|  |
| --- |
| sAR: Security Assessment Report |
| 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. |

|  |
| --- |
| UNCLASSIFIED |

Table of Contents

[1. Executive Summary 3](#_Toc345608359)

[1.1. Application Characteristics 3](#_Toc345608360)

[1.2. Summary of Quality Indicators 3](#_Toc345608361)

[1.3. Assessment Highlights 3](#_Toc345608362)

[2. Measures of Security 4](#_Toc345608363)

[2.1. Importance of measuring all layers of an application 4](#_Toc345608364)

[2.1.1. Security Weaknesses. 5](#_Toc345608365)

[2.1.1.1. OWASP 2010 - A1 – Injection Flaws 5](#_Toc345608366)

[2.1.1.2. OWASP 2010 - A2 – Cross Site Scripting (XSS) 8](#_Toc345608367)

[2.1.2. Lack of Defensive Mechanisms. 9](#_Toc345608368)

[2.1.2.1. OWASP 2010 - A8 – Insecure Cryptographic Storage 9](#_Toc345608369)

[2.1.2.2. OWASP 2007 - A6 – Information Leakage and Improper Error Handling 9](#_Toc345608370)

[2.1.2.3. Other OWASP vulnerabilities: 11](#_Toc345608371)

[2.1.3. Bypassing the Architecture. 12](#_Toc345608372)

[2.1.4. Failure to Control Processing Volumes. 13](#_Toc345608373)

[2.1.5. Application Resource Imbalances. 14](#_Toc345608374)

[3. Appendix - Assessment Approach Overview 15](#_Toc345608375)

[4. Appendix: Understanding Quality Indicators, Quality Rules 16](#_Toc345608376)

[5. Appendix: CWE/SANS Top 25 18](#_Toc345608377)

# 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. 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 % |

## 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 |

# 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 |
| Architecture - Multi-Layers and Data Access | 2.2 | 0.00 % |
| Architecture - Object-level Dependencies | 1.4 | 0.00 % |
| Architecture - OS and Platform Independence | 3.23 | 0.00 % |
| Architecture – Reuse | 3.45 | 0.00 % |
| Complexity - Algorithmic and Control Structure Complexity | 2.2 | 0.00 % |
| Complexity - Dynamic Instantiation | 1.4 | 0.00 % |
| Complexity - OO Inheritance and Polymorphism | 3.23 | 0.00 % |
| Complexity - SQL Queries | 3.45 | 0.00 % |
| Dead code (static) | 2.3 | 0.00 % |
| Programming Practices - Error and Exception Handling | 3.5 | 0.00 % |
| Programming Practices - OO Inheritance and Polymorphism | 1 | 0.00 % |
| Programming Practices - Structuredness | 2.2 | 0.00 % |
| Programming Practices - Unexpected Behaviour | 1.4 | 0.00 % |
| Secure Coding - Time and State | 3.23 | 0.00 % |
| Volume - Number of Components | 1 | 0.00 % |

## 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.

### Security Weaknesses.

Applications are vulnerable to security attacks when they lack appropriate sanitization checks on user inputs in all relevant tiers of the application. Most secure applications contain a component dedicated to managing access rights and related security matters. As displayed in Figure 5, one of the easiest ways for an application’s security to be breached is when code in the user interface provides direct access to an application’s functionality prior to the requester’s access rights being authenticated as valid. This problem should be detected by analysing the input requests from secure functionality to determine whether they have passed through authentication mechanisms.



*Figure 5. Security Weaknesses*

### OWASP 2010 - A1 – Injection Flaws

“Injection flaws, particularly SQL injection, are common in web applications. There are many types of injections: SQL, LDAP, XPath, XSLT, HTML, XML, OS command injection and many more. Injection occurs when user-supplied data is sent to an interpreter as part of a command or query. Attackers trick the interpreter into executing unintended commands via supplying specially crafted data.”

|  |
| --- |
| SQL injection flaws |
| LDAP injection flaws |
| OS command injection flaws |
| XPath injection flaws |
| File path manipulation flaws |

The User Input Security Analyzer can be customized to specific need interactively to add specific sanitization, input or target methods for a given type of flaw. The Dataflow User Input Security Analyzer offer manager and developers the view of the precise path(s) where the vulnerability exists in a call path view directly in CAST AD Dashboard.

