|  |
| --- |
| Assessment Report Sample 2 |
| Monday, xx July 2012  My Application Name  Version: Version Number |
| PURPOSE OF DOCUMENT:  Purpose of this report is to provide an objective assessment of the quality of the deliverables. The primary audience for this report is IT executives who are responsible for My Application Name application.   |  | | --- | |  | |

Table of Contents

[1. Executive Summary 3](#_Toc345516134)

[1.1. Application Characteristics 3](#_Toc345516135)

[1.2. Summary of Quality Indicators 3](#_Toc345516136)

[1.3. Assessment Highlights 3](#_Toc345516137)

[2. Technical Debt 4](#_Toc345516138)

[3. How Can Technology Address Application Quality Challenges? 5](#_Toc345516139)

[3.1. Potential Points of Failures: Critical rules 6](#_Toc345516140)

[3.2. Potential Points of Failures: Transaction wide Risk Index 6](#_Toc345516141)

[3.3. Potential Point of Failures: Propagated Risk Index 7](#_Toc345516142)

[3.4. Strenghts and weaknesses 8](#_Toc345516143)

[4. Measures of Robustness 9](#_Toc345516144)

[5. Measures of Performance & Efficiency 13](#_Toc345516145)

[6. Measures of Security 17](#_Toc345516146)

[7. Measures of Maintainability 21](#_Toc345516147)

[8. Appendix - Assessment Approach Overview 25](#_Toc345516148)

[9. Appendix: Understanding Quality Indicators, Quality Rules 26](#_Toc345516149)

[10. Appendix: Importance of measuring all layers of an application 28](#_Toc345516150)

[10.1. Bypassing the Architecture. 28](#_Toc345516151)

[10.2. Failure to Control Processing Volumes. 28](#_Toc345516152)

[10.3. Application Resource Imbalances. 28](#_Toc345516153)

[10.4. Security Weaknesses. 28](#_Toc345516154)

[10.5. Lack of Defensive Mechanisms. 28](#_Toc345516155)

[11. Appendix: Technical Debt Calculation in the CAST AIP 30](#_Toc345516156)

[11.1. Purpose 30](#_Toc345516157)

[11.2. Calculation of Technical Debt per Module and Application 30](#_Toc345516158)

[11.3. Definition of Variables 31](#_Toc345516159)

# Executive Summary

The Application Assessment evaluates the overall quality of the My Application Name application.

My Application Name is a Small/Medium/Large/ExtraLarge application and has a VeryLow/Low/Medium/Good/VeryGood quality with a ***Total Quality Indicator (TQI) of 0.00 on a scale of 4.*** Each of the additional health metrics and their scores are identified below.

## Application Characteristics

**Top 5 Technologies**

|  |  |
| --- | --- |
| Name | LOC |
| Techno 1 | 000,000 |
| Techno 2 | 000,000 |
| Techno 3 | 000,000 |
| Techno 4 | 000,000 |
| Techno 5 | 000,000 |

**Technical Size**

|  |  |
| --- | --- |
| Name | Number |
| kLOCs | 000 |
| Files | 0,000 |
| Classes | 0,000 |
| SQL Art. | 00 |
| Tables | 00 |

## Summary of Quality Indicators

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | TQI | Robu | Perf | Secu | Trans | Chang |
| Curr. version | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** |
| Prev. version | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Variation | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |

## Assessment Highlights

**Statistics on Violations**

|  |  |
| --- | --- |
| Name | Number |
| Critical Violations | 0,000 |
| per File | 0.00 |
| per kLOCs | 0.00 |
| Complex Objects | 0,000 |
| with violations | 000 |

**Top 10 Critical Violations**

|  |  |
| --- | --- |
| Rules | Count |
| Rule 1 | 0 |
| Rule 2 | 0 |
| Rule 3 | 0 |
| Rule 4 | 0 |
| Rule 5 | 0 |
| Rule 6 | 0 |
| Rule 7 | 0 |
| Rule 8 | 0 |
| Rule 9 | 0 |
| Rule 10 | 0 |

# Technical Debt

The complexity of My Application Name application has been converted into Technical Debt – the cost of fixing the structural quality violations that cause serious business disruption. The data on technical debt provides an objective, empirical frame of reference for the developer community. They also provide a platform for characterizing the management tradeoffs between expending resources on correcting weaknesses in the source code versus risking the problems these flaws may cause such as outages or security breaches.

