CAST
Software Intelligence for Digital Leaders

CRASH Report

2017 Global Sample

CAST Research on Application Software Health

Global trends in Software Structural Quality

This is the fourth report produced by CAST on global trends in the structural quality of business application software. The data are drawn from CAST's Appmarq Repository which contains structural quality analyses of large, multi-layer, multi-language business applications. These reports highlight trends in five structural quality characteristics, or health factors: Robustness, Security, Performance Efficiency, Transferability and Changeability. The report identifies factors that affect the structural quality of applications.

Executive Summary

This is the fourth in a series of reports produced by CAST providing benchmarks on the structural quality of IT applications developed across the globe. This benchmark was developed from 1850 applications consisting of 1.03 BLOC (billions of lines of code), distributed across 329 organizations and 8 countries.

From a technology standpoint, this sample is predominantly made of Java-EE (40%), COBOL (22%), and .NET (10%), along with numerous other technologies such as ABAP, JSP, Oracle Server, PHP, etc. The sizes of these applications are spread widely, with more than 228 consisting of over a MLOC (million lines of code). The size of an application had negligible-to-no relation to its structural quality.

This report describes the effects of different development factors on structural quality. Structural quality differed across technologies with COBOL and Oracle Server generally having the lowest scores and JEE generally having the highest scores. Applications in CMMI Level 1 organizations had lower scores than those in Level 2 or 3 organizations. Factors associated with higher scores included use of hybrid development methods (combining up front design with rapid feedback from iterations), teams with 10 or fewer members, and greater than 5,000 end users.

Table of Content

About CAST's Appmarq Repository

Appmarq (www.appmarq.com) is CAST's application quality benchmarking repository. Over the past decade, CAST has analyzed the structural quality of thousands of business applications across different industries. These industries span telecommunications, insurance, financial services, national and local governments, retail, manufacturing, and many other IT-intensive sectors. The data have been collected and anonymized from organizations primarily across Europe, North America, and India.

The data in Appmarq provide unique insight into the structural quality trends of business application software. With Appmarq benchmarking services, companies can benchmark the structural quality of their applications against that of their peers in the same industry, or against applications developed with similar technologies, to identify strengths, weaknesses, and areas for improvement based on their structural quality characteristics.

The Appmarq Repository contains structural quality data on 2,200 applications from 400 organizations, totaling over 2 billion lines of code. The structural quality analyses were performed against over 1,200 rules of good architectural and coding practice in the areas of Robustness, Security, Performance Efficiency, Changeability, and Transferability.

2,200 applications

400+ organizations

2B+ lines of code

1,200+ quality rules

About the Authors

Bill CURTIS

Senior Vice President, Chief Scientist

Dr. Bill Curtis is an industry luminary who is responsible for influencing CAST's scientific and strategic direction, as well as for helping CAST educate the IT market on the importance of managing and measuring the quality of its software. He is best known for leading development of the Capability Maturity Model (CMM) which has become the global standard for evaluating the capability of software development organizations. Bill has a passion for software analysis and measurement because, after improving the development

Michael MULLER

Appmarq Product Owner at CAST

Michael Muller is a 15-year veteran in the software quality and measurement space. His areas of expertise include code quality, technical debt assessment, software quality remediation strategy and application portfolio management. As part of his scope, Michael manages the Appmarq product and the benchmark database, and is part of the CAST Research Labs analysis team that generates the industry-renowned CRASH reports.

Nagaraja S. ADIGA

Application Data Manager

Nagaraja Adiga has several years of experience in the field of Data Analytics. At CAST, Nagaraja works as an Application Data Manager for the Appmarq's repository.

Lev LESOKHIN

Executive Vice President, Strategy and Analytics

Lev Lesokhin, CAST EVP of Strategy and Analytics, is responsible for CAST's market development, strategy, thought leadership and product marketing worldwide. He has a passion for making customers successful, building the ecosystem and advancing the state of the art in business technology.

1 Introduction to CRASH Reports

This is the third CRASH report produced by CAST Research Labs on global trends in the structural quality of business application software. The data were drawn from CAST's Appmarq Repository which houses data collected during system-level structural analyses of large business applications. Structural quality refers to the engineering soundness of the architecture and coding of an application, rather than to the correctness with which it implements the customer's functional requirements. Structural quality is occasionally referred to as non-functional, technical, or internal quality.

1.1 Health Factors

Structural quality is measured by detecting violations of rules representing good architectural and coding practice in each of five areas called *health factors* defined in Table 1. Scores for health factors are computed on a scale of 1 (low/poor) to 4 (high/good) by analyzing the application to detect violations of over 1200 rules of good architectural and coding practice distributed across these five health factors as shown in Table 1.

Health Factor	Definition	Quality Rules
Robustness	Robustness measures the likelihood of outages, the difficulty of recovery and the possibility of data corruption linked to poor coding practices.	546
Security	Security measures violations of secure coding practices that allow unauthorized entry, deceptive interactions, theft of data, or breach of confidentiality.	333
Performance Efficiency	Efficiency measures the likelihood of potential performance degradation and inefficient use of resources such as processors, memory and networks linked to poor coding practices.	244
Changeability	Changeability measures the difficulty of modifying applications, adding features, correcting errors, or changing the application's environment.	603
Transferability	Transferability measures the difficulty of transferring work, or the difficulty of understanding the application and becoming productive in working with it.	601

Table 1. Health Factor definitions and number of rules

Evaluating an application for violations of structural quality rules is critical since they are difficult to detect through standard testing. Structural quality flaws are the defects most likely to cause operational problems such as outages, performance degradation, unauthorized access, or data corruption. CRASH reports provide an objective, empirical foundation for discussing the structural quality of software applications throughout industry and government.

1.2 Scoring the Health Factors

Scoring begins by creating a compliance ratio for each quality rule that compares the number of times the rule was violated to the number of opportunities in the source code where the rule could have been violated. This approach is similar but not identical to six sigma scoring schemes in that it identifies all structural elements within an application where various quality rules apply. In computing a score for each health factor, each quality rule contributes a specific weight based on its compliance ratio, its severity, and its impact on the specific health factor. Quality rules are divided between two levels of analysis:

- **Code-level rules:** These are rules evaluated at the code-unit level inside a class, method, sub-routine, module, or other foundational code unit. Violations at this level typically involve coding hygiene issues and simple defects.
- **System-level rules:** Architectural rules whose evaluation involves multiple components often spread across several layers of the application. These violations are difficult or impossible to detect through ordinary testing. They are typically the culprits that cause outages, security breaches, data corruption, and difficulty in sustaining or scaling the application.

Quality scores are first computed at the code-unit level and then aggregated to the application level. The health factor scoring scheme was designed to accentuate operational risk and cost. Thus, a single violation with the highest severity rating can have as much impact on lowering a Health Factor score as a collection of lower severity violations.

2 The CRASH Sample

The CRASH data drawn from the Appmarq Repository include 1,850 applications submitted by 329 organizations for analysis. In the aggregate, these applications totaled 1.03 BLOC (billion lines of code). The submitting organizations are located primarily in Continental Europe (France, Belgium, Italy, Germany, and Spain), the United Kingdom, North America (the United States and Canada) and India. The sample includes applications written primarily in COBOL, Java-EE, .Net, Oracle Server, and other technologies such as PHP, C/C++, PowerBuilder, C#, and Visual Basic.

2.1 Application Size

The lower threshold for accepting applications into this CRASH sample was 10 KLOC (kilo or thousand lines of code). Figure 1 divides the sample into size categories wherein 26% of the applications contain between 10 KLOC and 50 KLOC, 32% contain between 50 KLOC and 200 KLOC, 30% contain between 200 KLOC and 1 MLOC, and 12% contain more than 1 MLOC, including 28 applications over 5 MLOC and 7 over 10 MLOC.

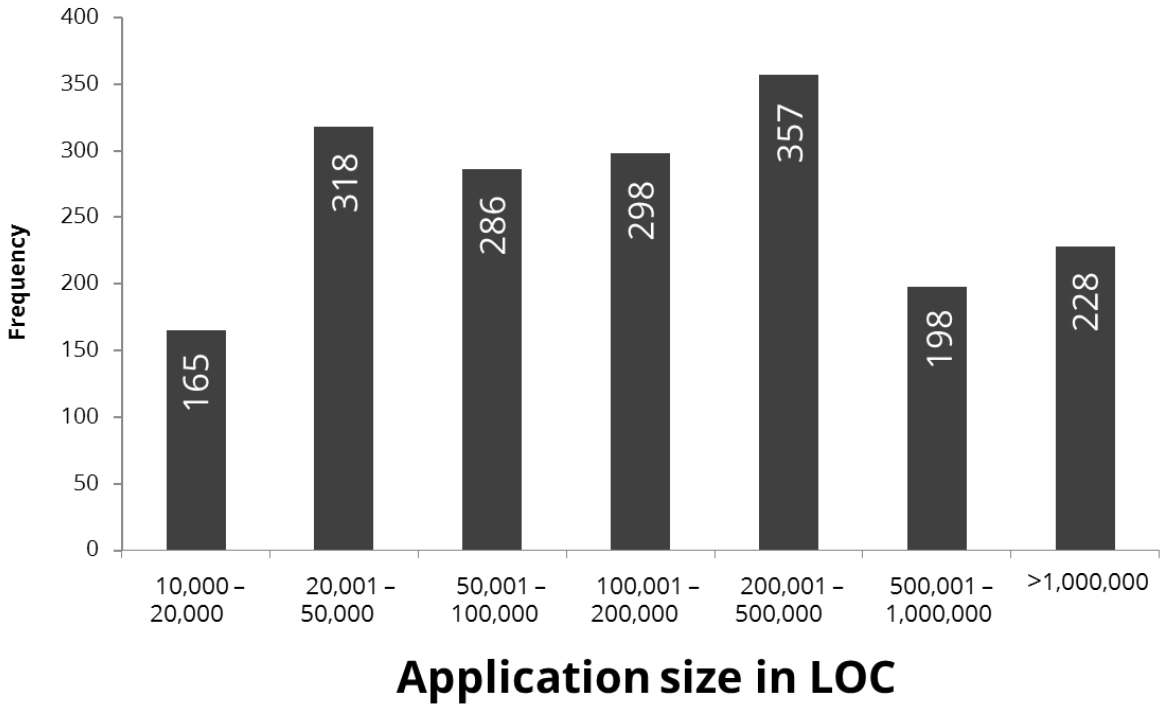


Figure 1. Distribution of application sizes for the full sample

Distributional statistics for the full CRASH sample are presented in Table 2. The mean of 554,783 LOC is almost 4 times larger than the median size of 146,359 LOC, indicating the sample is extremely positively skewed by the applications over 5 MLOC, the largest of which contained 52,773,950 lines of code. The extreme positive skewness of this sample is further revealed by the standard deviation being three times larger than the mean score and almost 13 times larger than the median score. In addition, the mean size is above the third quartile, indicating the heavy effect of a few extremely large programs. The 52 MLOC application was written in RPG and will not appear in many of the analyses in this report since they focus on factors affecting the most frequently used technologies.

Minimum	1st Quartile	Median	Mean	3rd Quartile	Maximum	Std. Dev.
10,039	47,243	146,359	554,782	456,186	52,773,950	1,867,526

Table 2. Distributional statistics for lines of code for the full CRASH sample

The sample is widely distributed across size categories and appears representative of the various types of applications in business use. However, the applications submitted for structural analysis and measurement tend to be business-critical systems, so we do not claim that this sample is statistically representative of the population of business applications globally. Rather, this sample appears most representative of the mission-critical subset of business applications. Not surprisingly, a sample weighted toward business-critical applications would contain some of the largest applications in an organization's portfolio.

2.2 Technology

As presented in Figure 2, COBOL and Java-EE (JEE) are the most frequently used technologies in the CRASH sample, with 40% and 22% of applications having been developed in these technologies respectively. The large sample of COBOL applications results from the large proportion of applications from the financial services and insurance industries. Other technologies involved in more than 40 applications in the CRASH sample included .NET, ABAP, Oracle Server, C, and C++. Thirteen percent of the sample was distributed across a range of technologies, of which C# and JSP alone account for over 30 applications each.

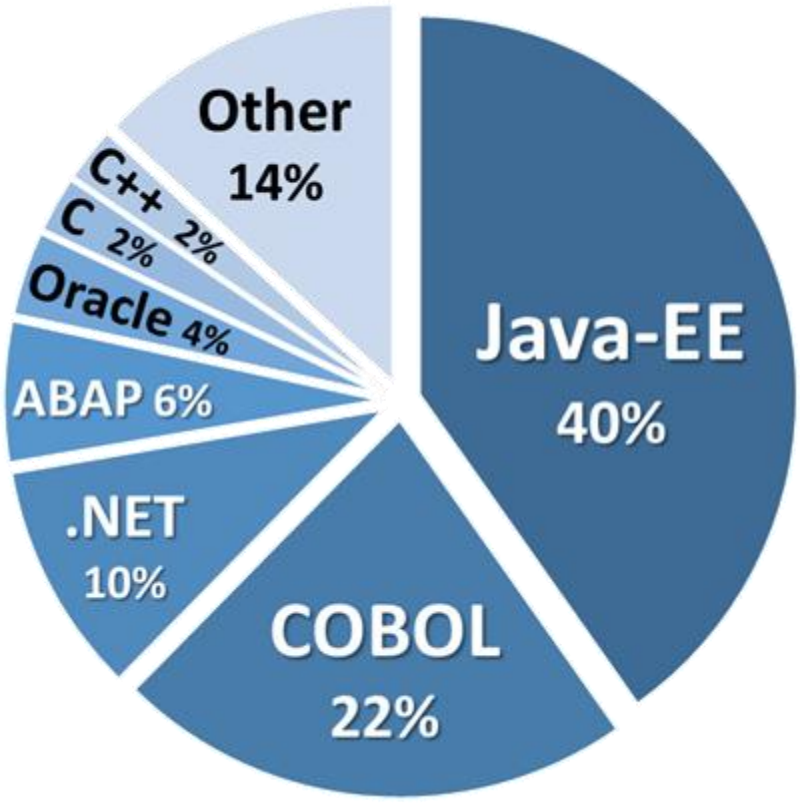


Figure 2. Distribution of applications across technologies in the CRASH sample

As evident in Figure 3, there are differences in how the technologies are distributed across size categories. Data are presented for the four technologies with the largest number of applications—Java-EE, COBOL, .NET, and ABAP. For applications with no more than 20,000 LOC, the highest percentages are in Java-EE and .Net. For applications with more than 1 MLOC, COBOL and ABAP have the highest percentages. In fact, the percentage of ABAP applications tends to increase with their size, to a high of 24% for applications with over 1 MLOC. Most Java-EE applications fall into the range of 20 KLOC to 500 KLOC. The percentage of .NET applications increases up to 500,000 and declines thereafter.

