CISQ Conformance Assessment Method:

Quality Characteristic Measures

A Guide for Assessing the Level of Conformance of a Static Analysis Product in Detecting the Rule Violations Comprising the CISQ Quality Characteristic Measures

CISQ–TR–2017–01

CONSORTIUM FOR IT SOFTWARE QUALITY

Executive Summary

This document presents the method for assessing a technology's conformance to the four CISQ Quality Characteristic Measures, which have been approved as OMG standards. These measures assess the structural quality of an IT application's source code, a primary source of risk affecting an application's operational performance and cost of ownership. Each measure is composed of a list of severe violations of good architectural and coding practice that can be detected using static analysis technology. Each measure is developed by counting the occurrences of its violations in the source code. These results can be used to certify the level of quality in a specific release of the application. The evidence required to demonstrate a technology's ability to detect and measure the structural quality violations is listed for each CISQ measure (Reliability, Security, Performance Efficiency, and Maintainability). Forms are provided for capturing the evidence and level of compliance, represented as the extent to which a technology detects and measures all the CISQ violations and the various forms in which they occur in source code.

Table of Contents

1. Structural Quality Certification

1.1 What Is a Structural Quality Certification?

Structural quality certification provides IT managers and executives with a quantitative indicator of the risk and cost associated with an IT application. Therefore, a set of certifications covering an organization's critical business applications provides evidence that the risks and costs of these applications are under IT governance. Each certification is based on the static analysis of application's source code to detect violations of good architectural and coding practice. The results are presented as measures for each of four quality characteristics—Reliability, Security, Performance Efficiency, and Maintainability. The analyses and their measures can be used as:

- an indicator of the business risk or maintenance cost associated with an application,
- a guide for remediating weaknesses that will reduce the risk of cost of an application,
- a basis for tracking trends in the risk or cost of an application over time, and
- a sign that the risks and costs of applications are being governed and managed.

CISQ does not certify applications. Rather, the level of quality in an application can be certified by a CISQ conformant service provider using a CISQ conformant technology to analyze and measure its various quality characteristics. CISQ conformant service providers will present customers with a certificate that expresses an application's quality level on one or more CISQ measures in sigma form (weaknesses per million opportunities, to be described later).

The CISQ certification is an indicator rather than an absolute measure of application quality for two reasons.

1. The CISQ measures do not include all weaknesses related to a specific quality characteristic, only those that were deemed severe enough to require remediation. However, a measure constructed on these weaknesses should be a strong correlate of the amount of less severe weaknesses that were not measured.
2. A CISQ quality measure is only stable until the next patch is implemented on the code. Consequently, the injection of new weaknesses through enhancements, modifications, or deletions will change the certification measurements. However, since the results are reported as sigma levels the actual quality levels may not change dramatically after standard maintenance activities, especially if good quality assurance activities are applied.

Consequently, a CISQ certification does not guarantee a level of defect-freeness or a level of operational performance. However, it correlates highly with these outcomes and provides management with insight into the risks and costs associated with an application. It also provides evidence that management is governing these risks and costs when unacceptable certification results lead to remedial actions.

1.2 What Are the CISQ Structural Quality Measures?

The CISQ Quality Characteristic measures were selected by executives from the companies who formed CISQ. These 4 measures were selected from among 10 candidates as being the most relevant to their application challenges in 2010. Teams of experts from the original 254 companies that formed CISQ met

over a 2-year period to select the weaknesses that would constitute each measure. For instance, the Security measure includes such weaknesses as SQL injection, cross-site scripting, and buffer overflows.

A weakness was only included in a CISQ measure if a majority of IT professionals would agree it had to be remediated. However, the Security was developed from the top 25 CWEs (of which 22 can be detected through static analysis). The four CISQ Quality Characteristic measures include:

- **Reliability**—a measure of the extent to which software contains weaknesses that cause outages, unexpected behavior, instability, data corruption, long recovery times, or other related problems.
- **Security**—a measure of the extent to which software contains weaknesses that can be exploited to gain unauthorized access to a system to steal data, cause damage, or other malicious acts.
- **Performance Efficiency**-- a measure of the extent to which software contains weaknesses that can degrade a system's performance or cause excessive use of processor, memory, or other resources.
- **Maintainability--** a measure of the extent to which software contains weaknesses that make software hard to understand or change, resulting in excessive maintenance time and cost as well as higher defect injection rates.

1.3 Why Are the CISQ Quality Measures Valuable for Certifications

The CISQ Quality measures are an international standard supported by OMG. They are the only standard that assesses quality at the level of weaknesses the source code at both the architectural system level and the level of modules or code units. Consequently, the CISQ measures provide that following benefits for certifications.

1. CISQ Quality Measures have an established and well-defined definition, therefore they provide a common language for customers, suppliers, auditors, contract officials, and others for communicating about application quality issues and expectations.
2. CISQ Quality Measures provide a common basis for benchmarking quality across systems, technologies, vendors, and companies. They provide a rigorous basis for developing baselines and thresholds that are specific to technology, industry vertical, application type and other demographic characteristics.
3. CISQ Quality Measures are directly related to the weaknesses in software that create risk and cost. Therefore, they provide leading indicators of potential operating problems or excessive maintenance costs.