If necessary, this value can be benchmarked with The CAST Appmarq repository. The CAST Appmarq repository provides a unique opportunity to calculate the Technical Debt across different technologies, based on the number of engineering flaws and violations of good architectural and coding practices in the source code. A parameterized formula for calculating the Technical Debt of each application in Appmarq is based on the percent of violations to be remediated at each level of severity, the time required to fix a violation, and the burdened hourly rate of a developer. Please see the Appendix for details on how Technical Debt is calculated at CAST.

CAST 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 of My Application Name = **0.00**

# How Can Technology Address Application Quality Challenges?

The quality attributes of an application can be characterized by the quality attributes of its component parts no more than the attributes of a molecule can be characterized by the attributes of its constituent atoms. Since high quality components do not equate to a high quality system in any field of engineering, code quality, although necessary, is not sufficient to ensure high quality applications. Organizations need the help of application quality diagnostic tools which can discover inter-component issues and measure the internal quality of the application across its tiers.

There are numerous commercial, freeware, and open source tools available that measure code quality specific to a programming language and are often integrated into Integrated Development Environments (IDEs). These tools are becoming standard components of every developer’s toolset since they provide quick feedback during the coding and unit test process. However, these tools are not sufficient to address application quality since they cannot evaluate interactions across the various languages, technologies, and tiers of an application.

Technology that measures application quality analyzes the integrated software produced by a build once the code is checked into a central repository by all the developers. In addition to analyzing each component, application quality technology analyzes their interactions for the types of problems described in earlier sections. Moreover, application quality trends can be compared across builds or releases to monitor the progress against application quality objectives and evaluate the risks posed by the application.

Application quality measurement tools provide several benefits for both the development team and management:

* ***Visibility across application(s):*** Consistent and continuous analysis of all core business applications provides executives with the metrics and information needed to better manage their portfolio of applications and projects.
* ***Analysis of the internal quality of an application***: Reviewing the integrated software system for quality in order to detect architectural and structural problems that hide in interactions between tiers, provides application or project managers with continual status about application quality and risk.
* ***Team performance:*** Since a detailed knowledge of the whole system is usually beyond any individual developer’s capabilities, analyzing application quality helps improves developer skills, the team’s breadth of application knowledge, and the efficiency of team performance.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | TQI | Robu | Perf | Secu | Trans | Chang |
| Curr. Vers | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** |
| Prev. Vers | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** |
| Variation | **0.00 %** | **0.00 %** | **0.00 %** | **0.00 %** | **0.00 %** | **0.00 %** |

**Statistics on Violations**

|  |  |
| --- | --- |
| Name | Number |
| Critical Violations | 0,000 |
| per File | 0.00 |
| per kLOCs | 0.00 |
| Complex Objects | 0,000 |
| with violations | 000 |

A dynamic business environment, new technology, and multiple sourcing options, amplify the complexity of business application software. Since even the most talented developers can no longer know all the nuances of all the different languages, technologies, and tiers in an application, their capability needs to be augmented by automated tools to evaluate the entire application. Without such assistance, defects hidden in the interactions between application tiers will place the business at risk for the outages, degraded service, security breaches, and corrupted data that are caused by poor quality applications.

## Potential Points of Failures: Critical rules

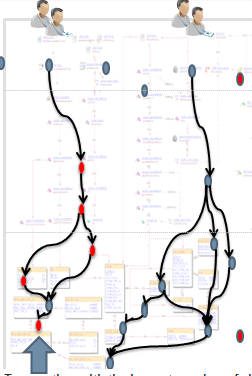
The CAST AIP quality model assess automatically the application and raise the main issue of the application through a weighted aggregation of more than +1000 rules across the different technology. The below list represent the different rules which contain some violation on some component which can create some abnornal behavior during the execusion of the application.