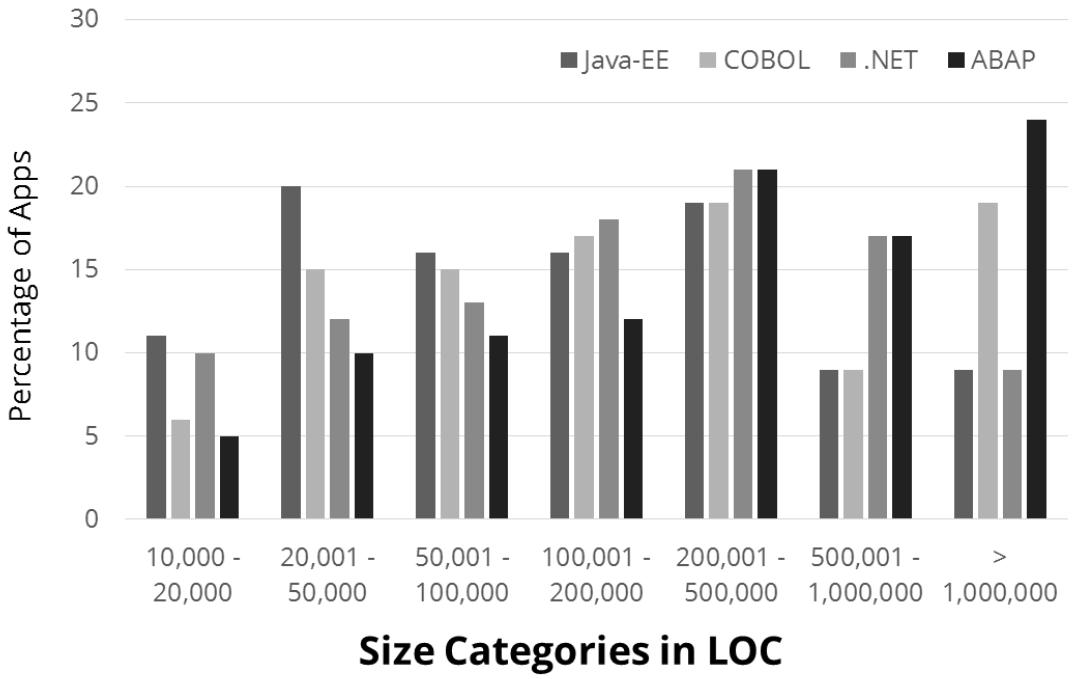


Figure 3. Distributions across size categories for Java-EE, COBOL, .NET, and ABAP

Descriptive statistics for these applications are presented in Table 2. The mean size of applications in the full sample is 554,782 lines of code. However, since the median size is 146,359, the sample is positively skewed by the extreme sizes of several very large applications. The largest average sizes are in COBOL and ABAP applications, while the smallest average sizes are in Oracle Server, .NET, and Java-EE. The two extreme outliers in the sample are written in RGP300 and consist of approximately 33 and 52 MLOC.

2.3 Industry Segment

The 2016 CRASH sample includes applications from 329 companies in 12 industry segments. These applications are displayed in Figure 4 by the number of applications and the number of contributing organizations in each segment. One third of the sample comes from financial services, 13% comes from insurance, and 12% from telecommunications. All other industry segments account for less than 10% of the sample. Applications that could not be classified by industry or were in segments with fewer than 10 applications are included in the category labelled 'Other'.

The distribution of languages across applications within each industry segment is presented in Table 3. Use of Java-EE, the most common technology in the CRASH sample, is widely distributed across all industry segments. Conversely, COBOL, the second most common

technology in the CRASH sample is primarily used in financial services and insurance. .NET and Oracle Server were used in all industry segments. ABAP was primarily used in manufacturing and utilities. Having been developed in Bell Labs, it is not surprising that the highest use of C and C++ was in telecom and the next highest use of C was in utilities.

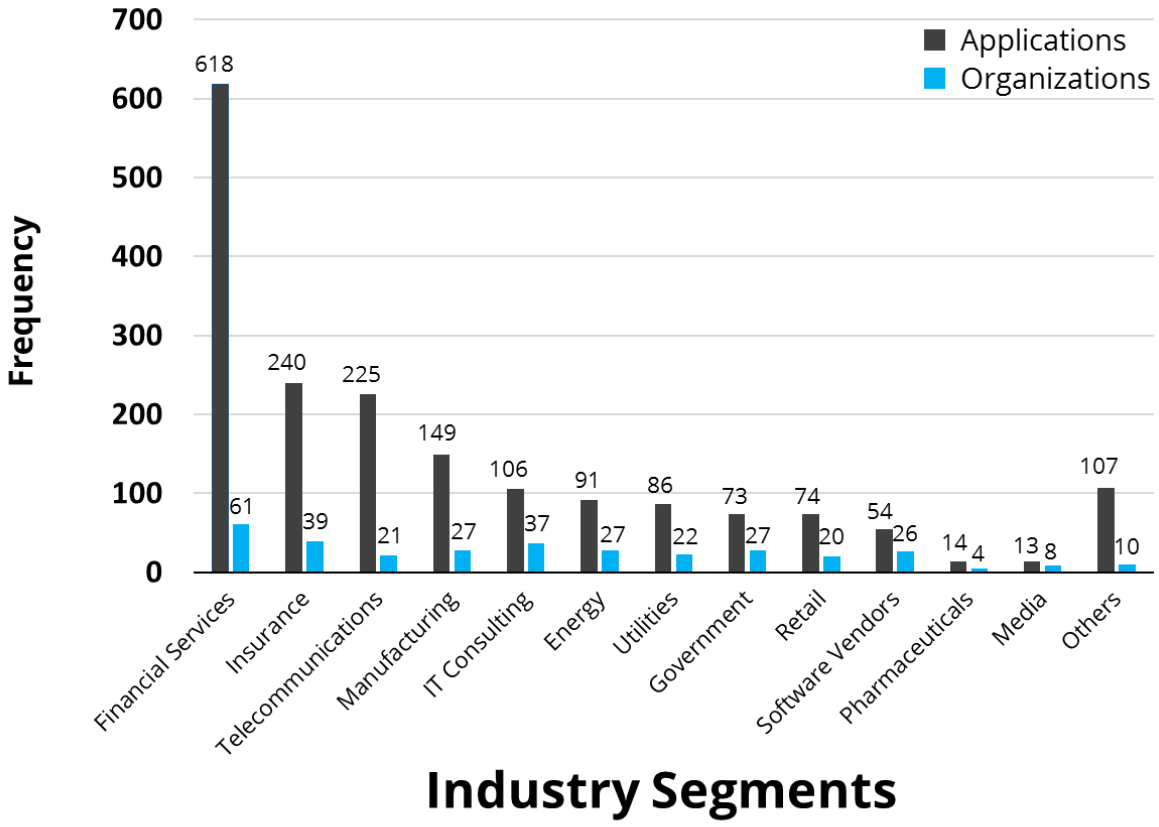


Figure 4. Frequency of applications and organizations in each industry segment

Industry	Java-EE	COBOL	.NET	ABAP	Oracle Server	C	C++	Total
Financial	220	253	21	5	20	3	8	**530**
Insurance	64	110	30	1	8	4	1	**218**
Telecom	122	3	24	11	9	14	13	**196**
Manufacture	56	6	26	40	3	2	1	**134**
IT Consulting	45	6	18	6	8		6	**89**
Energy	26	1	15	23	1	1	5	**72**
Utilities	39	11	1	5	3	10	2	**71**
Government	45	2	5	9	2	2		**65**
Retail	35	3	9	10	4	2	1	**64**
Software ISV	25	1	17		1		2	**46**
Pharma	7		2	2	1			**12**
Media	8		2		3			**13**
Total	**692**	**396**	**170**	**112**	**63**	**38**	**39**	**1510**

Table 3. Frequency of applications by technology within industry segments

2.4 Type of Application System

The type of system was listed for 1,388 applications. Table 4 provides the total frequencies of system types and how these types are distributed by technology. Frequencies are not displayed for technologies with too few applications, so the technology columns will not sum to the frequency for the total sample. The most frequent system types in the CRASH sample were Core Transaction Systems and Enterprise Resource Planning systems. Core Transaction Systems were most frequently written in COBOL, and to a lesser extent Java-EE. Enterprise Resource Planning systems were often written in Java-EE, and of course ABAP. Customer-facing websites were most frequently written in Java-EE, but surprisingly 18% were written in COBOL. Enterprise Portals, Customer Resource Management, and Analytics systems were primarily written in Java-EE.

App Type	Java-EE	COBOL	.NET	ABAP	Oracle	C	C++	Total
Core Trans	152	300	16	1	16	1	14	**551**
ERP	125	19	32	78	24	20		**352**
Website	80	29	25	1	4	6	4	**162**
Ent. Portal	92	4	18	12	1	3	3	**161**
CRM	34	4	8	13	5	5	1	**91**
Analytics	27	5	16	1	4	1	6	**71**
Total	**510**	**361**	**115**	**106**	**54**	**36**	**28**	**1388**

Table 4. Frequency of application type by technology

Table 5 presents the frequencies of each system type within each industry segment. Applications with no listed industry segment are not included, so columns will not sum to the total frequency for each system type. Core transaction systems are concentrated most heavily in the financial services and insurance industries. ERP systems are widely spread across all industry segments, but are most heavily concentrated in manufacturing, telecom, and utilities. Customer-facing websites are most heavily concentrated in financial services insurance and telecom, while internal portals are most frequent in financial services, telecom, and manufacturing. Analytics systems were most frequent in financial services and telecom.

Industry	Core Trans	ERP	Website	Portal	CRM	Analytics	Total
Financial	366	54	30	47	7	18	**522**
Insurance	102	14	26	8	10	5	**165**
Telecom	12	50	21	25	18	18	**144**
Manufacture	1	62	1	25	7	2	**98**
IT Consulting	3	29	5	12	13	7	**69**
Energy	25	23	4	8	7	5	**72**
Utilities	4	40	1		5	3	**53**
Government	5	12	13	4		6	**40**
Retail	12	24	9	7	12	2	**66**
Software ISV	14	10	15	5	2	1	**47**
Pharma	1	6	3	1	1		**12**
Media	1	3	6	1	1	1	**13**
Total	**546**	**327**	**134**	**143**	**83**	**68**	**1,301**

Table 5. Frequency of application type by industry segment

Table 4 demonstrated there is a strong relationship between system type and the technology used to develop it. Table 5 shows a relationship between industry segment and system types. These observations suggest that differences between industries in structural quality may be more attributable to the mix of application types than to the qualities of the industry.

3 Structural Quality

3.1 Global Trends

Comparisons of scores across Heath Factors is inexact because different numbers of rules were defined for each. Since the scoring system partially mitigates these differences, we can be gain some insight into the general structural quality trends. The mean scores for the five Health Factors are displayed for the full CRASH sample in Table 6. Means for the operational risk factors of Robustness, Security, and Performance Efficiency were higher than those for the cost/maintainability issues of Changeability and Transferability.

Sample	#	Robust	Security	Perform	Change	Transfer
Total	**1850**	3.20	3.22	3.15	3.03	3.00

Table 6. Mean Health Factor scores for the global sample

Box and whisker charts comparing the distributions of technology scores on the five Health Factors are presented in Figure 5 along with an explanation of how to these charts should be interpreted in this report. While the pattern of means is evident, the most striking observation from this chart is the large variance in scores. While the interquartile ranges varied from 0.4 to 0.6 in width, the 2 sigma limits above and below each mean are quite large. Even more dramatic, the range of scores for each Health Factor is extended by outliers. Although the means for Security and Performance Efficiency were high, the variance in their scores was quite large, as evidenced by their larger interquartile range (the barrel from the 1st to 3rd quartile that contains 50% of the data points for that category). While some applications achieved the highest scores attainable on Security and Performance Efficiency, several applications achieved scores that were low on these Health Factor scales.

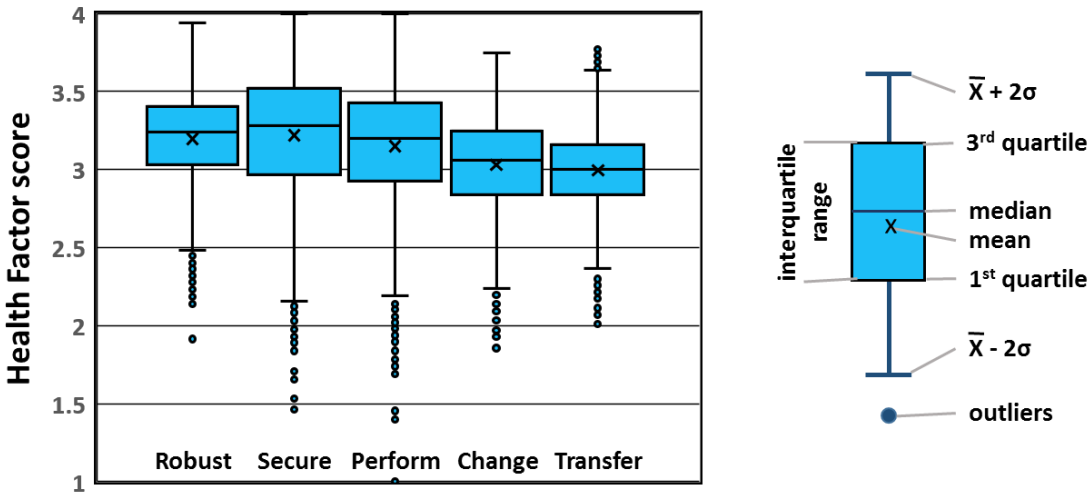


Figure 5. Distribution of Health Factor scores in the total sample

3.2 Relationships among Health Factors

Table 7 presents the strength of relationships among pairs of Health Factors. For easier comparison, correlation coefficients were squared into the percentages of common variation in scores shared by each pair. Since the pattern of relationships differs between technologies, results are presented for COBOL and Java-EE.

In COBOL, there are two clusters of Health Factors that were confirmed in a factor analysis. The first cluster includes the operational risk factors of Robustness, Security, and Performance Efficiency, where the relationships accounted for between 41% and 76% of the variation in scores. The second factor includes the cost/maintainability factors of Changeability and Transferability, which shared 49% of the variation in their scores. The relationships between the Health Factors in these two clusters never accounted for more than 6% of the shared variation, except for the moderate relationships of Robustness with the Changeability and Transferability.

The pattern of relationships is quite different in Java-EE and .NET. With the exception of Security, moderate-to-strong relationships existed between all of the Health Factors where the shared variation ranged from 29% to 58% in Java-EE and from 25% to 40% in .NET. Security, which had smaller relationships with Performance Efficiency and Transferability in Java-EE and with Changeability and Transferability in .NET.