In addition to the dataflow searched quality rules, CAST AI Platform provides out of the box the following framework based rules also grouped into the technical criteria metric called “Secure Coding - Input Validation”:

|  |
| --- |
| ASP.NET: Always validate user input with Request variables |
| Struts Validator: Avoid action mappings validator turned off |
| Struts Validator: Avoid Form Field without Validator |
| Struts Validator: Avoid multiple validation form with the same name |
| Struts Validator: Avoid unused validation form |
| Struts Validator: Avoid Validator field without Form Field |
| Struts Validator: Enable Struts Validator plugin |
| Struts Validator: Form Bean must extend Validator Class |
| Struts Validator: Validator form validate() method must call super.validate() |

For application following the OWASP recommendation to implement object relational mapping (ORM) libraries such as Hibernate, CAST provides a specific rule that checks that only Hibernate is used to access the database:

|  |
| --- |
| Hibernate: Use only Hibernate API to access to the database |

CAST provides also 13 rules to check that Hibernate architecture & performance best practices are in place.

|  |  |
| --- | --- |
| Avoid file path manipulation vulnerabilities | |
| Description | Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed et accumsan felis. Etiam pharetra semper suscipit. Mauris hendrerit placerat lorem sit amet commodo. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam erat volutpat. |
| Remediation | Aliquam erat volutpat. Vestibulum quam ante, venenatis at bibendum vitae, viverra eget nulla. Donec pulvinar consequat varius. Morbi eget adipiscing lacus. Sed et libero odio, eget tempus massa. Phasellus venenatis commodo enim eget aliquet. Quisque posuere elit sed nunc aliquam eu ornare elit lacinia. Curabitur luctus, eros id venenatis lacinia, dolor libero tincidunt nibh, eget dapibus orci lectus pellentesque nisl. Ut quis velit est. |

|  |  |
| --- | --- |
| Avoid OS command injection vulnerabilities | |
| Description | Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed et accumsan felis. Etiam pharetra semper suscipit. Mauris hendrerit placerat lorem sit amet commodo. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam erat volutpat. |
| Remediation | Aliquam erat volutpat. Vestibulum quam ante, venenatis at bibendum vitae, viverra eget nulla. Donec pulvinar consequat varius. Morbi eget adipiscing lacus. Sed et libero odio, eget tempus massa. Phasellus venenatis commodo enim eget aliquet. Quisque posuere elit sed nunc aliquam eu ornare elit lacinia. Curabitur luctus, eros id venenatis lacinia, dolor libero tincidunt nibh, eget dapibus orci lectus pellentesque nisl. Ut quis velit est. |

|  |  |
| --- | --- |
| Avoid SQL injection vulnerabilities | |
| Description | Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed et accumsan felis. Etiam pharetra semper suscipit. Mauris hendrerit placerat lorem sit amet commodo. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam erat volutpat. |
| Remediation | Aliquam erat volutpat. Vestibulum quam ante, venenatis at bibendum vitae, viverra eget nulla. Donec pulvinar consequat varius. Morbi eget adipiscing lacus. Sed et libero odio, eget tempus massa. Phasellus venenatis commodo enim eget aliquet. Quisque posuere elit sed nunc aliquam eu ornare elit lacinia. Curabitur luctus, eros id venenatis lacinia, dolor libero tincidunt nibh, eget dapibus orci lectus pellentesque nisl. Ut quis velit est. |

|  |  |
| --- | --- |
| Avoid LDAP injection vulnerabilities | |
| Description | Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed et accumsan felis. Etiam pharetra semper suscipit. Mauris hendrerit placerat lorem sit amet commodo. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam erat volutpat. |
| Remediation | Aliquam erat volutpat. Vestibulum quam ante, venenatis at bibendum vitae, viverra eget nulla. Donec pulvinar consequat varius. Morbi eget adipiscing lacus. Sed et libero odio, eget tempus massa. Phasellus venenatis commodo enim eget aliquet. Quisque posuere elit sed nunc aliquam eu ornare elit lacinia. Curabitur luctus, eros id venenatis lacinia, dolor libero tincidunt nibh, eget dapibus orci lectus pellentesque nisl. Ut quis velit est. |

