Core Application Structural Quality

Executive Report

Prepared for Acme Financial Services by:

John Doe, Software Consulting, Inc (SCI)

September 2016

Table of Contents

1. Core Applications Summary

Table 1: Summary of applications analyzed

Application	Description	Production Date	Version Number	Releases per Year	Sourcing Method	Number of End-Users	Cost of Outage
ETS	Equity trading system for all equity trading products	01/01/2000	6.2	1 (4 minor releases)	In-house	10,000	$$$$$
FIS	Fixed income system for trading of fixed income products	07/01/2003	4.3	1 (4 minor releases)	Outsourced	6,000	$$$$$

Table 2: Summary of application characteristics

Quality Characteristics	ETS	FIS
Total Quality Index (TQI)	2.73	2.93
Transferability	2.68	3.08
Changeability	2.99	3.19
Robustness	2.83	3.03
Performance	2.68	3.08
Security	2.53	2.93
Critical Violations	1300	900
– per File	3.05	1.34
– per kLOC	25.4	13.0
– per Application	433	450
Complex Objects	44	52
– w/ violations	38	26

Size Characteristics	ETS	FIS
Technical Size		
– kLOC	51	69
– Files	425	674
Functional Weight		
– BFP	1895	2640
– FP (Est.)	358	459
– Total CC	8560	12232
Technologies Top 5 in kLOC		
– JEE	0	44
– C++	30	5
– Cobol	5	0
– SQL	3	2
–.Net	1	1
– Powerbuilder	1	1

1.1 Overview of change in health of Core Applications

The Total Quality Index (TQI) and Health Factors (Robustness, Performance, Security, Transferability and Changeability) analyzed by the CAST AIP for Core Applications are:

Figure 1: ETS Health Factor Scores

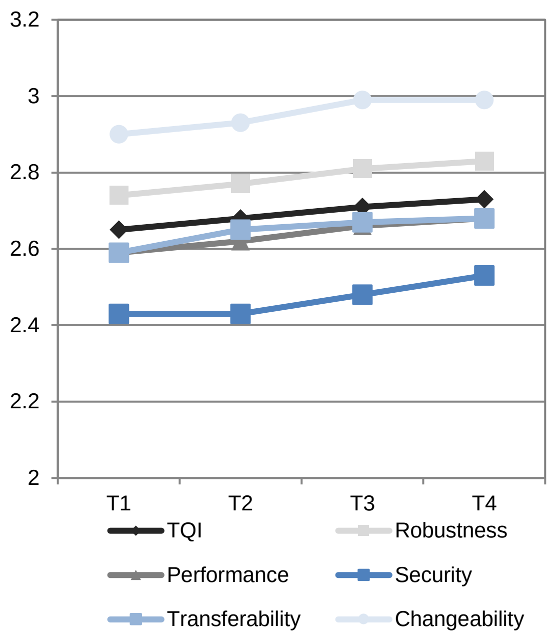
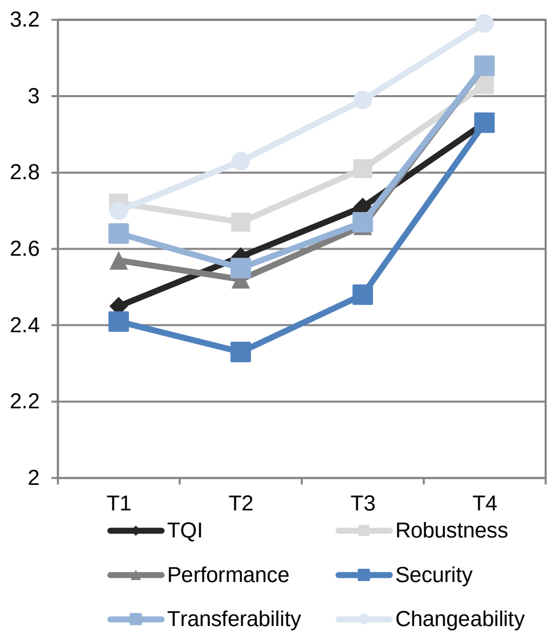
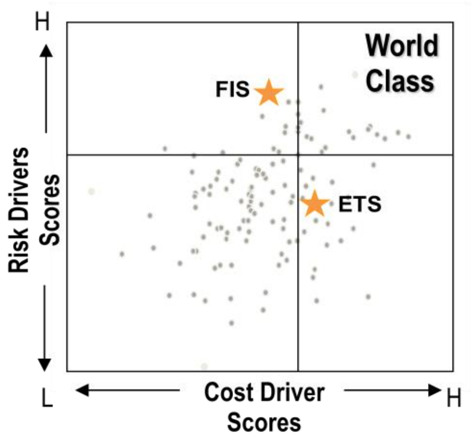


Figure 2: FIS Health Factor Scores



1.2 Benchmarking Analysis

Figure 3: Cost – Risk Matrix of CAST AIP Health Factors



- Benchmarking analysis with similar applications from Financial Services industry showed that FIS is on par or better than the industry average on Risk parameter, while ETS is below
- ETS is being reviewed for sun setting by the Architecture Review Board
- Remediation plan has been prepared for ETS to mitigate any major risks while a decision is being made on the new green field project