Health Factors	Robust	Secure	Perform	Change	Transfer
COBOL					
Robustness		76%	41%	12%	25%
Security	76%		45%	4%	5%
Performance	41%	45%		2%	6%
Changeability	12%	4%	2%		36%
Transferability	25%	5%	6%	36%	
Java-EE					
Robustness		52%	29%	58%	35%
Security	52%		14%	34%	15%
Performance	29%	14%		30%	36%
Changeability	58%	34%	30%		49%
Transferability	35%	15%	36%	49%	
.NET					
Robustness		40%	25%	33%	43%
Security	40%		25%	12%	7%
Performance	25%	25%		40%	27%
Changeability	33%	12%	40%		37%
Transferability	43%	7%	27%	37%	

Table 7. Percentages of variation accounted for between Health Factors

The difference in these patterns of relationships may result from fundamental changes in programming style from older technologies such as COBOL to more modern technologies such as Java-EE. In COBOL, modules tend to be quite large—often 10 times larger than those in Java-EE and other more modern languages. Further, since they are quite often used in transaction processing systems on mainframes, they have been optimized for reliability,

speed, and security. Such use often comes at the expense of ease of change and understanding. Modern technologies such as Java-EE have been constructed to support object-oriented programming where components such as classes and methods are much smaller than COBOL modules. When properly engineered, smaller components are easier to understand and change, and thus the relationships between operational risk factors and cost/maintainability factors are stronger, as seen in Table 7.

3.3 Relations of Health Factors to Size

Correlations between the size of applications and their Health Factor scores are presented in Table 8 for the total CRASH sample and separately for each technology with at least 40 applications. Although some of the correlations are significant because of the large sample size, they are uniformly low. Only five of the correlations indicated that size accounted for as much 4% of the variation in Health Factor scores, and none accounted for more than 6%.

Health Factor	Total	J-EE	COBOL	.NET	ABAP	Oracle	C	C++
Robustness	**-0.08**	-0.12	-0.13	-0.14	-0.19	-0.07	-0.01	-0.06
Security	**-0.05**	-0.12	0.09	-0.21	-0.15	-0.01	0.11	-0.01
Performance	**-0.17**	-0.17	-0.19	-0.21	-0.16	0.02	0.01	0.06
Changeability	**-0.14**	-0.14	-0.20	-0.04	-0.15	0.12	0.15	-0.19
Transferability	**-0.02**	-0.01	-0.04	-0.24	-0.20	0.00	-0.02	-0.08

Table 8. Correlations of Size with Health Factors in the total sample and by technology

To help visualize these relationships, Figure 6 presents the scatterplot for the largest correlation in Table 7, which is -0.24 and which indicates a small negative relationship between lines of code and Transferability. Visually the variation appears random around the mean and even the effect of the application being large, with over 2.7 MLOC does not change the small negative relationship. Scatterplots for the other correlations appear similar to Figure 7, indicating that size has a very weak relation with the structural quality of applications. This observation was replicated across technologies.

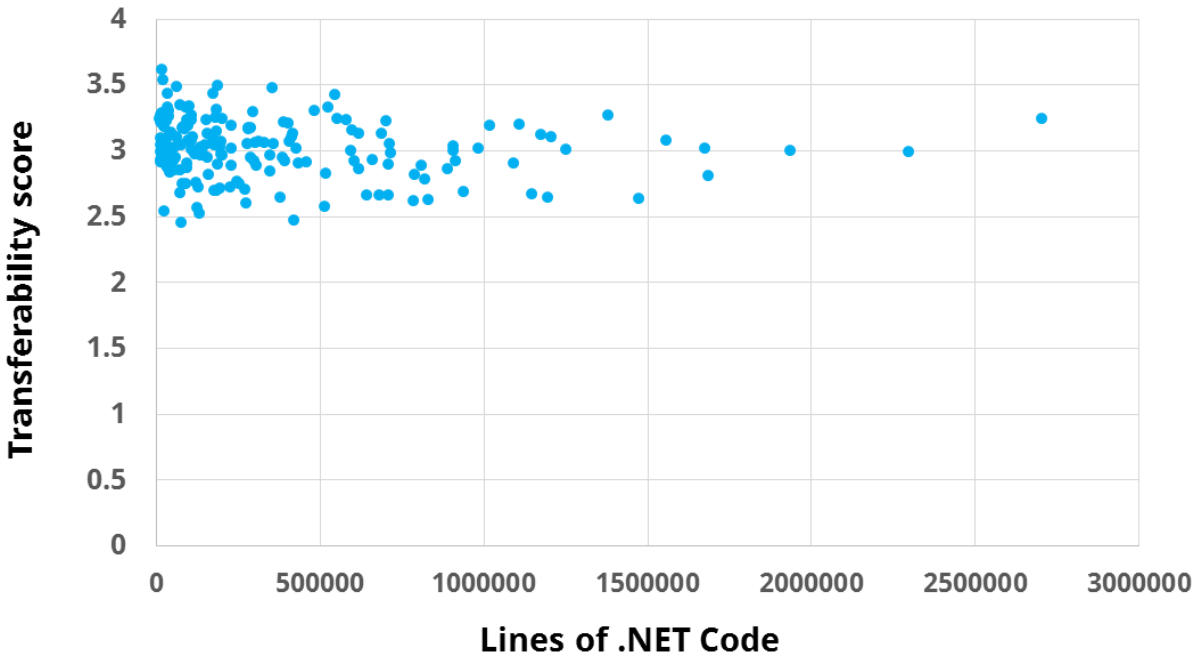


Figure 6. Scatterplot of the relationship between size and Transferability in .NET

3.4 Violation-Level Observations

Violation-level data were available from a maximum of 1,353 applications. As presented in Table 9, just over 97 million violations of quality rules were detected in these applications. However, there were over 1.3 billion opportunities in the source code where the rules could have been violated. Thus, source code structures were found to be in non-compliance with relevant quality rules in only 7.4% of the instances in which they applied.

Category	Apps	Violations	Opportunities	% Non	Sigma
All violations	**1353**	97,684,758	1,315,818,839	7.40	2.94
Violation level					
System level	**1247**	1,172,064	19,321,892	6.07	3.05
Code-unit level	**1353**	96,512,694	1,296,496,947	12.07	2.94
Criticality					
Critical	**1217**	1,516,696	128,131,744	1.18	3.76
Non-critical	**1353**	96,168,062	1,187,687,095	8.10	2.90
Health Factor					
Robustness	**1353**	27,451,312	741,120,828	3.70	3.29
Security	**1341**	9,501,785	561,489,255	1.69	3.62
Performance	**1352**	5,129,628	231,927,082	2.21	3.51
Changeability	**1353**	58,642,855	630,392,741	9.30	2.82
Transferability	**1353**	80,127,497	601,483,347	13.32	2.61

Table 9. Non-compliance and process sigma levels for quality rule violations

System-level rules were violated in only 6.07% (1,172,064 violations) of the more than 19 million total opportunities where system level rules applied in these applications. Recent research found that system-level violations often account for approximately 10% of total violations. However, these system-level violations can account for over 50% of corrective maintenance costs.[2]

Of the 97 million violations of quality rules, only 1.6% are rated as critical. However, the quality rules underlying these critical violations exhibited a much higher compliance rate (99%) than the quality rules for non-critical violations (92%). Among the Health Factors, compliance was higher for the operational risk factors of Robustness, Security, and Performance Efficiency than it was for the compliance ratios for the cost factors of changeability and transferability. In general, compliance with quality rules was greatest

[2] Li, et al. (2011). Characteristics of multiple component defects and architectural hotspots: A large system case study. *Empirical Software Engineering*, 16 (5), 667-702

where the risk was highest (i.e. with system-level violations, critical violations, and violations related to business risk Health Factors).

Table 7 also reports process sigma levels computed as defects per million opportunities. The process sigma level is computed as a standard score representing the number of standard deviations out on a distribution of defects per million opportunities. The popular six sigma notation indicates that at 6 standard deviations out, there are only 4 defects per million opportunities. In these data, the highest scores are between 3 and 4 sigma. In most cases, software technology and practice are not sufficiently advanced to achieve six sigma levels. Based on these data, most IT organizations should strive to achieve scores between 3.5 and 4 sigma for business- or mission-critical applications.

3.5 Case Study: Code unit vs system-level violations

In Table 9, system-level violations account for only 6.07% of the total violations analyzed in this CRASH Report. However, industrial experience and research have demonstrated that they have a disproportionate effect on operational risks and corrective maintenance costs. The distinction between system and code unit-level violations is important since most static analysis technologies only detect code unit-level violations. Code unit-level violations most often involve the Changeability and Transferability weaknesses that affect ease of maintenance and not operational risks such as Robustness and Security.

This system vs. code unit-level distinction is highlighted in a case study from one of the organizations contributing applications to Appmarq. A large consumer brand with thousands of developers, multiple ADM vendors, and hundreds of large applications embarked on a program to manage code quality. They focused first on code quality at the code unit level. After running the unit-level quality program for two years, they initiated system-level analysis. When they measured the quality of their business-critical applications, they found that, as shown in Figure 7, their earlier remediation efforts had primarily achieved more maintainable software. Health Factor scores for operational risk factors of Robustness, Security, and Performance Efficiency were significantly lower. In fact, all Health Factor scores were significantly lower than those reported for the full Appmarq sample in Table 6. In particular, the scores for the operational risk factors of Robustness, Security, and Performance Efficiency were far lower than those of the global sample.

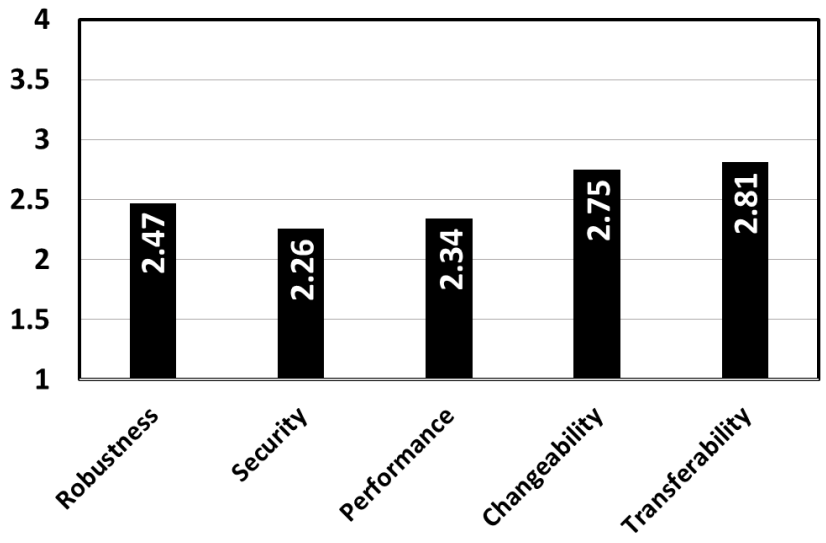


Figure 7. Case study Health Factor scores

These results were not surprising because system-level issues are more often related to Robustness, Security, and Performance Efficiency. Code unit-level quality analyses most often detect hygiene issues that affect maintainability, but cannot detect the flawed interactions among components across the system that affect reliability, security, and performance. Although system-level violations account for a small percent of the total violations detected, their detection is critical to creating a full account of the flaws that potentially affect reliability, security, and performance during operations.

4 Factors Affecting Structural Quality

This section presents analyses of various factors that can influence structural quality. As will be demonstrated in Section 4.1, there are significant differences among the technologies used in developing applications in the CRASH sample. Since different numbers of quality rules were defined for each technology, comparisons among Health Factor scores across technologies are not exact. A few comparative observations will be made among technologies, but in later sections the analyses will be conducted within single technology categories. Consequently, analyses of factors that influence structural quality will only be conducted within technologies when there are at least 20 applications in each category of the factor under analysis.

4.1 Industry Segment

Differences among mean scores on each Health Factor for the ten industry segments with at least 40 applications in the CRASH sample were compared in an analysis of variance (ANOVA). Statistically significant differences ($p < .001$) in means were observed between industry segments for all five Health Factors. However, as reported in Table 10, differences among industry segments never accounted for more than 4% of the variation in scores. Table 10 summarizes the mean scores for each of the industry segments on each Health Factor. The following paragraphs describe in greater depth the distribution of scores and which industry segments were responsible for the significant differences.

Industry	#	Robust	Security	Perform	Change	Transfer
Financial	618	3.13	3.15	3.05	2.99	2.97
Insurance	240	3.19	3.28	3.14	2.98	3.04
Telecom	225	3.21	3.17	3.22	3.08	2.98
Manufacture	149	3.17	3.23	3.10	3.01	3.00
IT Consulting	106	3.26	3.21	3.28	3.06	3.02
Energy	91	3.26	3.32	3.12	3.04	3.03
Utilities	86	3.24	3.34	3.19	3.03	2.94
Government	73	3.35	3.46	3.39	3.25	3.08
Retail	74	3.22	3.17	3.21	3.06	3.00
Software ISV	54	3.29	3.21	3.18	3.02	3.00
% variance explained		4%	3%	4%	4%	2%

Table 10. Means and variance explained in Health Factors by industry segment

***Robustness*.** Differences among the ten industry segments accounted for only 4% of the variation in Robustness scores. Post hoc tests revealed the primary differences occurred between government, which earned the highest mean score and financial services, which earned the lowest mean score. However, as we can see in Figure 8, financial services displayed wide variation, containing both the highest and lowest Robustness scores. In fact, the median for financial services was closer to that of other industries, indicating that the greater proportion of scores at the lower end of the financial services distribution is primarily responsible for the Robustness mean being lower than that of other industries.

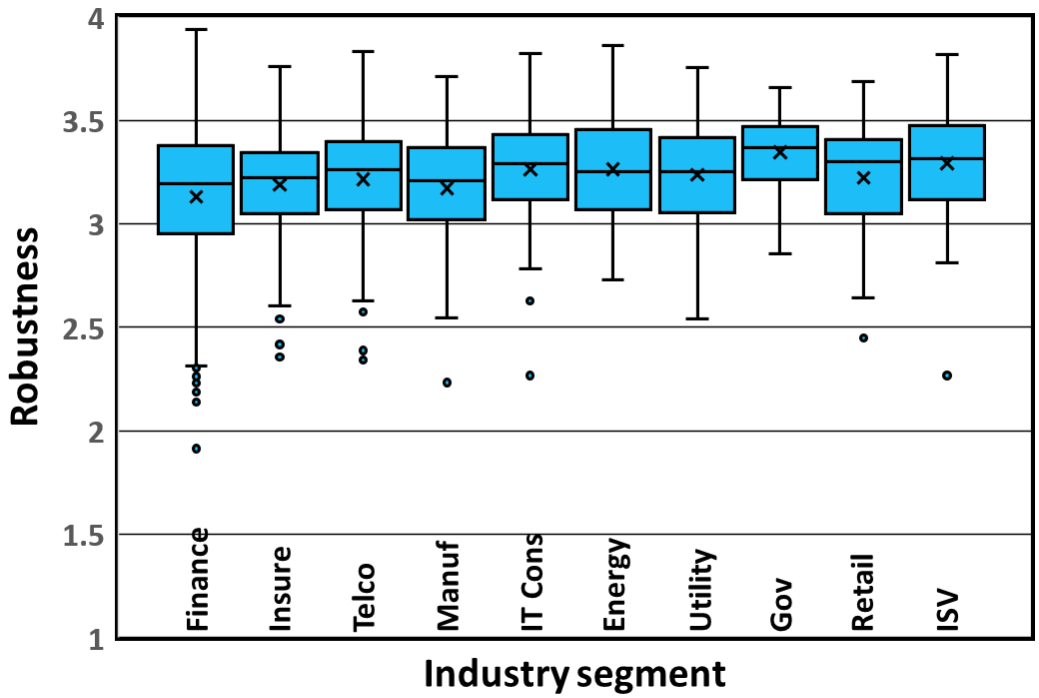


Figure 8. Distribution of Robustness Scores by Industry Segment

Security. Differences among the ten industry segments accounted for only 3% of the variation in Security scores. Post hoc tests revealed the primary differences occurred between government with the highest mean score with the lowest variance, while financial services, telecommunications, and retail posted the lowest mean scores. However, as we can see in Figure 9, there were wide variations in Security scores in many industries—especially in financial services, which contained both the highest and lowest Security scores.