|  |  |
| --- | --- |
| Avoid XPath injection vulnerabilities | |
| Description | Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed et accumsan felis. Etiam pharetra semper suscipit. Mauris hendrerit placerat lorem sit amet commodo. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam erat volutpat. |
| Remediation | Aliquam erat volutpat. Vestibulum quam ante, venenatis at bibendum vitae, viverra eget nulla. Donec pulvinar consequat varius. Morbi eget adipiscing lacus. Sed et libero odio, eget tempus massa. Phasellus venenatis commodo enim eget aliquet. Quisque posuere elit sed nunc aliquam eu ornare elit lacinia. Curabitur luctus, eros id venenatis lacinia, dolor libero tincidunt nibh, eget dapibus orci lectus pellentesque nisl. Ut quis velit est. |

### OWASP 2010 - A2 – Cross Site Scripting (XSS)

“Cross-site scripting, better known as XSS, is in fact a subset of HTML injection. XSS is the most prevalent and pernicious web application security issue. XSS flaws occur whenever an application takes data that originated from a user and sends it to a web browser without first validating or encoding that content.”

Being a special form of injection, a part of the solution to get rid of the XSS threat comes from checking the protection in place for A2 Injection Flaws (see A2). Indeed, beside A2 protection (validation of all incoming data), the OWASP Top Ten document recommends the “appropriate encoding of all output data”.

CAST supports checking that using the same technology as for Injection Flaws: dataflow and tainted variable analysis to track input data thru out the source code in order find a path where the input data is sent back to the user without prior sanitization / encoding.

The User Input Security Analyser offers directly supports Cross-Site Scripting flaws that can be found in the technical criteria metric called “Secure Coding - Input Validation”.

As for A1-Injection flaws, the user can customize interactively the analysis to meet specific needs: adding specific sanitization, input or target methods for a given type of flaw is done directly thru the GUI.

XSS results are displayed in the dashboard which offers the view of precise path(s) where the vulnerability exists in a call path view directly in CAST AD Dashboard.

|  |  |
| --- | --- |
| Avoid cross-site scripting vulnerabilities | |
| Description | Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed et accumsan felis. Etiam pharetra semper suscipit. Mauris hendrerit placerat lorem sit amet commodo. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam erat volutpat. |
| Remediation | Aliquam erat volutpat. Vestibulum quam ante, venenatis at bibendum vitae, viverra eget nulla. Donec pulvinar consequat varius. Morbi eget adipiscing lacus. Sed et libero odio, eget tempus massa. Phasellus venenatis commodo enim eget aliquet. Quisque posuere elit sed nunc aliquam eu ornare elit lacinia. Curabitur luctus, eros id venenatis lacinia, dolor libero tincidunt nibh, eget dapibus orci lectus pellentesque nisl. Ut quis velit est. |

### 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.

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

* Application Architecture Practices
  + Multi-layer design compliance
  + Security best practices (Input Validation, SQL Injection, Cross-Site Scripting, etc. See CERT’s Top 25 <http://www.sans.org/top25-programming-errors/> )
* Programming Practices (code level)
  + Error & Exception handling
  + Security best practices (system functions access, access control to programs)

### OWASP 2010 - A8 – Insecure Cryptographic Storage

“Protecting sensitive data with cryptography has become a key part of most web applications. Simply failing to encrypt sensitive data is very widespread. Applications that do encrypt frequently contain poorly designed cryptography, either using inappropriate ciphers or making serious mistakes using strong ciphers. These flaws can lead to disclosure of sensitive data and compliance violations.”

CAST supports checking that the OWASP recommended protection is in place thru the implementation of custom quality rules. OWASP recommends checking that the application:

”Do not use weak algorithms, such as MD5 / SHA1. Favour safer alternatives, such as SHA-256 or better”.

Custom quality rule checks for A8 will check that these wrong API are not used.