2. Technical Debt

2.1 Technical Debt Calculation Assumptions

Technical Debt is calculated in this report as the cost of fixing the structural quality problems in an application that, if left unfixed, put the business at serious risk. AIP categorizes violations into low, medium and high severity. The Technical Debt calculation assumes that only 50% of high-severity violations, 25% of medium-severity violations, and 10% of low-severity violations require fixing to prevent business disruption. With this in mind, the formula for technical debt becomes:

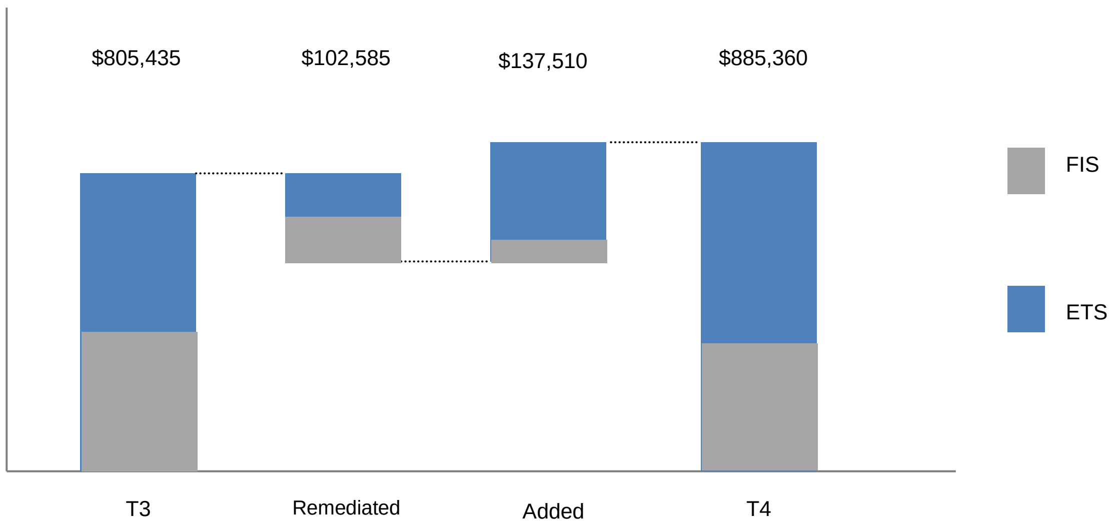
TECHNICAL DEBT = [(10% * L) + (20% * M) + (50% * H)] C * T]

Where:

- L is the Number of Low-Severity Violations
- M the Number of Medium-Severity Violations
- H the Number of High-Severity Violations
- C the Cost to Fix a Violation ($ per Hour); assumed to be $75.00 per hour
- T the Time to Fix a Violation (Number of Hours); assumed to be 1 hour per violation
- In the case of Core Applications, the values are: L = 1843; M = 4847; H = 1359
- TECHNICAL DEBT for Core Applications

2.2 Technical Debt Evolution Since Last Release

Figure 4: Technical debt evolution



3. Risk Profile of Core Applications

3.1 Violations per Core Application

We identified over 8000 violations of 221 quality rules in Core Applications. Not all of these violations have the same impact on the level of risk for the Core Applications. The table on the left shows the total number of violations has decreased by 9% since T3. On the right we have the critical violations which increased 11% since T3.

Table 3: Total violations by release

All Violations	ETS	FIS
T1	7,600	5,000
T2	7,200	5,000
T3	7,000	5,000
T4	5,500	4,000

9% reduction

Table 4: Critical violations by release

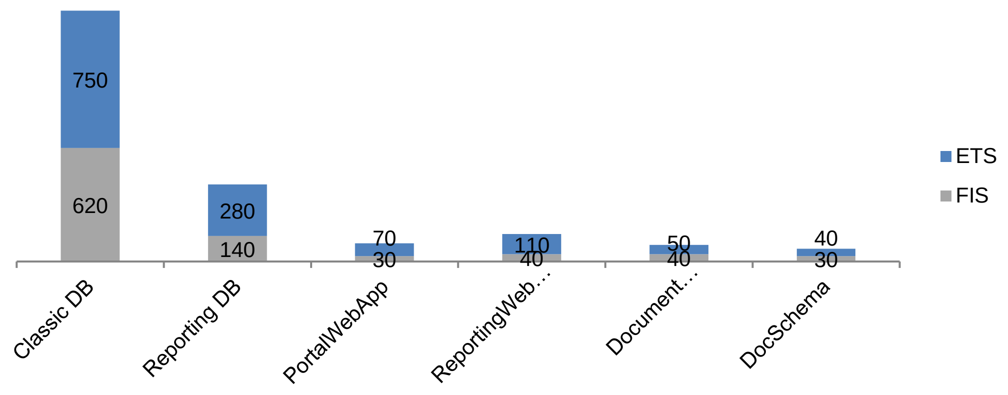
Critical Violation	ETS	FIS
T1	1,500	1,000
T2	1,300	800
T3	900	800
T4	800	800

11% increase

3.2 Violations per Application Layer

2200 of these violations are related to the 14 quality rules that are flagged as critical. Looking at the distribution of those violations it appears that the database layer and the presentation layer are concentrating 85% of those violations in 11 rules and the database layer is the one that reveals the highest number of violations per rule violated.