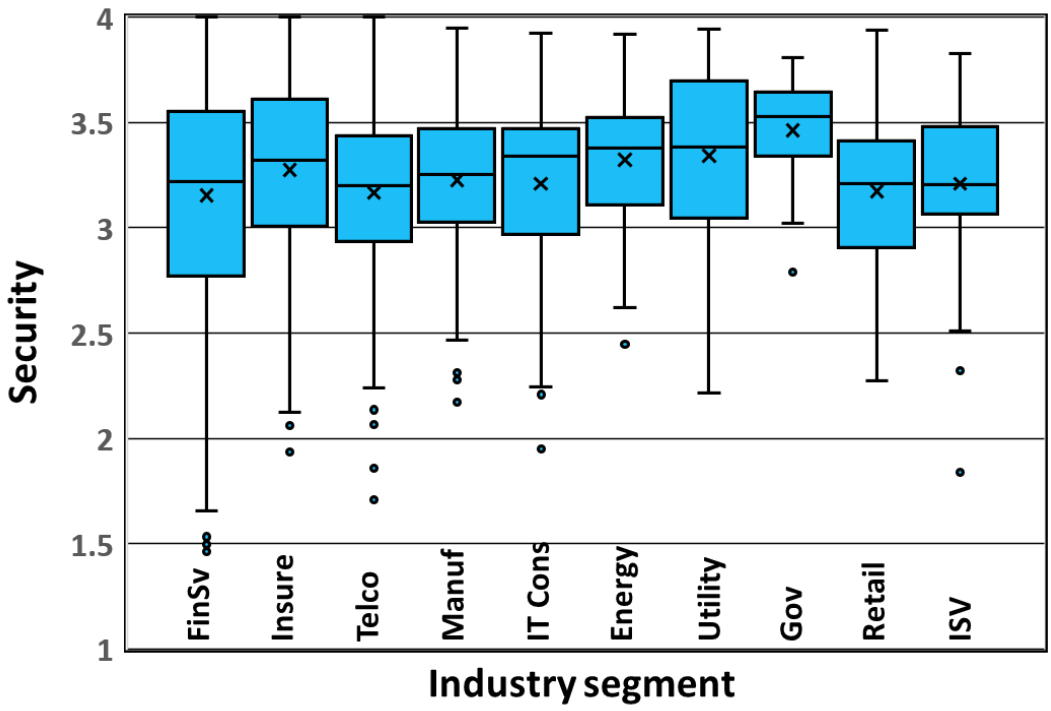


Figure 9. Distribution of Security Scores by Industry Segment

Performance Efficiency. Differences among the ten industry segments accounted for only 4% of the variation in Performance Efficiency scores. Post hoc tests revealed that the primary differences occurred between government and IT consulting, which had the highest mean scores, and financial services, manufacturing, energy, and insurance, which had the lowest mean scores. However, as can be seen in Figure 10, there were wide variations in Security scores in many industries with the largest being observed in financial services. Several industries earned very high scores, while utilities earned the lowest.

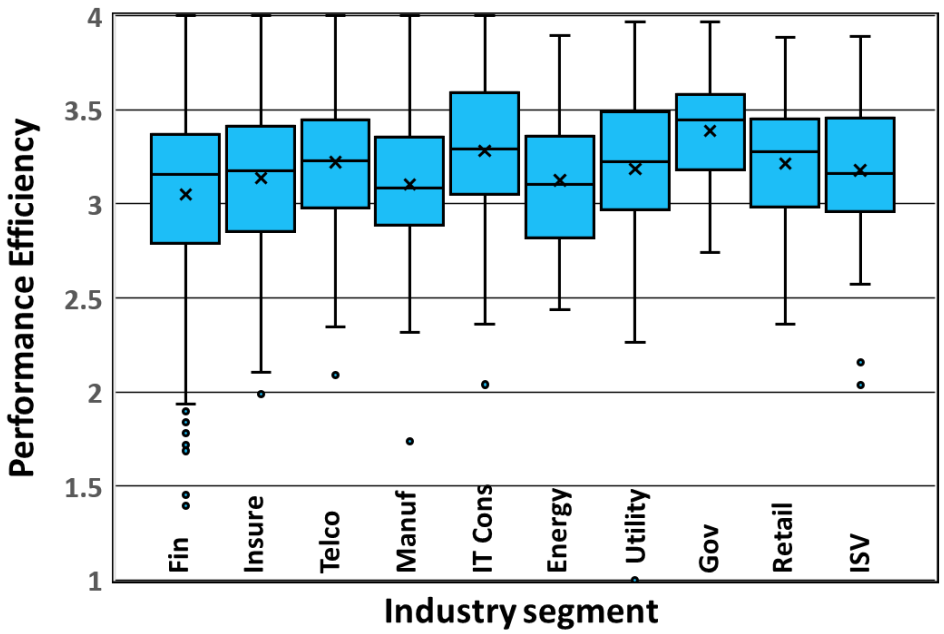


Figure 10. Distribution of Performance Efficiency Scores by Industry Segment

***Changeability*.** Differences among the ten industry segments accounted for only 4% of the variation in Changeability scores. Post hoc tests revealed that the primary differences occurred between government, which had the highest mean score (3.25) and the other industries, which had means clustered between 2.97 and 3.08. As is evident in Figure 11, variance was moderate among Changeability scores.

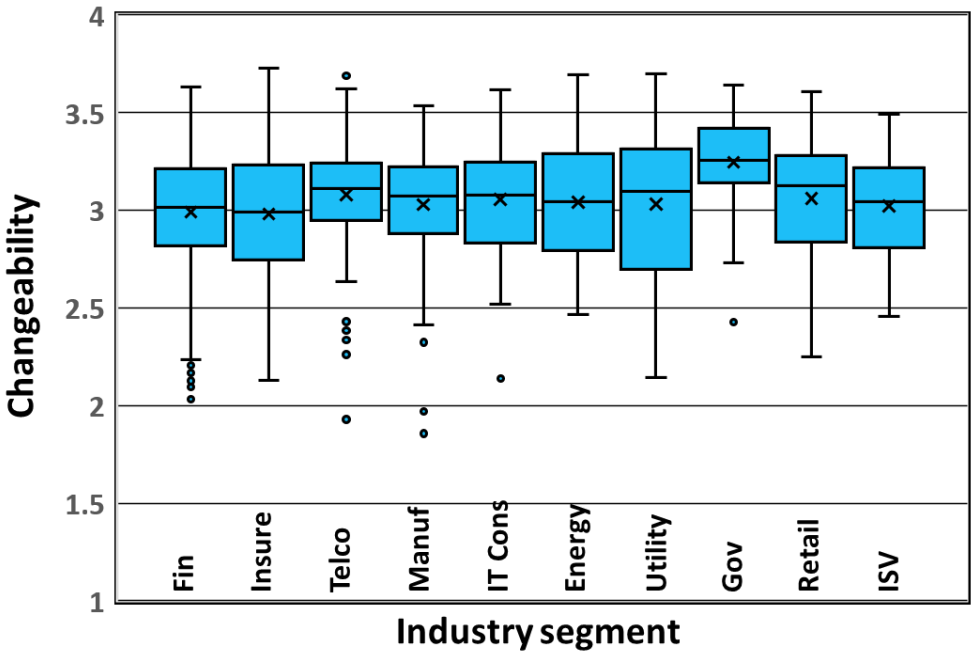


Figure 11. Distribution of Changeability Scores by Industry Segment

***Transferability*.** Differences among the ten industry segments accounted for only 2% of the variation in Transferability scores. Post hoc tests revealed that the primary differences occurred between government, which had the highest mean score (3.25) and financial service and utilities, which had mean scores below 3.0. Figure 12 reveals very similar patterns for all industry segments on Transferability.

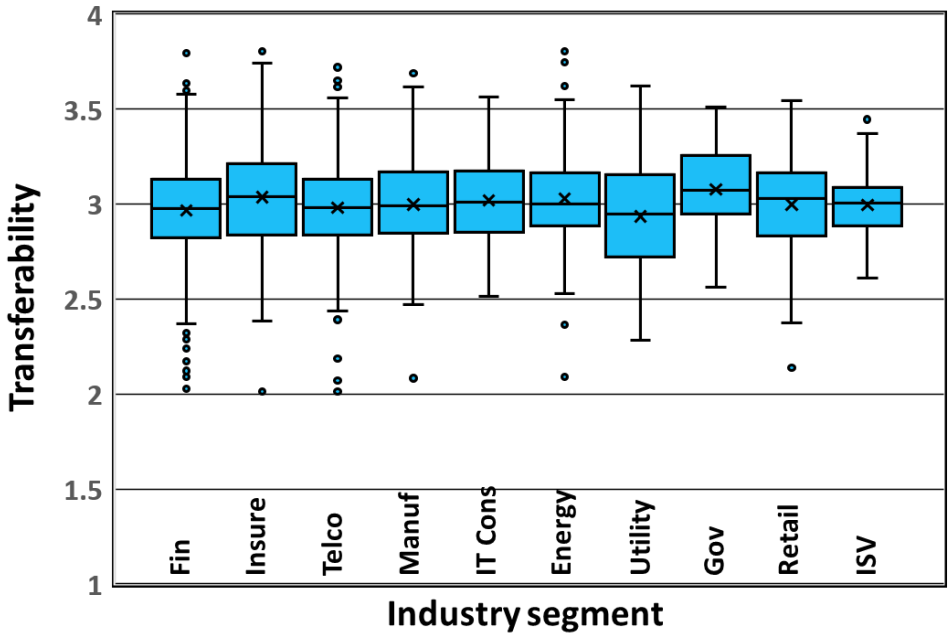


Figure 12. Distribution of Transferability Scores by Industry Segment.

Summary. With the exception of government applications, knowing the industry in which an application was developed provided little information about its Health Factor scores. Financial services tended to have lower scores across all of the Health Factors, but this was probably because of its greater proportion of COBOL-based core transaction systems, which was evident in Tables 3, 4, and 5 in Section 2. However, financial services showed wider variation than other industries in its scores for the operational risk factors of Robustness, Security, and Performance Efficiency. In each of these cases, it contained both the highest and lowest scores.

The primary factors affecting structural quality are beyond the industry segment since it explains little of the variation in Health Factor scores. The next sections will explore other factors that may be more influential in affecting structural quality.

4.2 Technology

Differences among mean scores on each Health Factor for the seven technologies with at least 40 applications in the CRASH sample were compared in an ANOVA. Among the analyses presented in this section, the analysis of technologies is the least exact since different numbers of rules were evaluated for different technologies. Even so, the scoring algorithm used in computing the Health Factor scores provided some correction for differences in the number of rules.

Even with the caveat about rule differences among technologies, we can still draw some interesting observations. Large and statistically significant differences ($p < .001$) in means were observed between technologies fora all five Health Factors. Table 11 summarizes the mean scores for each of the technologies on each Health Factor. The following paragraphs will describe in greater depth the distribution of scores, which technologies were responsible for the significant differences, and why the percentage of variation accounted for by technology was smaller in Security and Transferability.

Technology	#	Robust	Security	Perform	Change	Transfer
Java-EE	744	3.32	3.24	3.25	3.18	3.04
COBOL	410	3.02	3.17	2.91	2.85	2.99
.NET	184	3.27	3.20	2.99	2.88	3.02
ABAP	112	3.11	3.39	3.03	3.05	3.00
Oracle	67	3.04	3.14	3.20	2.87	2.72
C	45	3.02	3.37	3.45	2.90	2.87
C++	44	3.26	3.40	3.43	3.01	2.95
% variance explained		**20%**	**3%**	**16%**	**26%**	**7%**

Table 11. Means and variance explained in Health Factors by technology

***Robustness*.** Differences among the seven technologies accounted for 20% of the variation in Robustness scores. Post hoc tests revealed that the primary differences occurred between Java-EE, .NET, and C++, which had higher means and COBOL, Oracle Server, and C, which had lower means. The mean for ABAP rested between these two clusters of high and low means.

As we can see in Figure 13, COBOL and Oracle Server both displayed wide variation, with COBOL earning both the highest and lowest Robustness scores.

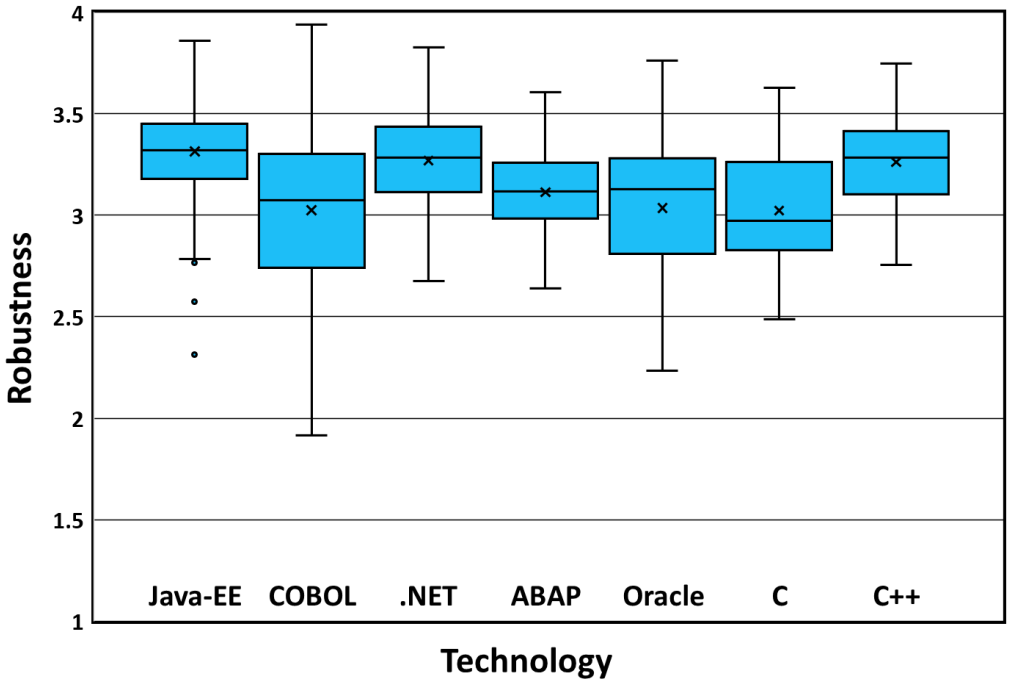


Figure 13. Distribution of Robustness Scores by Technology

***Security*.** Differences among the seven technologies accounted for only 3% of the variation among Security scores. Post hoc tests revealed that the primary differences occurred between C++, ABAP, and C, which had higher means and COBOL and Oracle Server, which had lower means. The means for Java-EE and .NET rested between these two clusters of high and low means. As we can see in Figure 14, COBOL and Oracle Server both displayed wide variation, with COBOL containing both the highest and lowest Security scores. In fact, even though COBOL had the largest percentage of scores below 3.0, it also had the largest percentage of scores above 3.5. Although COBOL applications are often among the most secure, some receive extremely low Security scores, which reduces the overall mean. These large variations among scores within several technologies are the reason that only 3% of the variation in Security scores was explained by differences in technologies.