### OWASP 2007 - A6 – Information Leakage and Improper Error Handling

“Applications can unintentionally leak information about their configuration, internal workings, or violate privacy through a variety of application problems. Applications can also leak internal state via how long they take to process certain operations or via different responses to differing inputs, such as displaying the same error text with different error numbers. Web applications will often leak information about their internal state through detailed or debug error messages. Often, this information can be leveraged to launch or even automate more powerful attacks.”

CAST supports the protection against “A6 – Information Leakage and Improper Error Handling” thru the use of the following rules integrated in the technical criteria metric called “Programming Practices - Error and Exception Handling”:

|  |
| --- |
| .NET: Avoid catching an exception of type Exception |
| .NET: Avoid empty catch blocks |
| .NET: Avoid empty finally blocks |
| .NET: Avoid throwing an exception of type Exception |
| ABAP: Avoid missing WHEN OTHERS in CASE statements |
| ABAP: Avoid unchecked return code (SY-SUBRC) after OPEN SQL or READ statement |
| ABAP: Avoid using AT events in combination of LOOP AT .... WHERE constructs |
| ASP: Use of error handling page |
| COBOL: Avoid DISPLAY ... UPON CONSOLE |
| COBOL: Avoid using HANDLE ABEND |
| COBOL: Avoid using HANDLE CONDITION |
| COBOL: Avoid using IGNORE CONDITION |
| COBOL: Include a WHEN OTHER clause when using EVALUATE |
| COBOL: Programs accessing relational databases must include the SQLCA copybook |
| Java: Avoid catch blocks with assertion |
| Java: Avoid catching an exception of type Exception, RuntimeException, or Throwable |
| Java: Avoid declaring throwing an exception and not throwing it |
| Java: Avoid direct Class inheritance from java.lang.Throwable |
| Java: Avoid empty catch blocks |
| Java: Avoid empty finally blocks |
| Java: Avoid missing default in switch statements |
| Java: Avoid return statements in finally blocks |
| Java: Avoid throwing an exception in a catch block without chaining it |
| Java: Avoid throwing an exception of type Exception |
| Java: Avoid using 'java.lang.Error' |
| Java: Avoid using 'java.System.exit()' |
| Java: Avoid using 'System.err' and 'System.out' outside a try catch block |
| Java: Avoid using 'System.err' and 'System.out' within a try catch block |
| Java: Avoid using 'System.printStackTrace()' outside a try catch block |
| Java: Avoid using 'System.printStackTrace()' within a try catch block |
| JSP: Pages should use error handling page |
| PL-SQL: Use WHEN OTHERS in exception management |
| T-SQL: Avoid Functions and Procedures doing an Insert, Update or Delete without including error management |
| T-SQL: Avoid Stored Procedures not returning a status value |
| VB: Avoid using "On error Resume Next" in the Class event terminate. |
| VB: Use a single Error Handling Method |

### Other OWASP vulnerabilities:

For A4, A5, A7, A10 of OWASP Top Ten 2010, the OWASP document itself describes these vulnerabilities as difficult or impossible to check using source code analysis.

|  |  |
| --- | --- |
| Avoid Log forging vulnerabilities | |
| Description | Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed et accumsan felis. Etiam pharetra semper suscipit. Mauris hendrerit placerat lorem sit amet commodo. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam erat volutpat. |
| Remediation | Aliquam erat volutpat. Vestibulum quam ante, venenatis at bibendum vitae, viverra eget nulla. Donec pulvinar consequat varius. Morbi eget adipiscing lacus. Sed et libero odio, eget tempus massa. Phasellus venenatis commodo enim eget aliquet. Quisque posuere elit sed nunc aliquam eu ornare elit lacinia. Curabitur luctus, eros id venenatis lacinia, dolor libero tincidunt nibh, eget dapibus orci lectus pellentesque nisl. Ut quis velit est. |

### 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.



*Figure 3. Example of the Failure to bypass the architecture*

**Architecture –Multi Layers and Data Access**

|  |  |  |
| --- | --- | --- |
| Rule | **Desc.** | **# Violations** |
| Rule1 | Desc1 | 1 |
| Rule2 | Desc2 | 2 |
| Rule3 | Desc3 | 3 |