**Top 10 Critical Violations**

|  |  |
| --- | --- |
| Rules | Count |
| Rule 1 | 0 |
| Rule 2 | 0 |
| Rule 3 | 0 |
| Rule 4 | 0 |
| Rule 5 | 0 |
| Rule 6 | 0 |
| Rule 7 | 0 |
| Rule 8 | 0 |
| Rule 9 | 0 |
| Rule 10 | 0 |

## Potential Points of Failures: Transaction wide Risk Index

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



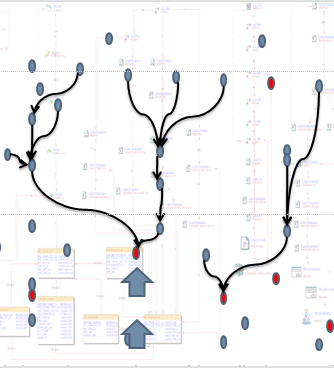
Transaction Wide Risk Index (TwRI) is an indicator of the riskiest transactions of the application.  The TwRI number reflects the cumulative risk of the transaction based on the risk in the individual objects contributing to the transaction; in the below list the focus is on the efficiency of the application. The TwRI 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. TwRI is a powerful metric to identify, prioritize and ultimately remediate riskiest transactions and their objects.

|  |  |
| --- | --- |
| Artefact name | TwRI |
| Artefact one | Twri value 1 |
| Artefact two | Twri value 2 |

Potential Points

## Potential Point of Failures: Propagated Risk Index

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



Propagated Risk Index (PRI) is a measurement of the riskiest artifacts or objects of the application along the Health Factors of Robustness, Performance and Security.

PRI takes into account the intrinsic risk of the component coupled with the level of use of the given object in the transaction. It systematically helps aggregate risk of the application in a relative manner allowing for identification, prioritization, and ultimately remediation of the riskiest objects.

The PRI number reflects the cummulative risk of the object based on its relationships and interdependencies. The PRI is calculated as a function of the rules violated, their weight/criticality, and the frequency of the violation.

The Top 15 objects with the highest PRI are:

|  |  |
| --- | --- |
| Artefact name | PRI |
| Artefact one | PRI value 1 |
| Artefact two | PRI value 2 |

## Strenghts and weaknesses

|  |  |  |
| --- | --- | --- |
| Technical Criteria Name | Grade | Evolution |
| Criteria 1 | 2.2 | 0.00 % |
| Criteria 2 | 1.4 | 0.00 % |
| Criteria 3 | 3.23 | 0.00 % |
| Criteria 4 | 3.45 | 0.00 % |
| Criteria 5 | 2.2 | 0.00 % |
| Criteria 6 | 1.4 | 0.00 % |
| Criteria 7 | 3.23 | 0.00 % |
| Criteria 8 | 3.45 | 0.00 % |
| Criteria 9 | 2.3 | 0.00 % |
| Criteria 10 | 3.5 | 0.00 % |

# Measures of Robustness

#### The root causes of poor reliability are found in a combination of non- compliance with good architectural and coding practices. This non-compliance can be detected by measuring the static quality attributes of an application. Assessing the static attributes underlying an application’s reliability provides an estimate of the level of business risk and the likelihood of potential application failures and defects the application will experience when placed in operation.

|  |  |  |
| --- | --- | --- |
| Technical Criteria Name | Grade | Evolution |
| Criteria 1 | 2.2 | 0.00 % |
| Criteria 2 | 1.4 | 0.00 % |
| Criteria 3 | 3.23 | 0.00 % |
| Criteria 4 | 3.45 | 0.00 % |
| Criteria 5 | 2.2 | 0.00 % |
| Criteria 6 | 1.4 | 0.00 % |
| Criteria 7 | 3.23 | 0.00 % |
| Criteria 8 | 3.45 | 0.00 % |
| Criteria 9 | 3.5 | 0.00 % |
| Criteria 10 | 2.3 | 0.00 % |

#### Measures: Assessing reliability requires checks of at least the following software engineering best practices and technical attributes:

* Application Architecture Practices
  + Multi-layer design compliance
  + Coupling ratio
  + Component or pattern re-use ratio
* Coding Practices
  + Error & Exception handling (for all layers GUI, Logic & Data)
  + Compliance with Object-Oriented and Structured Programming best practices (when applicable)
* Complexity
  + Transaction complexity level
  + Complexity of algorithms
  + Complexity of programming practices
  + Dirty programming

Depending on the application architecture and the third-party components used (such as external libraries or frameworks), custom checks should be defined along the lines drawn by the above list of best practices to ensure a better assessment of the reliability of the delivered software.