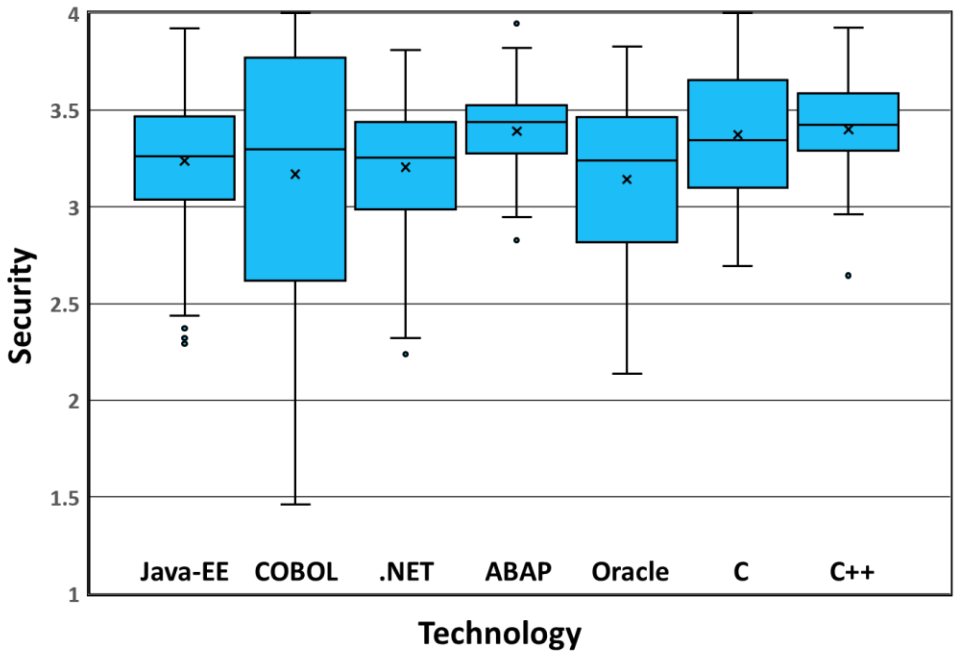


Figure 14. Distribution of Security Scores by Technology

***Performance Efficiency*.** Differences among the seven technologies accounted for 16% of the variation in Performance Efficiency scores. Post hoc tests revealed that the primary differences occurred between C and C++, which had higher means and COBOL, which had lower means. Means for other technologies were positioned between these two groups. As we can see in Figure 15, COBOL displayed wide variation and contained both the highest and lowest Performance Efficiency scores. Since C and C++ are designed to allow developers to get close to the machine, it is not surprising that their scores are higher for Performance Efficiency. The high COBOL scores at the upper end of its distribution likely result from many years of tuning a transaction processing system for the speed of handling large volumes of transactions.

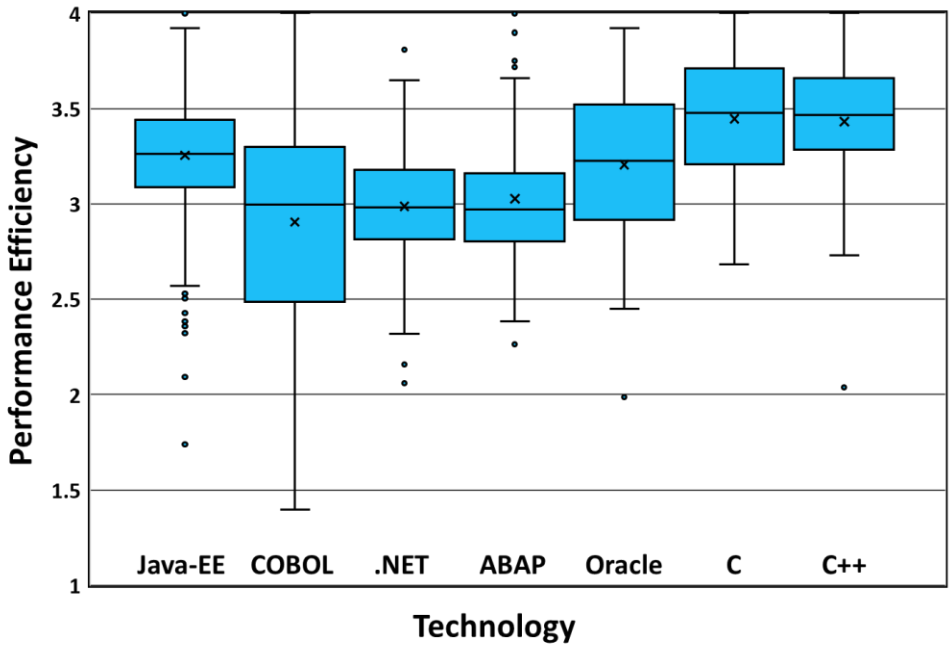


Figure 15. Distribution of Performance Efficiency Scores by Technology

***Changeability*.** Differences among the seven technologies accounted for 26% of the variation in Changeability scores. Post hoc tests revealed that the primary difference occurred between Java-EE, which had a higher mean and COBOL, .NET, Oracle Server, and C which had lower means. The means for ABAP and C++ rested between these two clusters of high and low means. As can be seen in Figure 16, COBOL and Oracle Server both displayed wide variation in Changeability scores. However, for most of the technologies, interquartile ranges for Changeability were smaller than they were for Robustness, Security, and Performance Efficiency, indicating less variability among applications in the centers of the distributions.

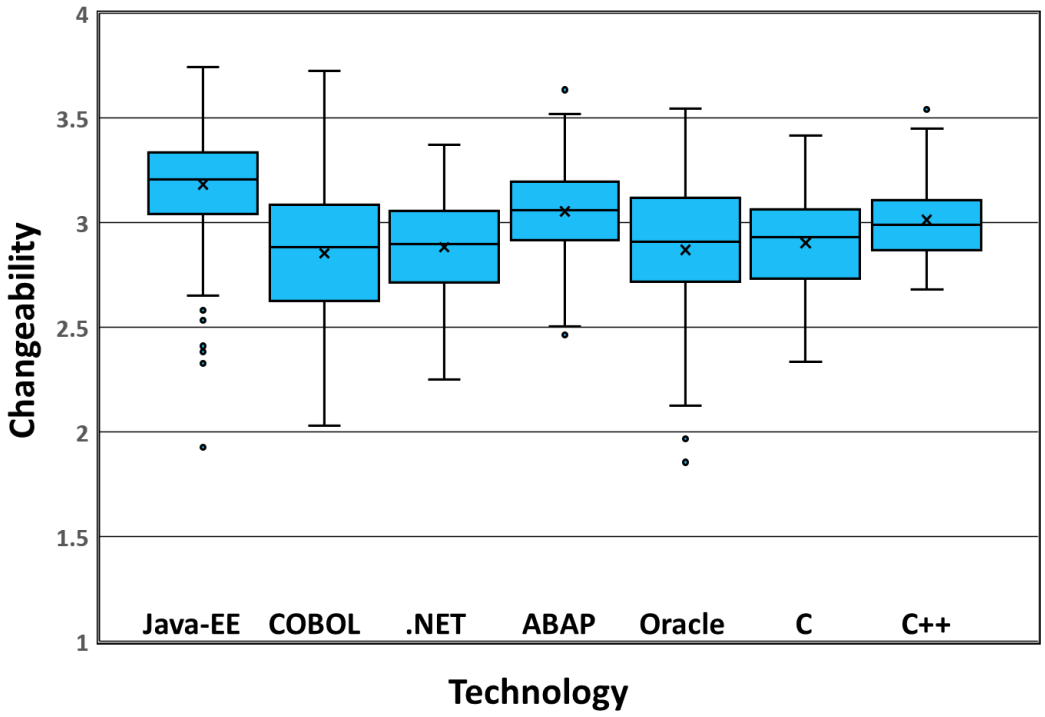


Figure 16. Distribution of Changeability Scores by Technology

***Transferability*.** Differences among the seven technologies accounted for only 7% of the variation in Transferability scores. Post hoc tests revealed that the primary difference occurred between Java-EE, .NET, ABAP, and COBOL, which had a higher means and Oracle Server, which had the lowest mean. The means for C and C++ rested between these two clusters of high and low means. As we can see in Figure 17, COBOL and Oracle Server both displayed slightly wider variation. However, for most of the technologies, interquartile ranges for Transferability were smaller than they were for the other Health Factors, with the exception of Changeability, indicating less variability among applications in the centers of the distributions.

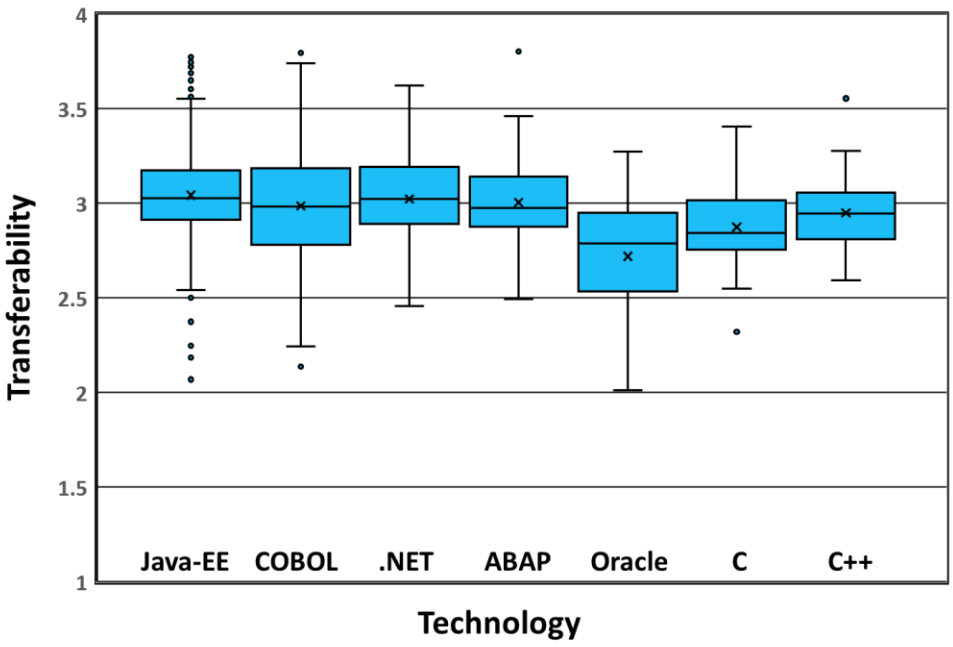


Figure 17. Distribution of Transferability Scores by Technology

Summary. For Robustness, Performance Efficiency, and Changeability, strong differences were observed between technologies. However, technology accounted for only a small percentage of the variation among scores for Security and Transferability. Applications built in COBOL and Oracle Server exhibited the lowest scores on most Health Factors. The lowest distributions of scores for all technologies were observed for the maintainability/cost factors of Changeability and Transferability.

Since technology accounted for a large portion of the variation in several Health Factors, most analyses in future sections will be conducted within a single technology so that Health Factor scores can be computed identically. The effect of technology is stronger than the effect of industry segment, and differences in technologies used across industry segments may explain some of the differences detected in Health Factor scores. When we compare the results for Java-EE, the only technology with enough applications in each industry to support analysis, in Table 12 to the similar results in Table 10, we see that the percentage of variation in Health Factor scores rose slightly.

Java-EE	Sample	Robust	Security	Perform	Change	Transfer
Industry	**677**	5%	5%	7%	4%	6%

Table 12. Percentage of variance explained in Health Factors across industry segments in Java-EE

4.3 Type of Application System

Differences among mean scores on each Health Factor for the six types of application systems with at least 40 applications in the CRASH sample were compared in an ANOVA. These types of systems included core transaction processing, enterprise resource planning customer-facing websites, enterprise portals, customer resource management, and analytics. Table 13 summarizes the mean scores for each type of system on each Health Factor. Statistically significant differences for all Health Factors were observed, and were significant above $p < .002$ for all factors except Transferability. However, as reported in Table 13, differences among types of applications never accounted for more than 6% of the variation in scores, and for Security and Transferability it barely reached 1%. The following paragraphs describe in greater depth the distribution of scores and which industry segments were responsible for the significant differences.

Type	#	Robust	Security	Perform	Change	Transfer
Core trans.	551	3.14	3.19	3.03	2.97	2.99
ERP	352	3.21	3.29	3.22	3.07	2.98
Website	162	3.23	3.26	3.23	3.03	3.00
Portal	161	3.29	3.27	3.21	3.12	3.03
CRM	91	3.21	3.16	3.34	3.10	3.03
Analytics	71	3.19	3.15	3.13	3.03	2.93
% variance explained		3%	1%	6%	4%	1%

Table 13. Means and variance explained in Health Factors by type of system

Robustness. Differences among the six types of application systems accounted for only 3% of the variation in Robustness scores. Post hoc tests revealed that the primary differences occurred between enterprise portals, which had the highest mean score (3.29) and core transactions systems, which had the lowest mean (3.14). As we can see in Figure 18, core

transaction systems displayed the widest variation, containing both the highest and lowest Robustness scores.