### 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.



*Figure 3. Example of the Failure to Set Processing Limits*

|  |  |  |
| --- | --- | --- |
| Rule | **Desc.** | **# Violations** |
| Rule1 | Desc1 | 1 |
| Rule2 | Desc2 | 2 |
| Rule3 | Desc3 | 3 |

### 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.



*Figure 4. Imbalance between Application Resources*

# 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: CWE/SANS Top 25

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2011 CWE/SANS Top 25 Most Dangerous Software Errors** | | | | | | | | | | | | | | |
| **Insecure Interaction Between Components** | | | | | | | | | | | | | | |
| These weaknesses are related to insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems. | | | | | | | | | | | | | | |
| **Rank** | **CWE ID** | | **Name** | | | | **Recommendation/Mitigation/Comments** | | | | **Coverage with CAST** | | | **Implementation areas** |
| [1] | CWE-89 | | Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') | | | | Checking for SQL Injection | | | | Included in the product | | | User Input Security (UIS) |
| [2] | CWE-78 | | Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') | | | | Checking for OS Command Injection | | | | Included in the product | | | UIS |
| [4] | CWE-79 | | Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') | | | | Checking for Cross-site scripting | | | | Included in the product | | | UIS |
| [9] | CWE-434 | | Unrestricted Upload of File with Dangerous Type | | | | Input Validation | | | | A custom rule would have to be developed by configuring file as a target of a user input and ask the user input to be sanitized. | | | UIS |
| [12] | CWE-352 | | Cross-Site Request Forgery (CSRF) | | | | Ensure that application is free of cross-site scripting issues (CWE-79), because most CSRF defenses can be bypassed using attacker-controlled script. | | | | CWE-79 can address this requirement | | | UIS |
| [22] | CWE-601 | | URL Redirection to Untrusted Site ('Open Redirect') | | | | Checking for Cross-site scripting | | | | Included in the product | | | UIS |
|  |  | |  | | | |  | | | |  | | |  |
| **Risky Resource Management** | | | | | | | | | | | | | | |
| The weaknesses in this category are related to ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources. | | | | | | | | | | | | | | |
| [3] | | CWE-120 | | Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') | | Perform input validation on any numeric input by ensuring that it is within the expected range. Enforce that the input meets both the minimum and maximum requirements for the expected range. | | | Check for valid constructors where buffer size is checked. Add rules to check for libraries which provide safer versions of overflow-prone string-handling functions. | | | | User Input Security (UIS) | |
| [13] | | CWE-22 | | Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') | | Checking for file path manipulation | | | Included in the product | | | | UIS | |
| [14] | | CWE-494 | | Download of Code Without Integrity Check | | Check Download Code Integrity | | | A custom rule could be generated with Architecture Checker to make sure libraries which avoid such downloads are used. | | | | UIS | |
| [16] | | CWE-829 | | Inclusion of Functionality from Untrusted Control Sphere | | When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs. | | | This one would require an evolution in the product to check for trusted/acceptable objects | | | | Analyzers | |
| [18] | | CWE-676 | | Use of Potentially Dangerous Function | | Checking for programming best practices | | | Included in the product | | | | Analyzers | |
| [20] | | CWE-131 | | Incorrect Calculation of Buffer Size | | Perform input validation on any numeric input by ensuring that it is within the expected range. Enforce that the input meets both the minimum and maximum requirements for the expected range. | | | Custom rules could be added to check variables size discrepancies, precision, signed/unsigned distinctions, truncation, conversion and casting between types, "not-a-number" calculations | | | | Analyzers | |
| [23] | | CWE-134 | | Uncontrolled Format String | | Whenever possible, use functions that do not support the %n operator in format strings. | | | Custom rules could be added to check that a static string is passed to the string format function, which cannot be controlled by the user and that the proper number of arguments are always sent to that function | | | | Analyzers | |
