distribution of Security scores.

Further analysis revealed that applications with higher Security scores were predominantly large mainframe and COBOL applications in the financial services sector (banking and insurance) which differed significantly from the other sectors in their scores ($p < .01$). In these industry segments high security for confidential financial information is mandated. Mainframe-based applications are also less exposed to the security threats that challenge Web-facing applications. Nevertheless, the lower Security scores for other types of applications are surprising. In particular, .NET applications received some of the lowest Security scores. These data suggest that the IT community's attention to security may be primarily driven by compliance regulations within industry segments.

4. CHANGEABILITY

Changeability scores evaluate the attributes of an application that make it easier and quicker to modify. Changeability scores presented in Figure 4 exhibit a bi-modal distribution indicating important differences in the maintainability of applications across the sample. Since the Changeability of an application is a critical driver of its cost of ownership, this distribution suggests large differences in the costs of ownership between applications with high and low Changeability scores.

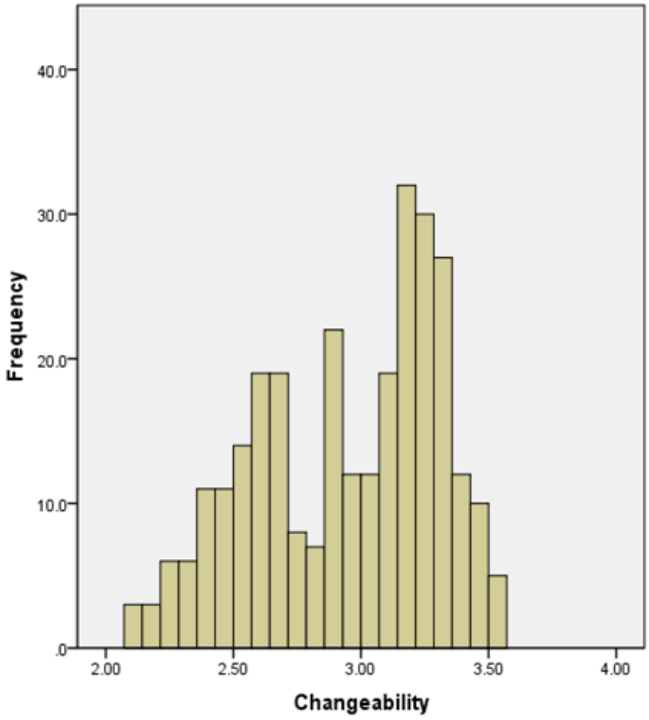

Figure 4. Distribution of Changeability scores.

We compared Changeability scores by industry segment. The results revealed that scores for applications in the public sector are significantly lower than those in other segments ($p < .01$). Our sample included government applications from both the US and EU. Although we do not have application cost data, these results suggest that government agencies are spending significantly more of their IT budgets on maintaining existing applications than on creating new functionality. Not surprisingly, in their 2009 IT Staffing & Spending Report Gartner reported that the government sector spends about 75% of its budget on maintenance, higher than any other segment.

Poor Changeability scores in the government sector may be partially explained by the high percentage of outsourced applications in the public as compared to the private sector. Seventy-five percent of the government applications in this sample were outsourced compared to 51% for the private sector. When government applications were removed from the sample, there was no significant difference between the Changeability scores for insourced and outsourced applications. The lower Changeability scores for government agencies may partially result from conditions unique to their acquisition. Multiple contractors working on an application over time, disincentives in contracts for building easily maintained code, and challenging acquisition practices are potential explanations for the Changeability scores for government applications.

5. PERFORMANCE

Performance scores evaluate the attributes that affect the responsiveness and efficiency of an application. Performance is

usually evaluated through measuring the execution behavior of the application dynamically. However, static analysis is able to detect violations of good coding practice, such as instantiating objects inside loops, that can cause performance problems as the database grows or the user increases. Scores for Performance were widely distributed as displayed in Figure 5, and in general were skewed towards better performance. However, Performance exhibited the widest variation of the distributions of any of the internal quality characteristics in this study.