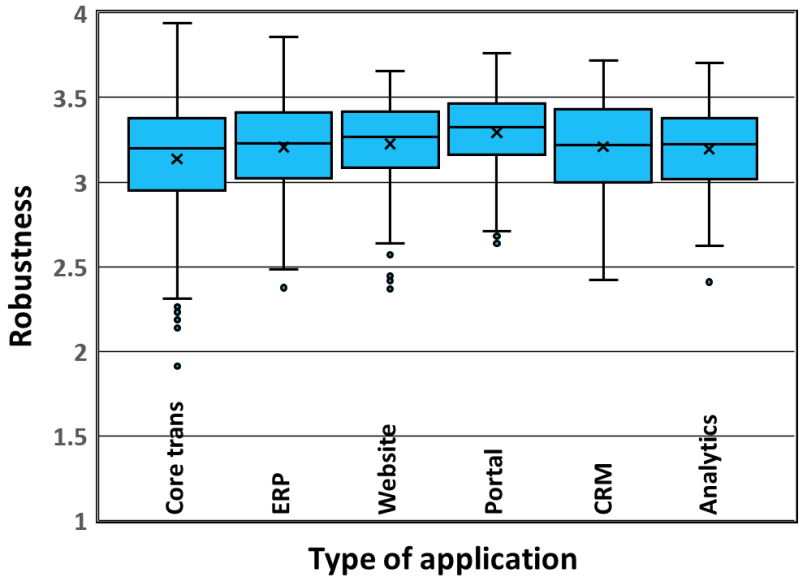


Figure 18. Distribution of Robustness scores by system type

***Security*.** Differences among the six types of application systems accounted for only 1% of the variation among Security scores. Post hoc tests revealed that the primary differences occurred between ERP, customer-facing websites, and enterprise portals, which had means between 3.26 and 3.29, and core transaction systems, CRM and analytics, which had mean scores between 3.16 and 3.19. As we can see in Figure 19, core transaction systems displayed the wide variation, containing both the highest and lowest Security scores. In fact, even though core transaction systems had the largest percentage of scores below 3.0, they also had the largest percentage of scores above 3.5.

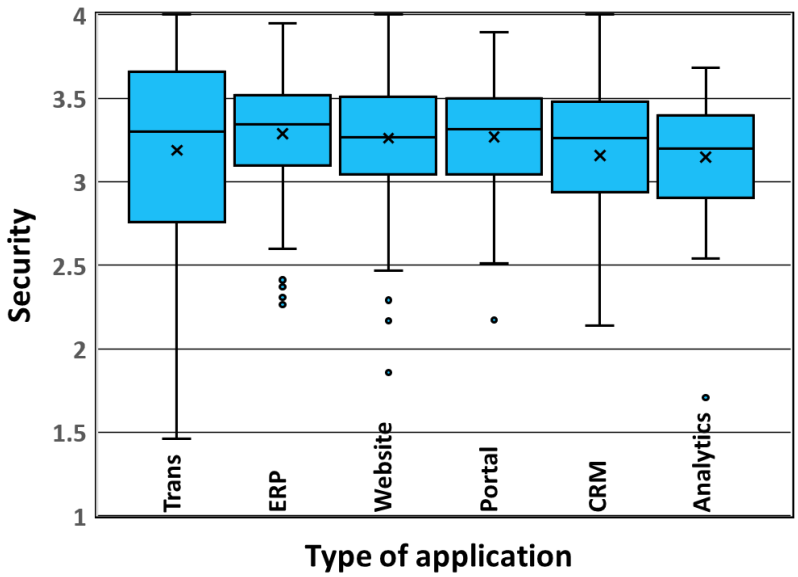


Figure 19. Distribution of Security scores by system type

***Performance Efficiency*.** Differences among the six types of application systems accounted for only 6% of the variation in Performance Efficiency scores. Post hoc tests revealed that the primary differences occurred between CRM, which had the highest (3.34) mean, and core transaction systems, which had with the lowest mean (3.03). As we can see in Figure 20, core transaction systems displayed the widest variation and contained both the highest and lowest Performance Efficiency scores.

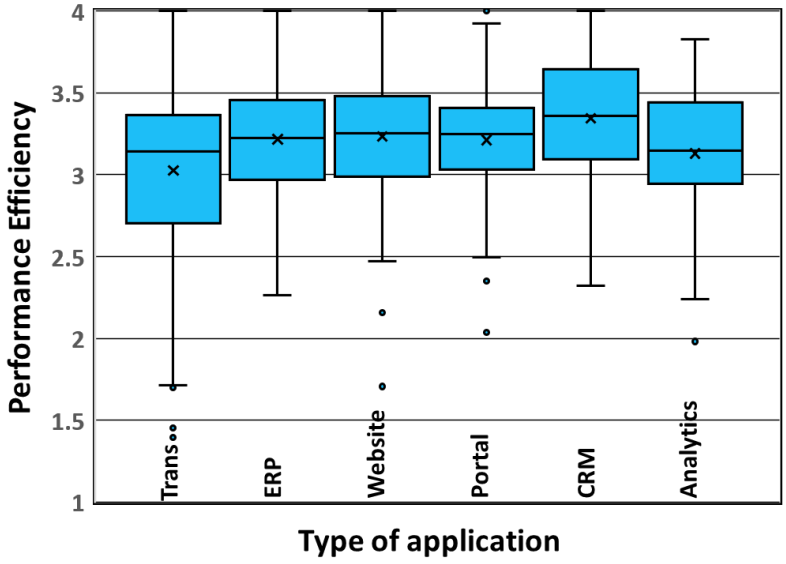


Figure 20. Distribution of Performance Efficiency scores by system type

***Changeability*.** Differences among the six types of application systems accounted for only 4% of the variation in Changeability scores. Post hoc tests revealed that the primary differences occurred between CRM (3.10) and enterprise portals (3.12), which had the highest mean scores, and core transaction systems, which had the lowest mean (2.97). As we can see in Figure 21, the variation in scores was not as wide as it had been for other Health Factors.

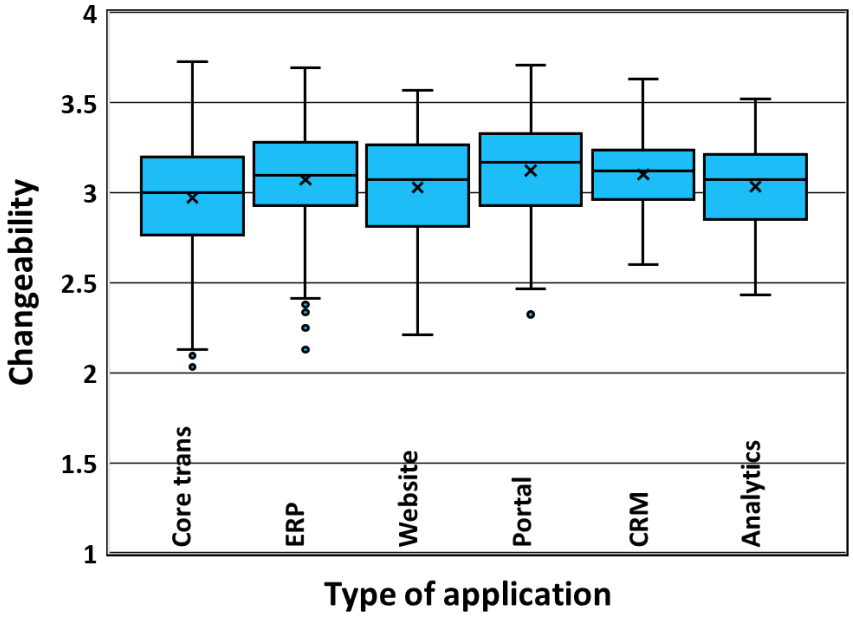


Figure 21. Distribution of Changeability scores by system type

***Transferability*.** Differences among the six types of application systems accounted for only 1% of the variation in Transferability scores. As is evident in Figure 22, post hoc tests revealed that the primary difference was the low score for analytics (2.93) compared to mean scores for other system types. As can be seen in Figure 20, the variation in scores was not as wide as it had been for other Health Factors.

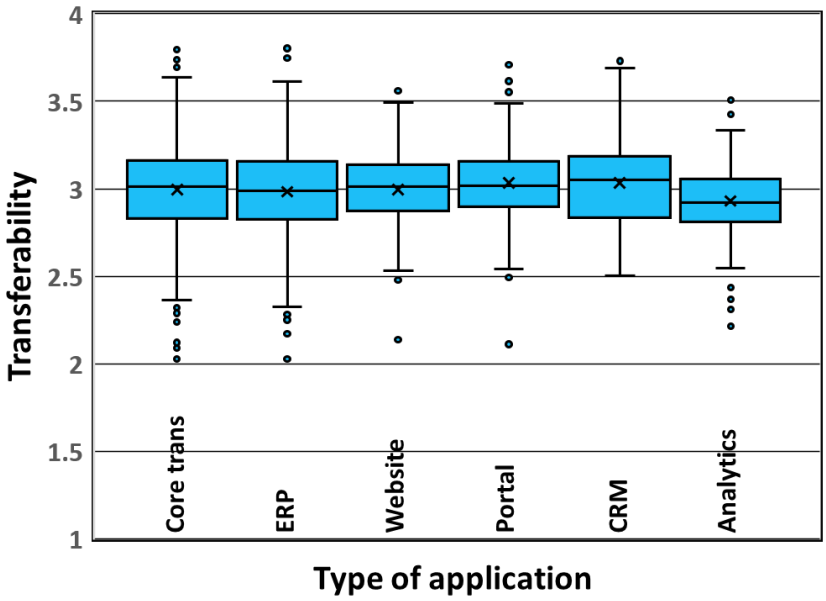


Figure 22. Distribution of Transferability scores by system type

System by technology comparisons. The technology used in developing the application system may be more important than the type of system itself in determining its Health Factor scores. In fact, there is a relation between the type of system and the technology used to develop it. Most core transaction systems, especially in financial services and insurance, have been developed in COBOL. ABAP is primarily used in ERP systems. Customer-facing websites and enterprise portals are most often built in Java-EE.

To investigate these interactions, ANOVAs were conducted on a single type of system developed in several technologies. As reported in Table 14, with a few exceptions (n.s. = not statistically significant), core transaction systems and customer-facing websites built in Java-EE were found to have significantly higher Health Factor mean scores that those developed in COBOL. Similarly, with the exception of Security, enterprise resource planning systems built in Java-EE posted higher mean scores than those developed in ABAP. The percentages of variance explained in these comparisons were often 10% or greater and explain why the variance explained by system type is low. In essence, differently developed technologies create substantial variance in structural quality results within a system type. Therefore, when benchmarking, it is critical to compare against similar technologies to gain useful insights.

Technology	#	Robust	Security	Perform	Change	Transfer
Core transaction system						
Java-EE	**152**	3.31	3.22	3.26	3.19	3.02
COBOL	**300**	3.03	3.17	2.85	2.86	3.00
% variance explained		**14%**	**n.s.**	**14%**	**24%**	**n.s.**
Customer-facing website						
Java-EE	**80**	3.34	3.27	3.32	3.21	3.06
COBOL	**29**	2.95	3.29	3.13	2.71	2.86
% variance explained		**39%**	**n.s.**	**7%**	**52%**	**16%**
Enterprise resource planning						
Java-EE	**125**	3.35	3.27	3.28	3.22	3.08
ABAP	**78**	3.15	3.45	3.03	3.11	3.02
% variance explained		**20%**	**10%**	**16%**	**7%**	**2%**

Table 14. Means and variance explained in Health Factors by technologies within system types (n.s. means not statistically significant)

4.4 Source

Differences among mean scores on each Health Factor for applications developed in-house versus outsourced were compared in an ANOVA. The ANOVAs were conducted separately within each technology, with at least 40 applications in each sourcing category. Table 15 summarizes the mean scores on each Health Factor within each technology. Eleven of the 15 comparisons across Java-EE, COBOL, and .NET were statistically insignificant. The four analyses that achieved statistical significance, three of which were in Java-EE, accounted for little of the variation in scores. Of those four, two involved differences in Transferability scores. Even in COBOL and .NET, where differences in means appear meaningful, the size of the variation among scores kept these differences from being significant. Thus, the choice between in-house and outsourced development made little difference in the Health Factor scores in the CRASH sample.

Source	#	Robust	Security	Perform	Change	Transfer
Java-EE						
In-house	**252**	3.30	3.24	3.22	3.18	2.99
Outsourced	**328**	3.35	3.27	3.26	3.22	3.08
% variance explained		**1%**	**n.s**	**n.s.**	**1%**	**4%**
COBOL						
In-house	**85**	3.10	3.35	3.21	2.90	2.93
Outsourced	**183**	3.17	3.47	3.14	2.82	3.02
% variance explained		**n.s.**	**n.s.**	**n.s.**	**n.s.**	**2%**
.NET						
In-house	**40**	3.28	3.15	3.02	2.88	3.00
Outsourced	**84**	3.29	3.24	2.97	2.88	3.05
% variance explained		**n.s.**	**n.s.**	**n.s.**	**n.s.**	**n.s.**

Table 15. Means and variance explained in Health Factors by source

4.5 Shore

Differences among mean scores for each Health Factor for applications developed offshore were compared in an ANOVA. The ANOVAs were conducted separately within each technology, with at least 40 applications in each shoring category. Table 16 summarizes the mean scores for each Health Factor within each technology. All comparisons for Java-EE and .NET were statistically insignificant. However, we observed statistically significant differences for COBOL applications, with particularly large differences in Transferability. In fact, as we can see in Figure 23, three-quarters of the onshore applications scored in the bottom quartile of the offshore applications for Transferability

Because of their age, most COBOL applications were likely developed onshore, and some were then sent offshore for maintenance and enhancement. As presented in Section 4.2, COBOL applications usually post among the lowest Health Factor scores. A possible factor affecting the decision to offshore COBOL applications, especially if they are core transaction systems, is to reduce operational risk when their transferability to offshore groups is hampered by hard-to-understand code. Whether work was done offshore or onshore made little difference for applications developed in modern technologies. However, for COBOL applications, the decision appears to involve structural quality implications, especially in terms of transferability.

Shore	#	Robust	Security	Perform	Change	Transfer
Java-EE						
Offshore	**109**	3.31	3.25	3.23	3.19	3.06
Onshore	**431**	3.33	3.26	3.25	3.21	3.04
% variance explained		**n.s.**	**n.s.**	**n.s.**	**n.s.**	**n.s.**
COBOL						
Offshore	**69**	3.28	3.58	3.23	2.93	3.23
Onshore	**176**	3.14	3.48	3.20	2.77	2.92
% variance explained		**4%**	**n.s.**	**n.s.**	**6%**	**18%**
.NET						
Offshore	**57**	3.27	3.19	3.00	2.89	3.04
Onshore	**58**	3.30	3.19	3.02	2.88	3.03
% variance explained		**n.s.**	**n.s.**	**n.s.**	**n.s.**	**n.s.**

Table 16. Means and variance explained in Health Factors by shore

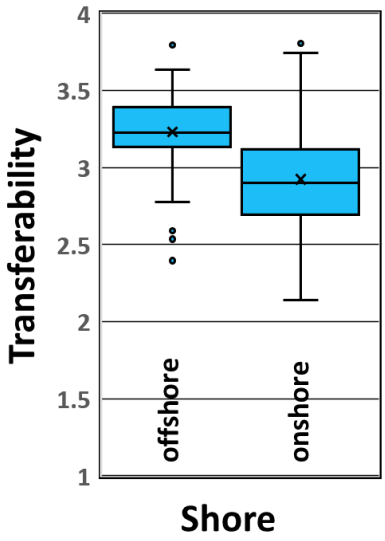


Figure 23. Distribution of Transferability scores by shore for COBOL

4.6 Geographical Region

Differences among mean scores for each Health Factor for applications developed in Continental Europe, the United Kingdom, the United States, and India were compared in an ANOVA. The ANOVAs were conducted only for Java-EE applications since that was the only technology for which there were at least 30 applications in each country. Table 17 summarizes the mean scores on each Health Factor in each country. The comparisons for Robustness, Security, and Transferability were statistically significant. However, only the comparison for Security accounted for more than 5% of the variation in scores.