| [24] | | CWE-190 | | Integer Overflow or Wraparound | | Perform input validation on any numeric input by ensuring that it is within the expected range. Enforce that the input meets both the minimum and maximum requirements for the expected range. | | | This one would require an evolution in the product to be able to perform sanity checks for integer overflows | | | | UIS | |
|  | |  | |  | |  | | |  | | | |  | |
| **Porous Defenses** | | | | | | | | | | | | | | |
| The weaknesses in this category are related to defensive techniques that are often misused, abused, or just plain ignored. | | | | | | | | | | | | | | |
| [5] | | CWE-306 | | Missing Authentication for Critical Function | | avoid implementing custom authentication routines and consider using authentication capabilities as provided by the surrounding framework, operating system, or environment. | | | A custom rule could be added with Architecture Checker to make sure the surrounding authentication capabilities are used | | | | Analyzers | |
| [6] | | CWE-862 | | Missing Authorization | | Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page. | | | A custom rule could be added to check custom methods for authorization | | | | Analyzers | |
| [7] | | CWE-798 | | Use of Hard-coded Credentials | | Store passwords, keys, and other credentials outside of the code in a strongly-protected, encrypted configuration file or database that is protected from access by all outsiders, including other local users on the same system. | | | An evolution of the product is required to check parameters/arguments to the connection methods | | | | User Input Security (UIS) | |
| [8] | | CWE-311 | | Missing Encryption of Sensitive Data | | Periodically ensure that you aren't using obsolete cryptography. Avoid using old encryption techniques using MD4, MD5, SHA1, DES, and other algorithms that were once regarded as strong. | | | A custom rule could be develop to identify and check the use of possible sanitization methods | | | | UIS | |
| [10] | | CWE-807 | | Reliance on Untrusted Inputs in a Security Decision | | consider getcookies as unsafe | | | Included in the product | | | | UIS | |
| [11] | | CWE-250 | | Execution with Unnecessary Privileges | | Checking for privileges being appropriately implemented based on the scenario/usecase. Perform extensive input validation and canonicalization to minimize the chances of introducing a separate vulnerability. | | | By using dataflow mechanism identify the different permissions that the software and its users will need to perform their actions, such as file read and write permissions, network socket permissions, and so forth. Then raise voilations, if there are extra rights left unattended. | | | | Analyzers | |
| [15] | | CWE-863 | | Incorrect Authorization | | consider getcookies as unsafe | | | Included in the product | | | | UIS | |
| [17] | | CWE-732 | | Incorrect Permission Assignment for Critical Resource | | Path manipulation | | | Included in the product | | | | UIS | |
| [19] | | CWE-327 | | Use of a Broken or Risky Cryptographic Algorithm | | Validating encryption algorithms | | | by checking use of AES and avoid old algorithms such as DES. This should be implemented in the J2EE/C/C++/.NET analyzers | | | | Analyzers | |
| [21] | | CWE-307 | | Improper Restriction of Excessive Authentication Attempts | | Check login implementation | | | by checking the login API/Functions to ensure appropriate login failure handling mechanism. | | | | Analyzers | |
| [25] | | CWE-759 | | Use of a One-Way Hash without a Salt | | Checking for programming best practices | | | Included in the product | | | | Analyzers | |
|  | |  | |  | |  | | |  | | | |  | |
| **CAST detects following security vulnerabilities identified by OWASP & CWE (not part of top-25):** | | | | | | | | | | | | | | |
| CWE-20: | | | | | Improper Input Validation | | | Checking for best programming practices | | Included in the product | | User Input Security (UIS) | | |
| CWE-116: | | | | | Improper Encoding or Escaping of Output | | | Checking for best programming practices | | Included in the product | | UIS | | |
| CWE-90 | | | | | LDAP Injection | | | Checking for LDAP injection | | Included in the product | | UIS | | |
| CWE-91 | | | | | XPATH Injection | | | Checking for XPATH injection | | Included in the product | | UIS | | |
| CWE-73: | | | | | External Control of File Name or Path | | | Checking for file path manipulation | | Included in the product | | UIS | | |
| CWE-99: | | | | | Improper Control of Resource Identifiers ('Resource Injection') | | | Checking for best programming practices | | Included in the product | | UIS | | |
| CWE-117: | | | | | Improper Output Neutralization for Logs | | | Checking for log forging | | Included in the product | | UIS | | |