**Top 10 Non Critical Violations**

|  |  |
| --- | --- |
| Rules | TwRI |
| Rule 1 | 0 |
| Rule 2 | 0 |
| Rule 3 | 0 |
| Rule 4 | 0 |
| Rule 5 | 0 |

# Measures of Performance & Efficiency

As with Reliability, the causes of performance inefficiency are often found in violations of good architectural and coding practice which can be detected by measuring the static quality attributes of an application. These static attributes predict potential operational performance bottlenecks and future scalability problems, especially for applications requiring high execution speed for handling complex algorithms or huge volumes of data.

|  |  |  |
| --- | --- | --- |
| Technical Criteria Name | Grade | Evolution |
| Criteria 1 | 2.2 | 0.00 % |
| Criteria 2 | 1.4 | 0.00 % |
| Criteria 3 | 3.23 | 0.00 % |
| Criteria 4 | 3.45 | 0.00 % |
| Criteria 5 | 2.2 | 0.00 % |
| Criteria 6 | 1.4 | 0.00 % |
| Criteria 7 | 3.23 | 0.00 % |
| Criteria 8 | 3.45 | 0.00 % |
| Criteria 9 | 2.3 | 0.00 % |
| Criteria 10 | 3.5 | 0.00 % |

#### Measures: Assessing performance efficiency requires checking at least the following software engineering best practices and technical attributes:

* Application Architecture Practices
  + Appropriate interactions with expensive and/or remote resources
  + Data access performance and data management
  + Memory, network and disk space management
* Coding Practices
  + Compliance with Object-Oriented and Structured Programming best practices
  + Compliance with SQL best practices

# Measures of Security

Most security vulnerabilities result from poor coding and architectural practices such as SQL injection or cross-site scripting. These are well documented in lists maintained by CWE http://cwe.mitre.org/, and CERT.

|  |  |  |
| --- | --- | --- |
| Technical Criteria Name | Grade | Evolution |
| Criteria 1 | 2.2 | 0.00 % |
| Criteria 2 | 1.4 | 0.00 % |
| Criteria 3 | 3.23 | 0.00 % |
| Criteria 4 | 3.45 | 0.00 % |
| Criteria 5 | 2.2 | 0.00 % |
| Criteria 6 | 1.4 | 0.00 % |
| Criteria 7 | 3.23 | 0.00 % |
| Criteria 8 | 3.45 | 0.00 % |
| Criteria 9 | 2.3 | 0.00 % |
| Criteria 10 | 3.5 | 0.00 % |

# Measures of Maintainability

Maintainability includes concepts of modularity, clarity, changeability, testability, reusability, and transferability from one development team to another. These do not take the form of critical issues at the code level. Rather, poor maintainability is typically the result of thousands of minor violations with best practices around documentation, complexity avoidance strategy, and basic programming practices that make the difference between clean and easy to read code vs. ugly and difficult to read code.

***Transferability***

|  |  |  |
| --- | --- | --- |
| Technical Criteria Name | Grade | Evolution |
| Criteria 1 | 2.2 | 0.00 % |
| Criteria 2 | 1.4 | 0.00 % |
| Criteria 3 | 3.23 | 0.00 % |
| Criteria 4 | 3.45 | 0.00 % |
| Criteria 5 | 2.2 | 0.00 % |
| Criteria 6 | 1.4 | 0.00 % |
| Criteria 7 | 3.23 | 0.00 % |
| Criteria 8 | 3.45 | 0.00 % |
| Criteria 9 | 2.3 | 0.00 % |
| Criteria 10 | 3.5 | 0.00 % |

***Changeability***

|  |  |  |
| --- | --- | --- |
| Technical Criteria Name | Grade | Evolution |
| Criteria 1 | 2.2 | 0.00 % |
| Criteria 2 | 1.4 | 0.00 % |
| Criteria 3 | 3.23 | 0.00 % |
| Criteria 4 | 3.45 | 0.00 % |
| Criteria 5 | 2.2 | 0.00 % |
| Criteria 6 | 1.4 | 0.00 % |
| Criteria 7 | 3.23 | 0.00 % |
| Criteria 8 | 3.45 | 0.00 % |
| Criteria 9 | 2.3 | 0.00 % |
| Criteria 10 | 3.5 | 0.00 % |

#### Measures: Assessing maintainability requires checking the following software engineering best practices and technical attributes:

* Application Architecture Practices
  + Multi-layer design compliance
  + Coupling ratio
  + Component or pattern re-use ratio
* Programming Practices (code level)
  + Compliance with Object-Oriented and Structured Programming best practices (when applicable)
* Complexity
  + Complexity level of transactions
  + Complexity of algorithms
  + Complexity of programming practices
  + Dirty programming
* Documentation
  + Code readability
  + Architecture, Programs and Code documentation embedded in source code
  + Source code file organization cleanliness
* Portability
  + Hardware, OS, middleware, software components and database independence

# Appendix - Assessment Approach Overview

This assessment is an effort to determine the overall quality of the application My Application Name and identify any risks that may be inherent in the application. The assessment looks at the implementation of My Application Name to determine whether the application is constructed according to industry best practices, follows best practices for software engineering, and is maintainable.