Figure 7: Violations by application layer



3.3 Aging Analysis of Critical Violations

Table 5: Aging analysis of critical violations

Application	Current	New	Existing		
	TOTAL	T4	T3	T2	T1
ETS	1300	250	410	110	530
FIS	900	170	230	280	220

3.4 Potential Points of Failure

3.4.1 Propagated Risk Index

Propagated Risk Index (PRI) is a measurement of the riskiest artifacts or objects of the application based on their contribution to application health and their impact on the rest of the application. The Top 10 objects with the highest PRI are:

Table 6: Top 10 objects with highest PRI

Object	PRI
[xxx.csi.architecture.common.iCSIException].xxx.csi.architecture.common.CSIExceptionManager.GetResolutionGin	214,692,660
[xxx.csi.architecture.common.iCSIException].xxx.csi.architecture.common.CSIExceptionManager.GetResolutionDetails	213,041,178
[xxxx.csi.express.common.iChbcommon].xxxx.csi.express.common.fclsOffering.UpdateOffering	110,865,755
[xxx.csi.architecture.common.iDRL].xxx.csi.architecture.common.iDRL.FileExists	38,774,778
[xxx.csi.architecture.common.iDRL].xxx.csi.architecture.common.iDRL.FileUpdate	38,247,230
[xxx.csi.architecture.common.iQuery].xxx.csi.architecture.common.iQuery.GetInformixErrorDetails	30,796,850
[xxx.csi.architecture.common.ICSILogin].xxx.csi.architecture.common.ICSILogin.StarSCIILogin	28,215,600
[xxx.csi.architecture.common.iQuery].xxx.csi.architecture.common.iQuery.ResetConnection	25,157,727
[xxx.csi.architecture.common.iQuery].xxx.csi.architecture.common.iQuery.GetInformixErrorMsg	24,637,480
GAIC04₩SQLGAIC.tcis..claimtbl	692,266

3.4.2 Transaction Risk Index

Transaction Risk Index (TRI) is an indicator of the riskiest transactions of the application. The Top 10 objects with the highest TRI are:

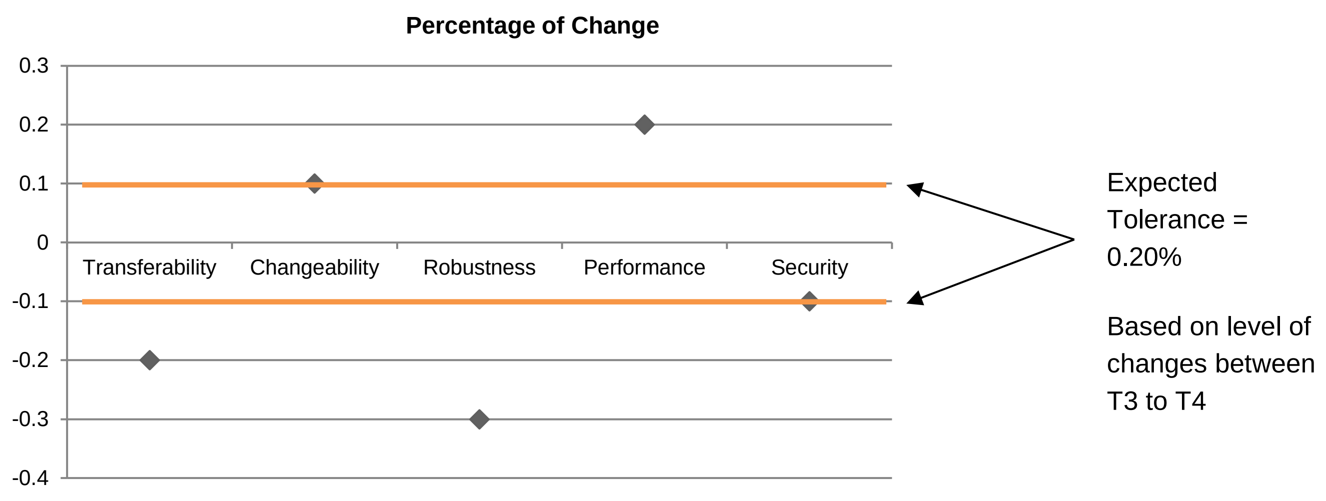
Table 7: Top 10 objects with the highest TRI

Transaction	TRI
frmBookPost	1,229,694,195
frmBindBook	1,169,694,195
frmMain	755,357,798
frm2Main	555,357,798
frmPurchase	302,730,366
frmOrganizeBehavior	301,730,366
mdiOpport	224,471,958
dbProdSheets	209,118,738
frmUpdateTransaction	121,471,958
dbCustTransaction	106,118,738

In summary, the assessment of Core Applications reveals several real areas, i.e. violations in quality areas for the application and database, objects with highest PRI, and several areas to reduce TRI for improvement to reduce both the cost and risk associated with the quality of the application.