Region	#	Robust	Security	Perform	Change	Transfer
Europe	595	3.23	3.34	3.24	3.05	2.96
UK	46	3.12	3.12	3.14	2.98	2.86
US	111	3.23	3.13	3.20	3.06	2.96
India	179	3.19	3.21	3.22	3.03	3.00
% variance explained		1%	5%	n.s.	n.s.	1%

Table 17. Means and variance explained for Health Factors in Java-EE by country

The distributions of Security scores in Java-EE applications are presented by region in Figure 23. Regional differences accounted for 5% of the variation in Security scores. Post hoc tests revealed that Continental Europe had the highest Security scores, while the UK and US had the lowest scores. The variation of Security scores from applications in the United States is especially wide. Figure 24 does not include distributions for Robustness and Transferability scores since they accounted for only 1% of the variation in scores.

The higher Health Factor scores achieved by applications developed and maintained in Continental Europe were investigated for explanations, but unfortunately, within the data available, no explanatory patterns were evident. One possibility is that Continental Europe was CAST's first market and companies in those countries have been working with Health Factor feedback longer and have had more time to improve applications.

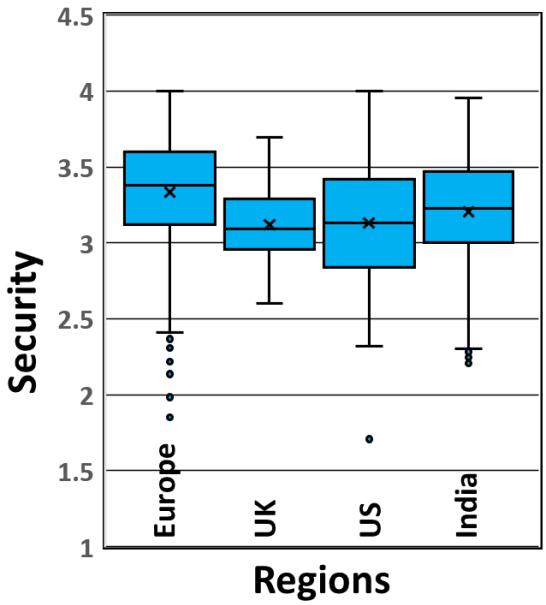


Figure 24. Distributions of Robustness, Security, and Transferability scores by region

4.7 Organizational Maturity

Differences among mean scores on each Health Factor for applications developed under different levels of organizational maturity were compared in an ANOVA. Organizational maturity was measured as reported appraisal results using the Capability Maturity Model Integration (CMMI) framework. Java-EE was the only technology for which there were enough appraisal results to conduct an analysis. Even so, there were 27 or fewer applications at each maturity level, and there were so few from high maturity organizations that this analysis only covers CMMI Levels 1-3. The analysis outcomes should be treated as suggestive of trends, since the samples in each category fell below the desired level of 40 applications.

Table 18 summarizes the mean scores for each Health Factor at each CMMI Level for applications developed in Java-EE. All comparisons were statistically significant and accounted for between 12% and 28% of the variation in scores. The percentages of variation accounted for could be somewhat enhanced by the smaller number of applications in this analysis. Nevertheless, the Health Factor score distributions in Figure 24 present a consistent picture of how organizational maturity affects structural quality.

CMMI Level	#	Robust	Security	Perform	Change	Transfer
Java-EE						
CMMI Level 1	23	3.22	3.06	3.11	3.14	2.93
CMMI Level 2	27	3.44	3.40	3.31	3.37	3.09
CMMI Level 3	22	3.40	3.32	3.28	3.26	3.08
% variance explained		28%	25%	15%	24%	12%

Table 18. Means and variance explained in Health Factors by CMMI Level

Since the patterns are virtually identical across all Health Factors, all 5 graphs have been combined into Figure 25. Post hoc analyses indicated that all significant differences resulted from the lower scores for CMMI Level 1 compared to Levels 2 and 3. In fact, almost ¾ of the Robustness scores for applications developed in CMMI Level 1 organizations fall into or below the lowest quartile of Robustness scores for applications developed in CMMI Level 2 and 3 organizations. A similar pattern was observed for Security scores where ½ half of the Security scores for applications developed in Level 1 organizations fall into or below the lowest quartile for applications developed in Level 2 or 3 organizations.

Performance Efficiency and Transferability scores showed the same pattern, but not as strongly as the Robustness and Security distributions. The distribution of scores for Changeability was different from that of the four other Health Factors. Post hoc tests revealed that the Changeability means for all three levels were significantly different from each other. While the mean for Level 3 was higher than that of Level 1, the mean for Level 3 was lower than that of Level 2, even though the medians were close. It is not clear what caused the spread in the lower tail of the Changeability distribution of Level 3 applications.

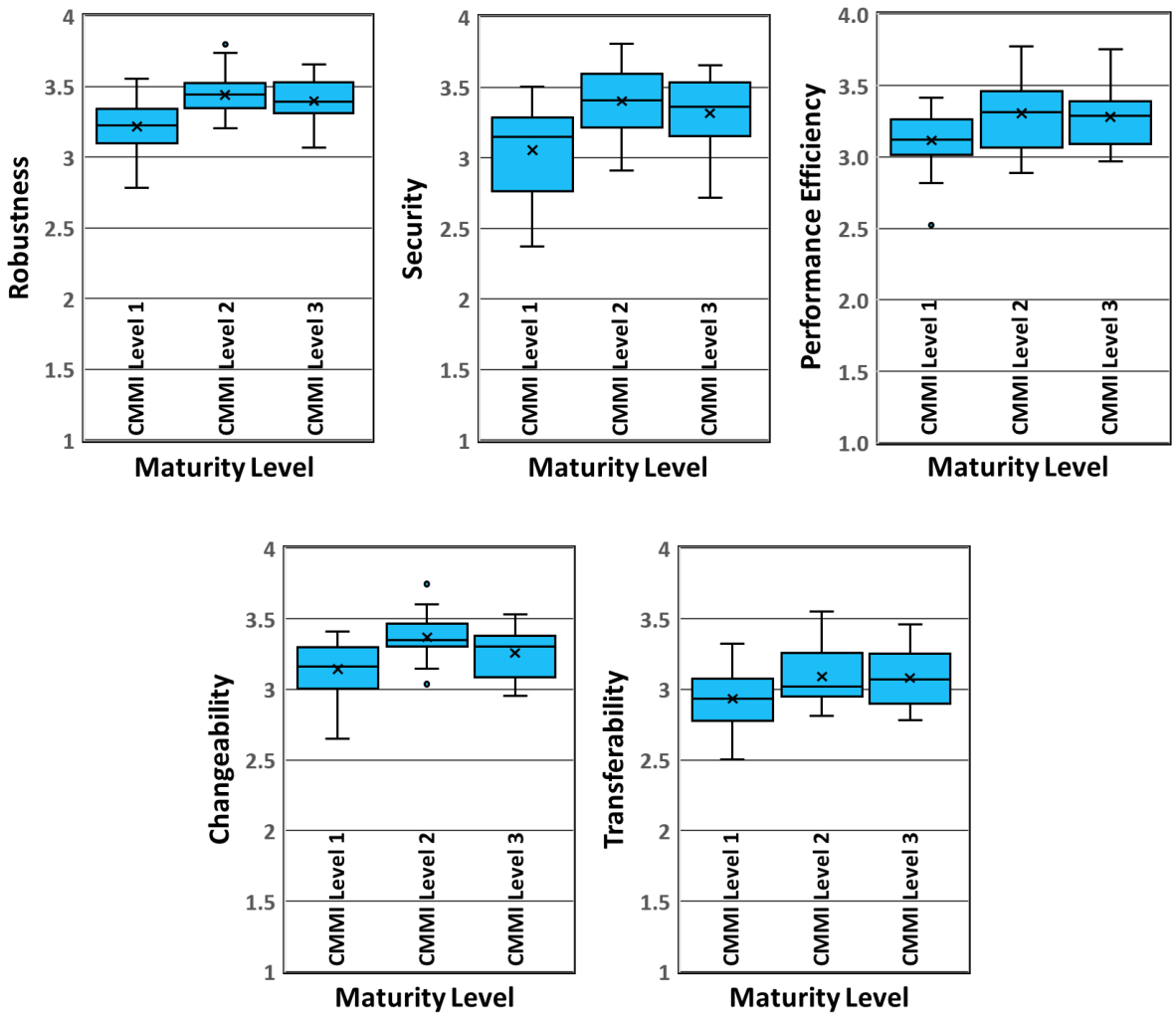


Figure 25. Distribution of Health Factor scores by maturity level

These results are not surprising since CMMI Level 1 environments are characterized by uncontrolled commitments and baselines. Developers are frequently overworked on unrealistic schedules. Consequently, they make lots of mistakes without having adequate time to detect and correct them. The lower scores for applications developed in CMMI Level 1 organizations reflect these conditions.

When organizations achieve a CMMI Level 2 capability, those in charge of projects or applications gain control of the commitment and baseline processes. Consequently, projects can plan and protect the time required to perform disciplined development. As a result, the structural quality of their work improves. Since CMMI Level 3 focuses on collecting best practices by implementing Level 2 capabilities and integrating them into a standard organizational process, the improvement made in achieving Level 3 status is one of organizational efficiency in an economy of scale, rather than one of developer performance.

Consequently, the primary improvement in structural quality occurs as organizations move from CMMI Level 1 to Level 2.

4.8 Development Method

Differences among mean scores on each Health Factor for applications developed using different development methods were compared in an ANOVA. Java-EE was the only technology for which there were enough reports of the method used to conduct analyses. Although there were only 20 applications reporting that no method was used, this category was retained to serve as a baseline.

Table 19 summarizes the mean scores for each Health Factor within each method. Four of the five comparisons were statistically significant. Differences among methods had their strongest effects on Robustness and Changeability, where they accounted for 10% and 14% of the variation respectively. Although the pattern of mean Security scores was similar to the other Health Factor means, as is evident in Figure 26, the larger variation in scores masked differences among Security means.

Method	#	Robust	Security	Perform	Change	Transfer
Java-EE						
Agile	**77**	3.26	3.14	3.22	3.12	3.00
Waterfall	**60**	3.24	3.21	3.16	3.13	2.95
Hybrid mix	**51**	3.36	3.27	3.37	3.27	3.08
None	**20**	3.13	3.15	3.14	3.00	2.92
% variance explained		**10%**	**n.s.**	**6%**	**14%**	**6%**

Table 19. Means and variance explained in Health Factors by development method

Post hoc tests revealed that the significant results in these data were a result of the fact that hybrid methods universally earned the highest scores and no method generally earned lower scores. The Health Factor differences between Agile and Waterfall methods were small, with both consistently earning lower scores than hybrid methods. The consistently better structural quality results for hybrid methods likely result from their combination of up front analysis and design of application architecture, as well as their rapid feedback on defects during short, iterative sprints. This allows developers to address system and

architectural level quality issues early, and then detect more granular issues as code is developed.