**Table 1:** Health Factor descriptions and business benefits of measuring them

This assessment is focused solely on the Source code and Database structure with no view to functionality provided by backend services.

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 1000+ 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:

|  |  |  |
| --- | --- | --- |
| Health Factor | Description | Example business benefits |
| 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 |
| Efficiency | 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 |
| 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 |

# Appendix: Understanding Quality Indicators, Quality Rules

CAST AIP has 1000+ quality rules and each rule produces a Grade. Depending on the impact the grades are aggregated into high level Indicators: **Quality indicators** and **Best practices indicators**.

Each aggregation is a weighted average of the contributing metrics grades where certain metric grades are flagged critical, i.e. it is nearly a defect. We talk about **Critical Violations**.

#### Quality Indicators

The structure, classification and terminology are from the ISO 9126‐3 and the subsequent ISO 25000:2005 quality model. The main focus is on internal structural quality. Subcategories have been created to handle specific areas like business application architecture and technical characteristics such as data access and manipulation or the notion of transactions. The dependence tree between software quality characteristics and their measurable attributes is represented in the following diagram, where each of the 5 characteristics that matter for the user or owner of the business system depends on measurable attributes: Application Architecture Practices, Coding Practices, Application Complexity, Documentation, Portability, and Technical & Functional Volume.

|  |  |
| --- | --- |
| Quality Indicator | Description |
| Performance / Efficiency | The source code and software architecture attributes are the elements that ensure high performance once the application is in run‐time mode. Efficiency is especially important for applications in high execution speed environments such as algorithmic or transactional processing where performance and scalability are paramount. An analysis of source code efficiency and scalability provides a clear picture of the latent business risks and the harm they can cause to customer satisfaction due to response‐time degradation. |
| Robustness / Reliability | An attribute of resiliency and structural solidity. Reliability measures the level of risk and the likelihood of potential application failures. It also measures the defects injected due to modifications made to the software (its “stability” as termed by ISO). The goal for checking and monitoring Reliability is to reduce and prevent application downtime, application outages and errors that directly affect users, and enhance the image of IT and its impact on a company’s business performance. |
| Security | A measure of the likelihood of potential security breaches due to poor coding and architectural practices. This quantifies the risk of encountering critical vulnerabilities that damage the business and provides a list of prevention measures. |
|  |  |
| Transferability | The effort necessary to diagnose the cause of a failure or section of code to be modified. It establishes the level of dependency on specific developers |
| Changeability | The effort necessary to modify the source code. It establishes the level of responsiveness to business-driven change requests |
| TQI | Total Quality Index (TQI) is computed on all the measures made by the CAST AIP |

#### Best practices Indicators

|  |  |
| --- | --- |
| Health Factor | Description |
| Programming Practices | 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 | 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 | 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. |

The risk level of a grade shall be assessed according to the below scale

|  |  |
| --- | --- |
| Scale | Risk Level |
| 4 | Low Risk |
| 3 | Moderate Risk |
| 2 | High Risk |
| 1 | Very High Risk |

# Appendix: Importance of measuring all layers of an application

Measuring the technical quality of business software applications is evolving from an art to a science with the availability of software tools that automate the process of code analysis. However, it is critical to understand that there are two categories of software quality with very different implications for operational performance. The first category is Code Quality which measures individual or small collections of coded components written in a single language and occupying a single tier (e.g., user interface, logic, or data) in an application. The second category, Application Quality, analyzes the software across all of the application’s languages, tiers, and technologies to measure how well all an application’s components come together to create its operational performance and overall maintainability.

Although the code quality of individual components is important, by itself it will not ensure the overall quality of the application. Quality is not an intrinsic property of code: the exact same piece of code can be excellent in quality or highly dangerous depending on the context in which it operates. Ignoring the larger context in which the code operates – the multitude of connections with other code, databases, middleware, and APIs – will often generate a large number of false positives.

Today’s business applications are complex, built in multiple languages on multiple technologies. Even more challenging, these applications usually interact with other applications built on different technologies. Analyzing the quality of modern applications is monstrously complex and can only be accomplished with automated software that analyzes the inner structure of all components and evaluates their interactions in the context of the entire business application.