3.5 Percent Change in Health Factors from T3 to T4

Figure 8: Percentage of change in health factors from previous release



There was slight deteoration in the overall Health Factors in the application compared to the previous release. The Health Factors of Changeability, Robustness and Security continue to be of concern. The number of additions for this release futher contributing to the risk and increasing the total cost of ownership (TCO) of the application.

- Performance improved at the top end of expectation
- Changeability, Robustness and Security took a larger drop than expected
- Changebility may have been compromised with all the changes in the latest release

3.6 Quick Wins to Improve Application Health

3.7 The Top Quick Win for Better Performance Efficiency

Table 8: Quick wins for improving performance

Criticality	Weight	Grade	Name	# Violation	#Ok
☐	7	3.25	Avoid Cursors inside a loop	7	1084
☐	7	3.25	Avoid using SQL queries inside a loop	37	1143
☐	4	1	Avoid direct definition of JavaScript Functions in a Web page	623	1024
☐	4	1	EJB Session access through their local interface	4	1
☐	1	1	Avoid direct definition of JavaScript Functions in a Web page	623	1024
	9	1.57	Avoid SQL queries that no index can support	335	660
	8	2.84	Avoid String concatenation in loops	40	7465
	8	4	Avoid artifacts having recursive calls	44	10865
	4	3.65	Avoid using Dynamic Instantiation	38	896
	4	3.99	Avoid using HashTable	63	7442
	1	1.43	Avoid to use Log.debug() without calling Log.isDebugEnables()	355	190

3.8 The Top Quick Win For Better Resilience

Table 9: Quick wins for improving resilience

Criticality	Weight	Grade	Name	# Violation	#Ok
☐	8	1	Avoid Functions and Procedures doing an Insert, Update, Delete, Create Table or XX	648	130
☐	6	3.46	Avoid empty catch blocks	92	9945
	9	4	Avoid thread creation for application running on application server	2	7690
	8	3.23	Avoid double checking locking	9	37
	8	3.85	Never exit a finally block with a return, break, continue or throw	4	473
	6	4	Avoid empty finally blocks	20	10368
	5	4	Check usage of '==' and '!=' on objects	48	7519
	1	1.31	Avoid Functions and Procedures doing an Insert, Update or Delete without XX	284	494
	0	2.21	Avoid using 'System.err' and System.out' within a try catch block	118	7387
	0	1.75	Avoid using 'System.printStackTrace()' within a try catch block	378	7127

3.9 The Top Quick Win For Better Security

Table 10: Quick wins for improving security

Criticality	Weight	Grade	Name	# Violation	#Ok
☐	10	2.5	Avoid cross-site scripting vulnerabilities	8	53
☐	10	2.5	Avoid file path manipulation vulnerabilities	1	60
	9	1.39	Avoid instance of methods that override or implement Object.equals(), XX	9	10
	9	3.78	Avoid using fields (non Static final) from other Classes	168	7337
	8	2.17	Favor PreparedStatement or CallableStatement over Statement	50	283
	8	1	Avoid fields in servlet classes that are not final static	16	1
	4	3.78	Avoid using fields (non Static final) from other Classes	168	7337

3.10 The Top Quick Win For Better Maintainability

Table 11: Quick wins for improving maintainability

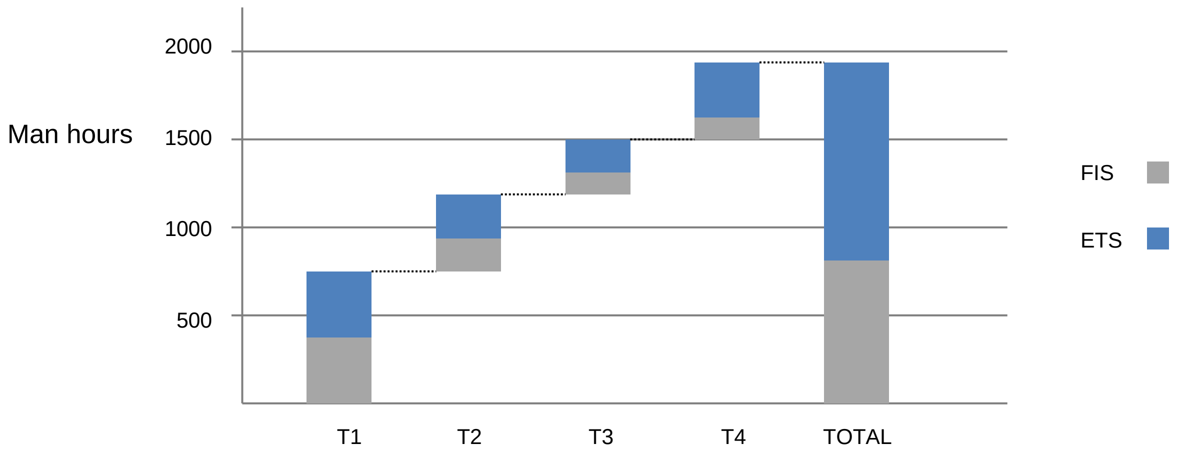
Criticality	Weight	Grade	Name	# Violation	#Ok
☐	9	2	Avoid classes overriding only equals() or only hashCode()	5	8
☐	9	2.75	Suspicious similar methods name or signature in an inheritance tree	17	1230
	8	2.34	Avoid Artifacts with high Commented-out Code Lines/Code Lines ratio	896	10672
	8	4	Avoid Artifacts having recursive calls	44	10865
	8	3.03	Avoid having multiple Artifacts inserting data on same SQL table	70	1015
	8	3.36	Avoid having multiple Artifacts updating data on same SQL table	41	1044
	7	3.9	Avoid Artifacts with High Essential Complexity	146	11071
	7	3.63	Avoid having multiple Artifacts deleting data on the same SQL table	36	1049
	6	3.82	Avoid Artifacts with High Depth of Code	134	11083
	6	3.01	Avoid Artifacts with High RAW SQL Complexity	85	1095