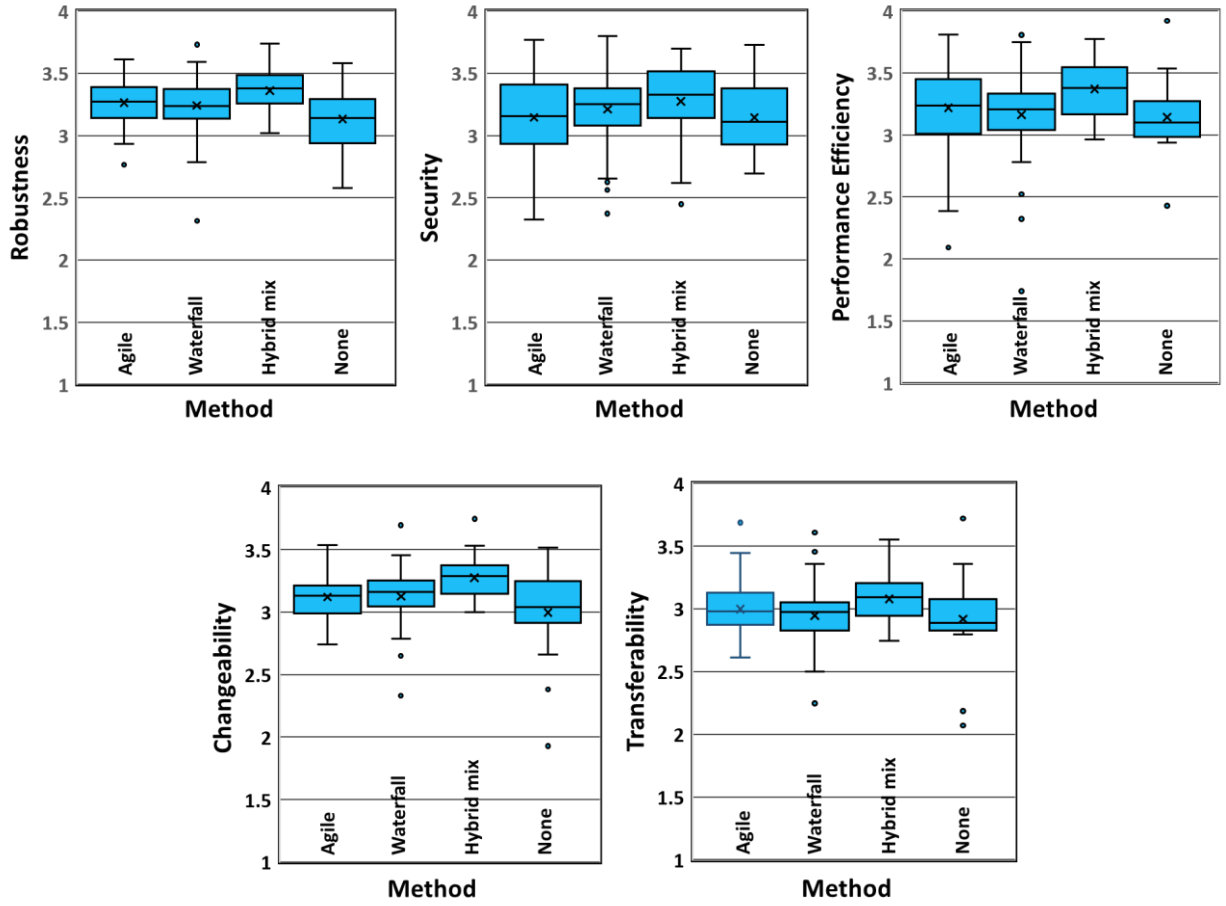


Figure 26. Distributions of Health Factor scores by development method

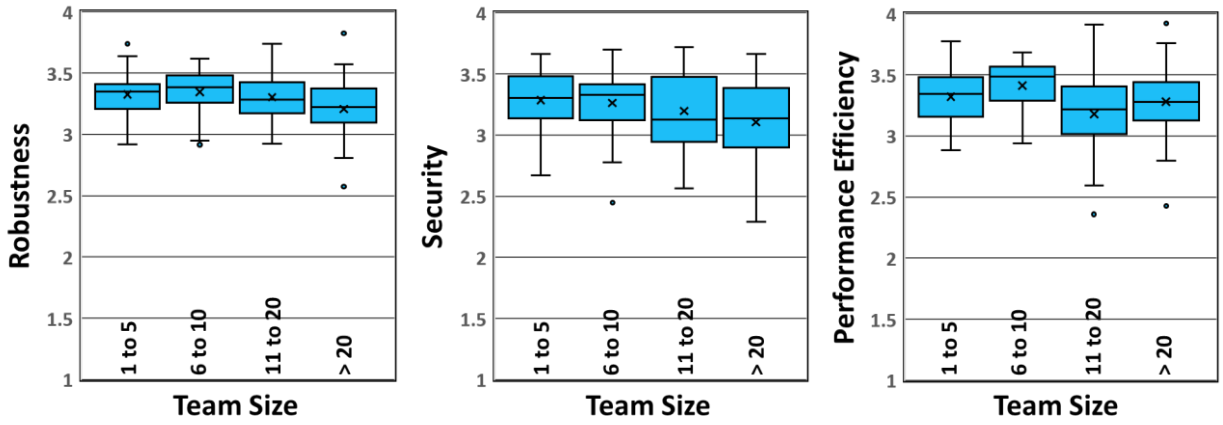
4.9 Team Size

Differences among mean scores for each Health Factor for applications developed by teams of different sizes were compared in an ANOVA. Java-EE was the only technology for which there were enough reports of team size to conduct analyses. Although there were only 29 applications reporting a team size of 6 to 10 developers, this category was retained for continuity.

Team size	#	Robust	Security	Perform	Change	Transfer
Java-EE						
1 to 5	49	3.33	3.29	3.32	3.23	3.10
6 to 10	29	3.35	3.26	3.41	3.21	3.09
11 to 20	44	3.30	3.20	3.18	3.25	3.01
> 20	44	3.21	3.11	3.28	3.09	3.02
% variance explained		7%	5%	8%	8%	n.s.

Table 20. Means and variance explained in Health Factors by team size

Table 20 summarizes the mean scores for each Health Factor for each team size. Four of the five comparisons were statistically significant. Differences in team size had moderate effects accounting for 5% to 8% of the variation respectively. Post hoc tests revealed that these differences primarily involved lower means for teams of more than 20 developers, except for Performance Efficiency, where teams of 11 to 20 developers posted the lowest mean scores. As we can see in Figure 27, the variation in Health Factor scores for all categories of team size tended to be larger for Security and Performance Efficiency.



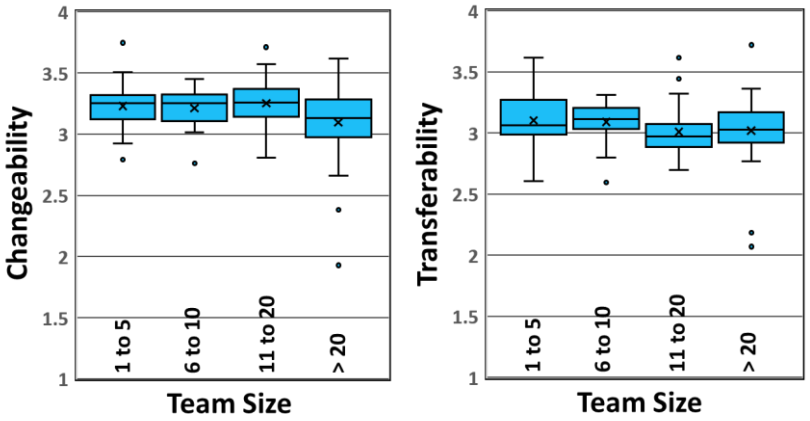


Figure 27. Distributions of Health Factor scores by team size

4.10 Number of Users

Differences among mean scores for each Health Factor for applications serving different numbers of users were compared in an ANOVA. Java-EE was the only technology for which there were enough reports of user numbers to conduct analyses. Since applications in the CRASH tended to be business-critical applications, most served over 5000 users, which indicates that they were customer-facing in most cases. The two categories between 501 and 5000 users had 30 or fewer applications each, so the wide variation in sample sizes warrants some caution in interpreting results.

Table 21 summarizes the mean scores for each Health Factor for different numbers of users. Four of the five comparisons were statistically significant. Differences in number of users were small, accounting for 3% to 5% of the variation respectively. Post hoc tests revealed that these differences primarily involved higher means for applications with over 5000 users, although in the case of Robustness and Performance Efficiency means for applications serving 5001 to 1000 users were close to those serving over 5000. As we can see in Figure 28, the variation in Health Factor scores for all categories of users served tended to be larger for Security and Performance Efficiency.

# of Users	#	Robust	Security	Perform	Change	Transfer
Java-EE						
≤ 500	73	3.24	3.16	3.19	3.12	3.02
501 to 1000	25	3.27	3.17	3.29	3.14	3.02
1001 to 5000	30	3.22	3.17	3.16	3.10	2.97
> 5000	134	3.32	3.27	3.31	3.20	3.06
% variance explained		4%	3%	5%	4%	n.s.

Table 21. Means and variance explained in Health Factors by number of users

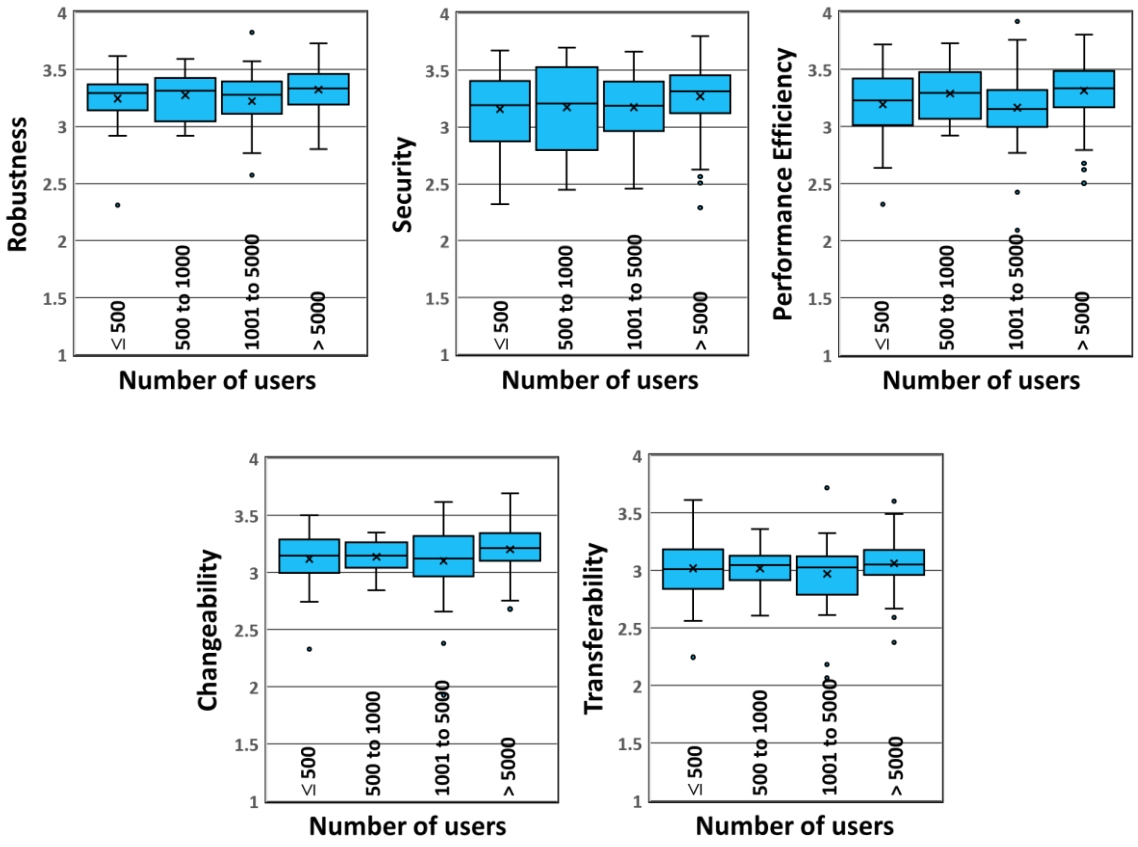


Figure 28. Distributions of Health Factor scores by number of users

5 Conclusions

5.1 Summary of Results

The findings from the 2017 Appmarq sample are believed to be generalizable to the larger population of business-critical applications. The heaviest concentrations of applications came from financial services, insurance, telecommunications, and manufacturing. Applications in a broad range of technologies were analyzed, but they were most often-developed in Java-EE, COBOL, .NET, and ABAP. The most frequent types of applications in the 2017 Appmarq sample were core transaction processing, enterprise resource planning, customer-facing websites, and enterprise portals. Not surprisingly, there were associations between the types of application systems and the technologies in which they were developed.

Although not exactly comparable because of different numbers of rules defined for each Health Factor, scores for those related to operational risk (Robustness, Security, Performance Efficiency) were higher than those related to cost/maintainability (Changeability, Transferability). Scores on Security varied widely, with some of the highest and lowest scores recorded. Analysis at the level of individual violations of quality rules indicated that greater compliance was observed for system-level, critical, and operational risk related rules.

The size of the application in lines of code had negligible effects on Health Factor scores. However, the relationships between operational risk and cost/maintainability Health Factors seem to have increased because of programming styles affected by the smaller module sizes used in modern development technologies.

Several factors were found to have significant influence on the structural quality of business applications. Table 22 presents a summary of the effect sizes for each application, in terms of percentage of variation, explained in Health Factor scores by the demographic category into which the application fell. Differences in technology were found to have a sizeable effect on Robustness, Performance Efficiency, and Changeability scores. Since different numbers of quality rules were measured for different technologies, the remainder of the demographic factors were evaluated separately within technologies that had enough applications in the various categories of each demographic to support statistical analysis. Only Java-EE contained enough applications in each category of each demographic factor to support analysis of all the factors. To support better comparability, Table 22 only reports the percent of variance in results for Java-EE applications.

Variable	Sample	Robust	Security	Perform	Change	Transfer
Technology	1606	20%	3%	16%	26%	7%
Java-EE						
Industry	677	5%	5%	7%	4%	6%
App type	510	1%	1%	1%	1%	1%
Source	580	1%			1%	4%
Shore	540					
Region	378	1%	5%			1%
Maturity	72	28%	25%	15%	24%	12%
Method	208	10%		6%	14%	6%
Team size	186	7%	5%	8%	8%	
# of users	262	4%	3%	5%	4%	

Table 22. Effect size comparisons among demographics affecting Health Factors in Java-EE

When compared across Java-EE applications, the demographic factors with the largest impact on Health Factor scores were organizational maturity and development method. Smaller effects were found for team size, industry segment, and number of users. The type of application— whether it was developed in-house or outsourced, or whether it was developed onshore or offshore—made little difference to Health Factor scores.

5.2 Recommendations

The following recommendations emerge from the 2017 CRASH Report:

1. Benchmarking should be conducted within technology and type of application in order to gain accurate insight into comparable performance. Results from benchmarking purely against industry segment can be misleading because of effects by other factors with greater influence that may vary across organizations.

2. Greater attention must be given to secure coding practices as many applications had scores that were unacceptably low. Security scores displayed wider variation than those of any other Health Factor.

3. To improve Health Factor scores, organizations must improve the maturity of their development processes and practices. Low maturity organizations consistently posted lower Health Factor scores.

4. Adopt hybrid methods for developing business critical applications. Hybrid methods achieved higher Health Factor scores than both agile and waterfall methods by integrating up front architectural analysis with rapid feedback on the quality.

5. Avoid creating teams of over 20 developers. Teams of less than 10 appear to be optimal.

6. When considering outsourcing or off shoring, pay attention to factors that are shown to affect structural quality, such as organizational maturity, development method, or team size, since these factors have greater influence than the source or shore of development.

7. Analyze source code on a regular basis prior to release to detect violations of quality rules that put operations or costs at risk. System-level violations are the most critical since they cost far more to fix and may take several release cycles to fully eliminate.

8. Treat structural quality improvement as an iterative process pursued over numerous releases to achieve the optimal quality thresholds.

While adopting these evidence-based recommendations cannot guarantee high structural quality, they have been shown empirically to be associated with higher quality applications.

About CAST

CAST is the software intelligence category leader. CAST technology can see inside custom applications with MRI-like precision, automatically generating intelligence about their inner workings - composition, architecture, transaction flows, cloud readiness, structural flaws, legal and security risks. It's becoming essential for faster modernization for cloud, raising the speed and efficiency of Software Engineering, better open source risk control, and accurate technical due diligence. CAST operates globally with offices in North America, Europe, India, China.
Visit castsoftware.com.

CAST Research Labs

CAST Research Labs (CRL) furthers the empirical study of software implementation in business technology. We provide practical advice and periodic benchmarks to the global application development community, as well as interacting with the academic community. Through scientific analysis of large software applications, we focus on providing insights that can improve application structural quality at the architectural and code unit levels. We also provide guidance on managing technical debt and improving developer, project, and organizational productivity.

Since 2007 CRL has been collecting metrics and demographic characteristics from custom applications deployed by large, IT-intensive enterprises across North America, Europe and India. This unique dataset is stored in the CAST Appmarq benchmarking repository.

The CAST Application Intelligence Platform

CAST Application Intelligence Platform (AIP) is an enterprise-grade software quality analysis and measurement solution designed to analyze multi-tiered, multi-technology applications for technical vulnerabilities and adherence to architectural and coding standards. The intelligence generated by CAST AIP provides:

- An analysis of technical debt to guide application development teams in improving these complex systems
- Insight into risks associated with upgrading software packages, coupled with automated and detailed technical documentation of these complex, legacy systems
- The real-time information needed to improve application health and development team performance

Contact

United States

321 W. 44th St., Suite 501
New-York, NY 10036
USA
+1 212 871 8330

Europe

3, rue Marcel Allégot
92190 Meudon
France
+33 1 46 90 21 00

castsoftware.com