Typical application quality problems are listed below to clarify the distinction between application and code quality. Performance testing alone is not sufficient to detect these application quality problems.

## Bypassing the Architecture.

Components in one tier of a multi-tier application are typically designed to access components in another tier only through an intermediate “traffic management” component. Bypassing this traffic management component will usually result in a cascade of problems.

## Failure to Control Processing Volumes.

Applications can behave erratically when they fail to control the amount of data or processing they allow. This problem is often caused by a failure to incorporate controls in each of several different architectural tiers.

## Application Resource Imbalances.

When database resources in a connection pool are mismatched with the number of request threads from an application, resource contention will block the threads until a resource becomes available, tying up CPU resources with the waiting threads and slowing application response times to a crawl.

## Security Weaknesses.

Applications are vulnerable to security attacks when they lack appropriate sanitization checks on user inputs in all relevant tiers of the application.

## Lack of Defensive Mechanisms.

Since the developers implementing one tier cannot anticipate every situation, they must implement defensive code that sustains the application’s performance in the face of stresses or failures affecting other tiers. Tiers that lack these defensive structures are fragile because they fail to protect themselves from problems in their interaction with other tiers. Each of these application quality problems will result in unpredictable application performance, business disruption, data corruption, and make it difficult to alter the application in response to pressing business needs. Reliably detecting these problems requires an analysis of each application component in the context of the entire application as a whole – an evaluation of application rather than code quality.

# Appendix: Technical Debt Calculation in the CAST AIP

## Purpose

Purpose of this specification is to add new indicators to the CAST AIP dashboard.

1. **Total Technical Debt per Application**
2. **Total Technical Debt per Module**
3. **Technical Debt Added in Current Release of the Application**
4. **Technical Debt Removed in Current Release of the Application**

Note: These should be calculated at module and application level and can be summed up to the system level in the portal.

## Calculation of Technical Debt per Module and Application

1. **Total Technical Debt per Module and Application =**

**{** (% of low severity violations to be fixed X # of low severity violations in Application and Module) X (Weighted time, in hours, for fixing low severity violations) +

(% of medium severity violations to be fixed X # of medium severity violations in Application and Module) X (Weighted time, in hours, for fixing medium severity violations) +

(% of high severity violations to be fixed X # of high severity violations in Application and Module) X (Weighted time, in hours, for fixing high severity violations) **}** X

Cost per staff hour to fix violations

1. **Technical Debt Added in Current Release per Application =**

**{** (% of low severity violations to be fixed X # of low severity violations added in current release of Application) X (Weighted time, in hours, for fixing low severity violations) +

(% of medium severity violations to be fixed X # of medium severity violations added in current release of Application) X (Weighted time, in hours, for fixing medium severity violations) +

(% of high severity violations to be fixed X # of high severity violations added in current release of Application) X (Weighted time, in hours, for fixing high severity violations) **}** X

Cost per staff hour to fix violations

1. **Technical Debt Removed in Current Release per Application =**

**{** (% of low severity violations to be fixed X # of low severity violations removed in current release of Application) X (Weighted time, in hours, for fixing low severity violations) +

(% of medium severity violations to be fixed X # of medium severity violations removed in current release of Application) X (Weighted time, in hours, for fixing medium severity violations) +

(% of high severity violations to be fixed X # of high severity violations removed in current release of Application) X (Weighted time, in hours, for fixing high severity violations) **}** X