4. Team Performance

4.1 Estimated Effort – What it should have cost

The following chart indicates a reverse estimate of how many man days have been expended in enhancing the core applications:

Figure 9: Estimated effort based on the code added/modified/deleted

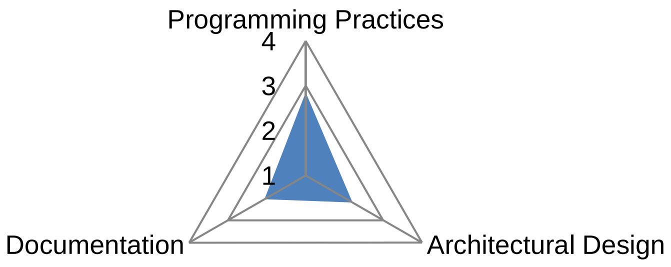


These data are calculated by combining a detailed view of all the components added, removed, or changed across the applications, sorted by complexity and technology. A complexity-technology-change type matrix is calibrated to reflect historical effort in man days. It is a rough estimate by definition, but it provides a good order of magnitude to compare to reported actuals. Major differences represent areas for further investigation.

4.2 Maturity Level of Software Engineering Competency

There are three primary areas of competency: Programming Practices, Architectural Deisgn, and Documentation that we examined. The core applications being measured demonstrate some deficiencies in all three areas.

Figure 10: Software engineering maturity level (1 – Low; 4 – High)



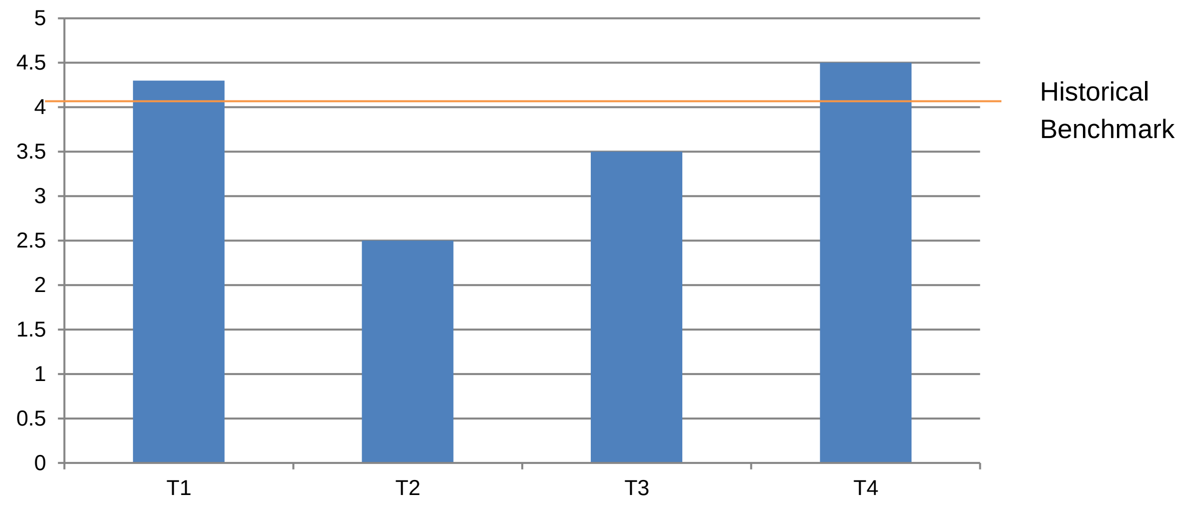
4.2.1 Scores for Best Practices

Table 12: Best practices scores

Best Practices	Grade	Description
Programming Practices	2.85	Measures the level of compliance of the application to coding best practices. Compliance to best practices reduces risks of failures in production and improves productivity through increased readability and reduced debugging.
Architectural Design	2.21	Measures the level of compliance of the application to software architecture and design rules. Compliance to architecture rules improves productivity through better use of existing frameworks and code and reduced debugging.
Documentation	2.06	Measures the level of compliance of the application to code documentation best practices. Compliance to documentation best practices improves productivity through increased readability and faster understanding of source code.