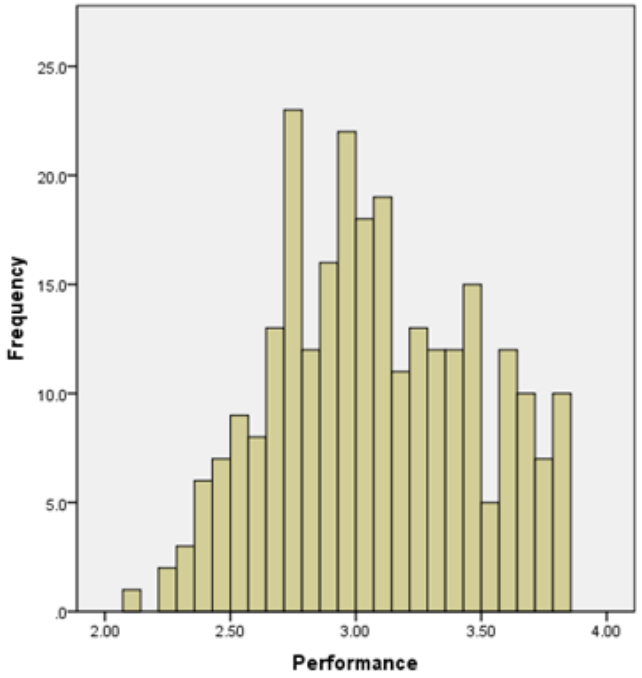

Figure 5. Distribution of Performance scores.

The skew toward higher scores observed for Performance might be explained by hypotheses that involve both technology and people factors. First, the availability and use of automated performance testing solutions has made performance problems easier to detect and address during development. Most modern testing platforms have embedded modules to test the performance of applications. Although they do not check the performance issues at the code level, they do highlight bottlenecks and attune developers to critical performance problems that could slow down an application or cause it to crash. Organizations using these tools would be expected to post high Performance scores. Second, performance problems are experienced immediately by users whose productivity it impacts. It is not uncommon for end-users to complain vociferously to the development team about slow performance, prioritizing the remediation of performance problems over other quality problems such as poor maintainability.

6. EFFECT OF SIZE ON QUALITY

The relationship between size and quality in this sample was investigated by combining the various quality characteristics into a combined Total Quality Index and correlating it with application size. The result for the sample as a whole contradicts the conventional wisdom that the quality of an application decreases as it grows larger. The correlation between the Total Quality Index and size was insignificant ($r < .01$), as were the correlations between each of the separate quality characteristics and size. Figure 6 presents the scatterplot between application size in thousands of lines of code and the Total Quality Index scores for the entire sample. This scatterplot was typical of the scatterplots observed for the individual quality characteristics.

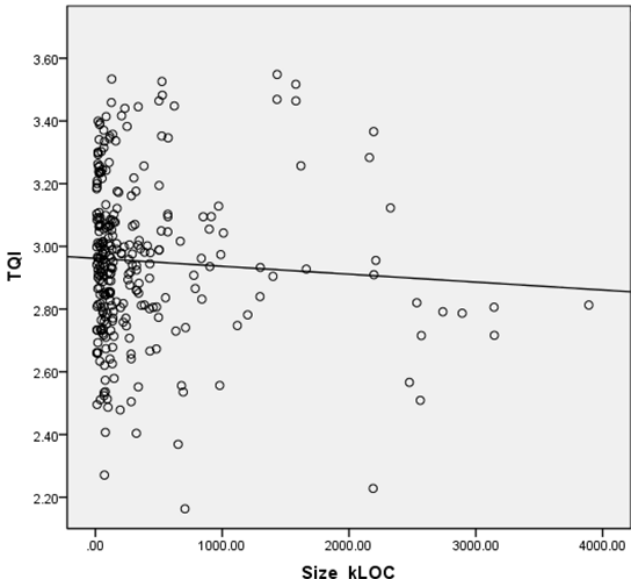

Figure 6. Scatterplot of the Total Quality Index with size.

However, when investigated within different language categories, the Total Quality Index did correlate negatively with the size of COBOL applications ($r = -.67$, $p < .01$). One possible explanation for this negative correlation is that COBOL does not encourage modularity, resulting in applications possessing many large and complex components. The design of more recent languages encourages modularity and other techniques that mitigate the affect of complexity as applications grow larger.