Cost per staff hour to fix violations

## Definition of Variables

| **Variable Name** | **Description** | **Configurable** | **Default Value** |
| --- | --- | --- | --- |
| % of low severity violations to be fixed | Only a portion of the low severity violations will be fixed | Yes | 0% |
| # of low severity violations | Actual # of low severity (level 1,2,3) violations across all health factors | No (comes directly from analysis) | Not Applicable |
| # of low severity violations added in current release | Actual # of low severity (level 1,2,3) violations across all health factors added in current release | No (comes directly from analysis) | Not Applicable |
| # of low severity violations removed in current release | Actual # of low severity (level 1,2,3) violations across all health factors removed in current release | No (comes directly from analysis) | Not Applicable |
| % of medium severity violations | Only a portion of the medium severity violations will be fixed | Yes | 50% |
| # of medium severity violations | Actual # of medium severity (level 4,5,6) violations across all health factors | No (comes directly from analysis) | Not Applicable |
| # of medium severity violations added in current release | Actual # of medium severity (level 4,5,6) violations across all health factors added in current release | No (comes directly from analysis) | Not Applicable |
| # of medium severity violations removed in current release | Actual # of medium severity (level 4,5,6) violations across all health factors added in current release | No (comes directly from analysis) | Not Applicable |
| % of high severity violations | Only a portion of the high severity violations will be fixed | Yes | 100% |
| # of high severity violations | Actual # of high severity (level 7,8,9) violations across all health factors | No (comes directly from analysis) | Not Applicable |
| # of high severity violations added in current release | Actual # of high severity (level 7,8,9) violations across all health factors added in current release | No (comes directly from analysis) | Not Applicable |
| # of high severity violations removed in current release | Actual # of high severity (level 7,8,9) violations across all health factors added in current release | No (comes directly from analysis) | Not Applicable |
| Weighted time, in hours, for fixing **LOW** severity violation | Not all violations will need the same amount of time, hence we take the weighted time to fix the violations. Weighted based on the distribution of level of difficulty to fix violations. Violations will be categorized as follows:   1. Easy 2. Hard 3. Very Hard   Wt. time to fix low severity violations=  (Low\_%Easy **X** Low\_Time\_Easy) **+**  (Low\_%Hard **X** Low\_Time\_Hard) **+**  **(**Low\_%Very\_Hard **X** Low\_Time\_Very\_Hard) | | |
| Low\_%Easy = % of violations which are “Easy” | Yes | 90% |
| Low\_Time\_Easy = Time take for fixing “Easy” violations | Yes | 0.5 hour |
| Low\_%Hard = % of violations which are “Hard” | Yes | 9% |
| Low\_Time\_Hard = Time take for fixing “Hard” violations | Yes | 1 hour |
| Low\_%Very\_Hard = % of violations which are “Very\_Hard” | Yes | 1% |
| Low\_Time\_Very\_Hard = Time take for fixing “Very\_Hard” violations | Yes | 8 hours |
| Weighted time, in hours, for fixing **MEDIUM** severity violation | Not all violations will need the same amount of time, hence we take the weighted time to fix the violations. Weighted based on the distribution of level of difficulty to fix violations. Violations will be categorized as follows:   1. Easy 2. Hard 3. Very Hard   Wt. time to fix low severity violations=  (Medium\_%Easy **X** Medium\_Time\_Easy) **+**  (Medium\_%Hard **X** Medium\_Time\_Hard) **+**  **(**Medium\_%Very\_Hard **X** Medium\_Time\_Very\_Hard) | | |
| Medium\_%Easy = % of violations which are “Easy” | Yes | 90% |
| Medium\_Time\_Easy = Time take for fixing “Easy” violations | Yes | 0.5 hour |
| Medium \_%Hard = % of violations which are “Hard” | Yes | 9% |
| Medium \_Time\_Hard = Time take for fixing “Hard” violations | Yes | 4 hour |
| Medium \_%Very\_Hard = % of violations which are “Very\_Hard” | Yes | 1% |
| Medium \_Time\_Very\_Hard = Time take for fixing “Very\_Hard” violations | Yes | 16 hours |
| Weighted time, in hours, for fixing **HIGH** severity violation | Not all violations will need the same amount of time, hence we take the weighted time to fix the violations. Weighted based on the distribution of level of difficulty to fix violations. Violations will be categorized as follows:   1. Easy 2. Hard 3. Very Hard   Wt. time to fix low severity violations=  (High\_%Easy **X** High \_Time\_Easy) **+**  (High \_%Hard **X** High \_Time\_Hard) **+**  **(**High \_%Very\_Hard **X** High \_Time\_Very\_Hard) | | |
| High \_%Easy = % of violations which are “Easy” | Yes | 80% |
| High \_Time\_Easy = Time take for fixing “Easy” violations | Yes | 1 hour |
| High \_%Hard = % of violations which are “Hard” | Yes | 19% |
| High \_Time\_Hard = Time take for fixing “Hard” violations | Yes | 8 hours |
| High \_%Very\_Hard = % of violations which are “Very\_Hard” | Yes | 1% |
| High \_Time\_Very\_Hard = Time take for fixing “Very\_Hard” violations | Yes | 24 hours |
| Cost per hour of developer time | Blended rate of different people who may work on a violation (architect, lead, developer,QA resource etc.) | Yes | $75/hr |