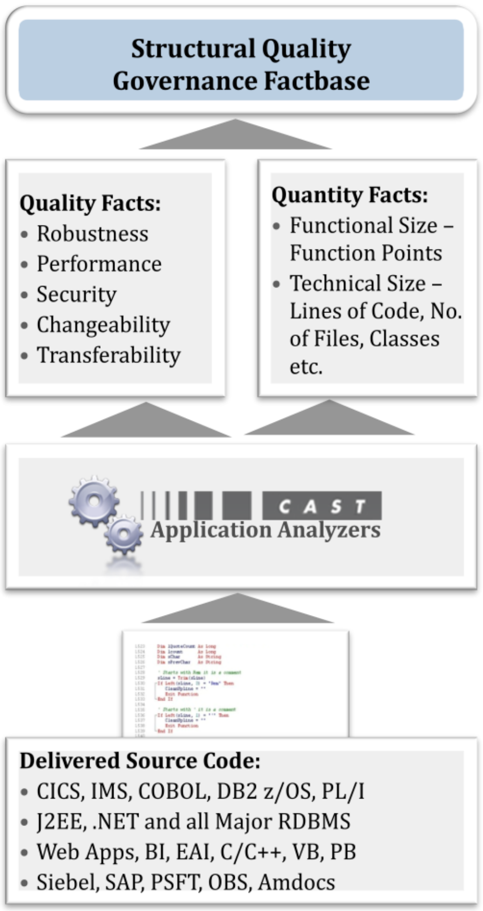
4.2.2 Software Weakness Injection Rate

Figure 11: New critical violations introduced per man-day estimate of effort



5. Appendix – Assessment Approach Overview

Figure 12: Assessment approach


Structural Quality
Governance Factbase
Quality Facts:
• Robustness
• Performance
• Security
• Changeability
• Transferability
Quantity Facts:
• Functional Size – Function Points
• Technical Size – Lines of Code, No. of Files, Classes etc.
CAST
Application Analyzers
Delivered Source Code:
• CICS, IMS, COBOL, DB2 z/OS, PL/I
• J2EE, .NET and all Major RDBMS
• Web Apps, BI, EAI, C/C++, VB, PB
• Siebel, SAP, PSFT, OBS, Amdocs

This assessment is an effort to determine the overall quality of the ACBA application and identify any risks that may be inherent in the application towards ACME Corp's objectives of extending the application. This assessment looks at the implementation of CORE APPLICATIONS to determine whether the application is constructed according to industry best practices, follows best practices for software engineering, and is maintainable. See Table 1.

This assessment is focused solely on the CORE APPLICATIONS and the SQL database with no view to functionality provided by backend services.

SCI uses the best-of-breed automated analysis platform, CAST AIP; to automatically scan the entire code base as well as having expert J2EE architects review the architecture, design, and code against current industry practices and standard approaches.

5.1 Automated Analysis

The CAST AIP is the industry leading automated code analysis platform, with coverage of all major development tools and languages. CAST AIP automatically scans and analyzes all of the source code and database elements that are part of an Enterprise system. CAST AIP applies over 900 metrics based on standards and measurements developed by the Software Engineering Institute (SEI), International Standards Organization (ISO), Consortium for IT Software Quality (CISQ), and Institute of Electrical and Electronics Engineers (IEEE). These metrics objectively measure software for the quality and quantity of work.

CAST AIP provides Application Analysts the ability to examine and drill down on critical application characteristics and attributes. The primary Application Health Factors that are addressed are:

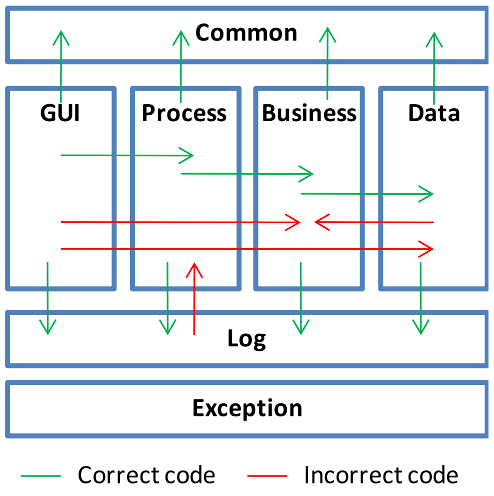
Table 13: CAST AIP health factor descriptions and business benefits of measuring them

Health Factor	Description	Example business benefits
Transferability	Attributes that allow new teams or members to quickly understand and work with an application	• Reduces inefficiency in transferring application work between teams • Reduces learning curves • Reduces lock-in to suppliers
Changeability	Attributes that make an application easier and quicker to modify	• Improves business agility in responding to markets or customers • Reduces cost of ownership by reducing modification effort
Robustness	Attributes that affect the stability of the application and the likelihood of introducing defects when modifying it	• Improves availability of the business function or service • Reduces risk of loss due to operational malfunction • Reduces cost of application ownership by reducing rework
Performance	Attributes that affect the performance of an application	• Reduces risk of losing customers from poor service or response • Improves productivity of those who use the application • Increases speed of making decisions and providing information • Improves ability to scale application to support business growth
Security	Attributes that affect an application's ability to prevent unauthorized intrusions	• Improves protection of competitive information-based assets • Reduces risk of loss in customer confidence or financial damages • Improves compliance with security-related standards and mandates

5.2 Architectural Analysis

CAST AIP Architecture Checker, helps translate clients target architecture intended at application design time into policies that can be tested during source code analysis. Any code developed that does not comply with the intended architecture is flaged for review and correction.