To investigate this explanation we compared the percentage of highly complex components across different languages. For the purpose of this analysis we defined a highly complex component as one whose Cyclomatic Complexity (McCabe, 1976) is 30 or greater. The percentage of highly complex components in COBOL applications is significantly higher than in other languages ($p < .01$). The median percent of highly complex objects for COBOL applications was just above 80%, while for all other languages it was below 20%. In particular, the median percent of highly complex components for the newer Object Oriented Technologies such as Java EE and .NET was below 5%, consistent with the objectives of object-oriented design.

7. TOP VIOLATIONS BY LANGUAGE

Table 1 presents the four most frequent violations of good architectural or coding practice by language that were rated 4 or higher on a 10 point scale of severity. High fan-out and high Cyclomatic Complexity were consistent problems across most of the languages encountered in this sample.

Table 1. Most frequent violations by language

Language	Violation
COBOL	Use of GOTO
	Use of PERFORM ... THROUGH | THRU
	Missing WHEN OTHER when using EVALUATE
	Components with High Cyclomatic Complexity
Java EE	Use of accessors to Private Fields
	Artifacts with High Fan-Out
	Unreferenced Methods
	Unreferenced Fields
.NET	Declaring Public Class Fields
	Artifacts with High Fan-Out
	Classes with a High Lack of Cohesion
	Artifacts with High Fan-In
C	Artifacts with High Fan-Out
	Large Functions - too many Lines of Code
	Functions with SQL statement including Subqueries
	Artifacts with High Cyclomatic Complexity
C++	Data Members that are not Private
	Artifacts with High Fan-Out
	Included files including other files
	Artifacts with High Cyclomatic Complexity
ABAP	Artifacts with a Complex SELECT Clause
	Artifacts with High Fan-Out
	Artifacts with High Cyclomatic Complexity
	Artifacts with High Essential Complexity

8. SUMMARY AND NEXT STEPS

Most benchmarking studies have been performed on project level data that only reported summary data for applications. This study suggests the possibility of establishing benchmarks for the internal quality of business applications. Although this sample is large, it will need to grow in order to provide sufficient cases to adequately characterize distributions for different languages and industry segments.

An obvious follow-on research topic is to relate the internal quality results of applications to their operational performance and business impacts. Unfortunately none of the companies that provided these applications had collected the type of data that would make this outcome research possible. We are working with several companies to collect the outcome data needed to validate the expected relations between internal quality characteristics and application risks and costs.

9. REFERENCES

[1] Boehm, B.W., Brown, J.R., & Lipow, M. (1976). Quantitative evaluation of software quality. *Proceedings of the 2nd International Conference on Software Engineering*. Los Alamitos, CA: IEEE Computer Society Press, 592-605.

[2] Consortium for IT Software Quality (2010). www.it-cisq.org.

[3] Curtis, B., Sheppard, S.B., Milliman, P., Borst, A., & Love, T. (1979a). Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. IEEE Transactions on Software Engineering, 5 (2), 96-104.

[4] Curtis, B., Sheppard, S.B., and Milliman, P. (1979b). Third time charm: Stronger prediction of programmer performance by software complexity metrics. Proceedings of the 4th International Conference on Software Engineering. Washington, DC: IEEE Computer Society, 356-360.

[5] ISO/IEC JTC1/SC7 (2010). *ISO 25000*. Montreal: École de technologie supérieure – Department of Software and IT Engineering, 1100 Notre Dame Ouest, Montréal, Québec Canada H3C 1K3.

[6] ISO/IEC JTC 1, SC 7 (2001). *ISO 9126*. Geneva: ISO.

[7] McCabe, T. (1976). A complexity measure. IEEE Transactions on Software Engineering, 2 (6), 308-320

[8] Spinellis, D. (2006). *Code Quality: The Open Source Perspective*. Boston: Addison-Wesley.

[9] Sappidi, J., Curtis, B., & Subramanyam, J. (2010). *CAST Worldwide Application Software Quality Study—2010*. New York: CAST Software.