Figure 13: Overview of architectural analysis



SCI reviewed the overall design and architecture of the application along with the specific implementation choices embedded in the application code. These experienced J2EE Architects, with the help of "Architecture Checker" feature of CAST AIP platform, evaluated the architecture of CORE APPLICATIONS against technology patterns and best practices to evaluate the quality and long-term viability of the application. The review considered factors such as the ability of developers to understand and maintain the application code, the flexibility and expandability of the system, the reuse of existing libraries, and the use of standard techniques. While reviewing the code, the architects applied their hard-earned wisdom to identify common performance, security, and maintenance problems.

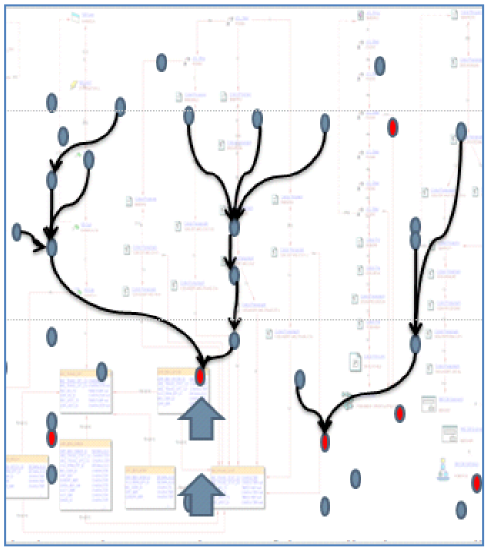
By considering alternative technologies and techniques available at the time of original development and at the time of assessment, and incorporating an understanding of the goals for the application, SCI provides a balanced assessment of whether the application under review is suitable as a strategic platform and the level of change necessary to achieve the enterprise goals.

In this Structural Quality Gate, SCI focuses particular attention on identifying risks that would prevent CORE APPLICATIONS from becoming a highly scalable, extensible mission-critical application, and whether the application follows industry-standard software engineering and security principles.

5.2.1 Propagated Risk Index Definition

Propagated Risk Index (PRI) enables easy identification of the riskiset objects/artifacts within the application

Figure 14: Overview of Propagated Risk Index (PRI)



The red object (above arrow) in this illustration has higher PRI because of more objects which depend on it

SCI reviewed the overall design and architecture of the application along with the specific implementation choices embedded in the application code. These experienced J2EE Architects, with the help of "Architecture Checker" feature of CAST AIP platform, evaluated the architecture of CORE APPLICATIONS against technology patterns and best practices to evaluate the quality and long term viability of the application. The review considered factors such as the ability of developers to understand and maintain the application code, the flexibility and expandability of the system, the reuse of existing libraries, and the use of standard techniques. While reviewing the code, the architects applied their hard-earned wisdom to identify common performance, security, and maintenance problems.

By considering alternative technologies and techniques available at the time of original development and at the time of assessment, and incorporating an understanding of the goals for the application, SCI provides a balanced assessment of whether the application under review is suitable as a strategic platform and the level of change necessary to achieve the enterprise goals.

In this Structural Quality Gate, SCI focuses particular attention on identifying risks that would prevent CORE APPLICATIONS from becoming a highly scalable, extensible mission-critical application, and whether the application follows industry-standard software engineering and security principles.

5.2.2 Transaction Risk Index Definition

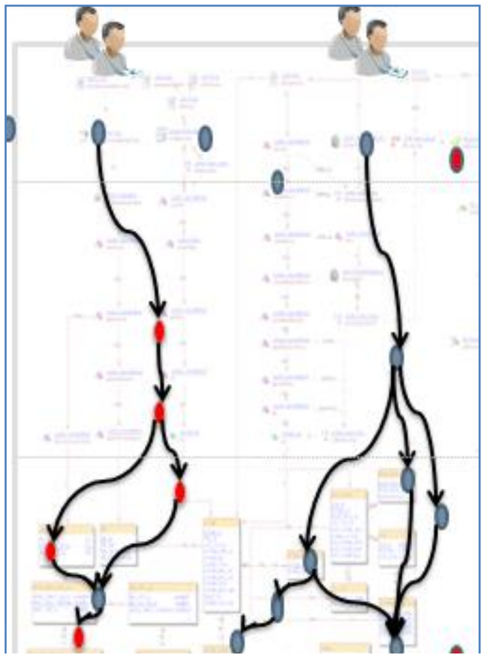
Transaction Risk Index (TRI) enables easy identification of the riskiset transactions within the application

Transaction Risk Index (TRI) is an indicator of the riskiest transactions of the application. The TRI number reflects the cumulative risk of the transaction based on the risk in the individual objects contributing to the transaction.

The TRI is calculated as a function of the rules violated, their weight/criticality, and the frequency of the violation across all objects in the path of the transaction.

TRI is a powerful metric to identify, prioritize and ultimately remediate riskiest transactions and their objects.

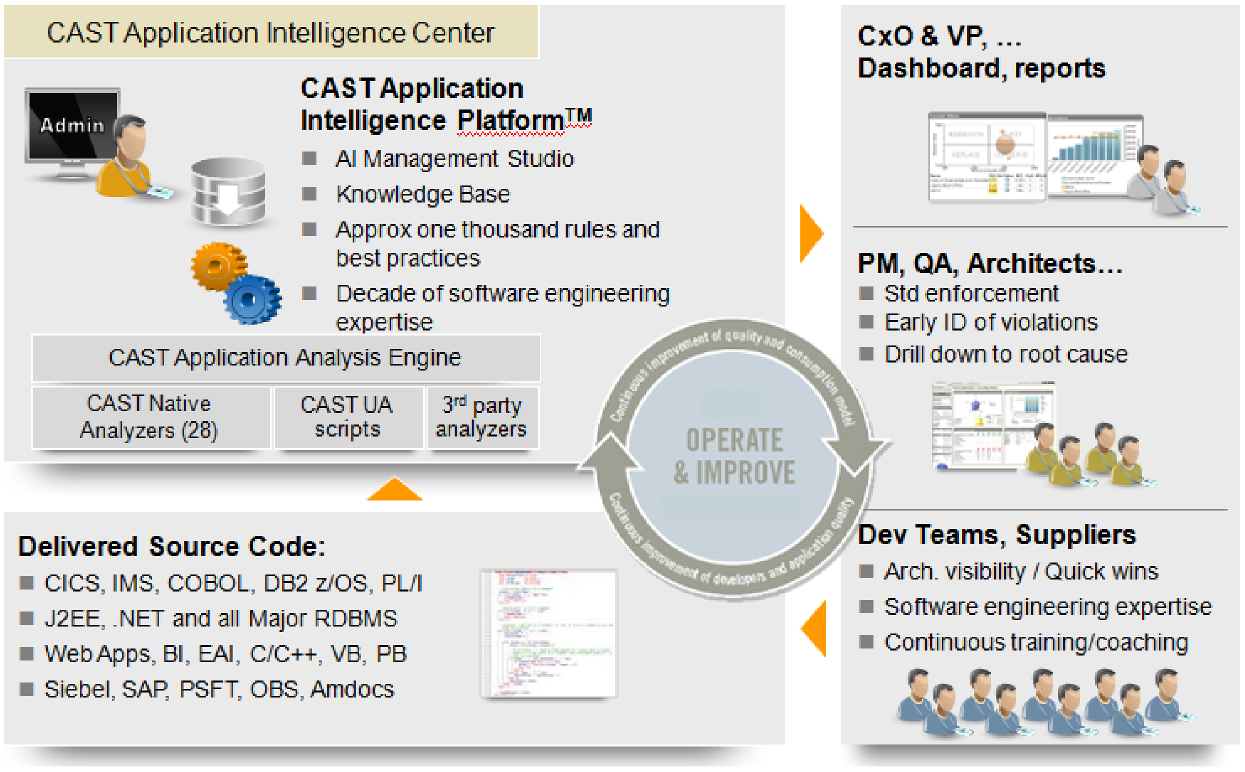
Figure 15: Overview of Transaction Risk Index (TRI)



5.3 The CAST Application Intelligence Platform

CAST plugs into all the major SCM systems, or can take source code in whatever format it is maintained in the organization. Source code is then processed and stored in the CAST Knowledge Base as metadata. That metadata then forms the basis for all the analysis and information provided by the CAST AI Platform. CAST looks at the entire application – even legacy components, packaged app customizations, and of course all the modern distributed technology environments. Data from third party code analyzers (like open source analyzers) can be integrated into CAST knowledge base and displayed in the AIP dashboards.

Figure 16: Working with CAST AIP


CAST Application Intelligence Center
Admin
CAST Application Intelligence Platform™
AI Management Studio
Knowledge Base
Approx one thousand rules and best practices
Decade of software engineering expertise
CAST Application Analysis Engine
CAST Native Analyzers (28)
CAST UA scripts
3rd party analyzers
Delivered Source Code:
CICS, IMS, COBOL, DB2 z/OS, PL/I
J2EE, .NET and all Major RDBMS
Web Apps, BI, EAI, C/C++, VB, PB
Siebel, SAP, PSFT, OBS, Amdocs
OPERATE & IMPROVE
Continuous improvement of quality and consumption model
Continuous improvement of developers and application quality
CxO & VP, …
Dashboard, reports
PM, QA, Architects…
Std enforcement
Early ID of violations
Drill down to root cause
Dev Teams, Suppliers
Arch. visibility / Quick wins
Software engineering expertise
Continuous training/coaching

5.4 The CAST Quality Model

CAST AIP can apply over 1000 metrics based on standards and measurements developed by the Software Engineering Institute (SEI), International Standards Organization (ISO), Center for Software Engineering, and Institute of Electrical and Electronics Engineers (IEEE), R&D compiler and RDBMS, publishers, all sorts of literatures and more recently, work done by the newly created Consortium for IT Software Quality, a child of the SEI and OMG partnership. These metrics objectively measure any complex business system, structurally speaking.