1.4 How Do CISQ Measures Relate to ISO Standards?

The definition and coverage of CISQ Quality Characteristic measures are conformant with the definitions of software quality characteristics and sub-characteristics in ISO/IEC 25010, which replaces the old ISO/IEC 9126. ISO/IEC 25010 defines a quality characteristic as being composed from several quality sub-characteristics. This framework for software product quality is presented in Figure 1 for the eight quality characteristics presented in 25010. The quality characteristics and their sub-characteristics selected for source code measurement by CISQ are indicated in blue.

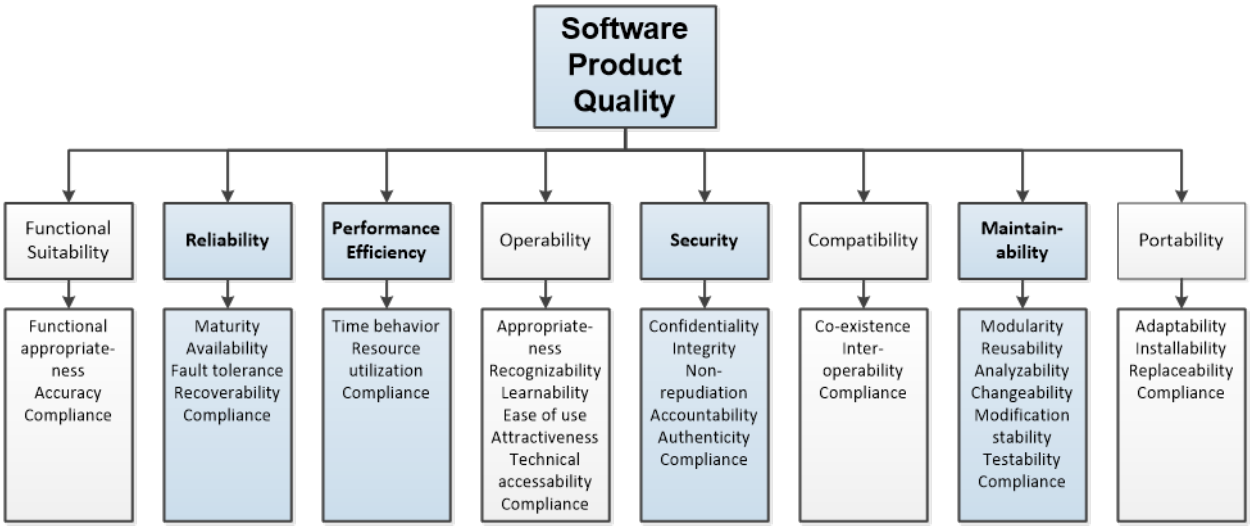

Figure 1. Software Quality Characteristics from ISO/IEC 25010 with CISQ focal areas highlighted

The definitions of actual software quality measures are provided in ISO/IEC 25023 where each quality characteristic is quantified by a collection of measurable attributes of software, such as the violation of a quality rule. However, the measures provided in ISO/IEC 25023 are generally defined at the level of system behavior and do not measure the actual source code, that is they are not based on measuring the software weaknesses that cause unacceptable system behaviors. The CISQ measures supplement ISO/IEC 25023 by providing measures of quality-related weaknesses in the source code.

1.5 How Are Certification Scores Presented to Customers?

A CISQ certification is presented in the Sigma format that many companies are familiar with from Six Sigma quality improvement programs. The use of Sigma levels provides a common representation supported by a rigorous, statistically-based method for benchmarking quality results. A Sigma (σ) is a standard deviation from the mean score of a distribution. In quality programs Sigma results are interpreted as frequencies of occurrence based on the Normal Distribution. For instance, a score of 6σ is six standard deviations from the mean, and indicates no more than 4 defects per million points of inspection. A point of inspection is interpreted as opportunity for a defect to have occurred and been detected. In measuring quality, the Sigma level is determined by the number of weaknesses per million opportunities. The fewer weaknesses per million opportunities, the higher the Sigma score.

In software, opportunities are measured by the number of times an architectural or coding rule could be applied to a construct in the source code. Thus the CISQ Certification score on one of the four measures applied to an application is the number of occurrences of each weakness included in the CISQ measure compared to the number of times the CISQ rule violated by the weakness could be applied to constructs in the source code. For instance, if there were 2000 modules in an application then there would 2000 opportunities to violate a rule regarding excessive coupling. The number of weaknesses per million opportunities declines exponentially as the Sigma level increases.

The total number of occurrences detected for the weaknesses included in a CISQ measure is then transformed into weaknesses per million opportunities to determine the Sigma level for that measure. The thresholds for each Sigma level are as follows.

- 1σ — 697,672 weaknesses per million opportunities — 69.7672 % defective
- 2σ — 308,537 weaknesses per million opportunities — 30.8537 % defective
- 3σ — 66,807 weaknesses per million opportunities — 6.6807 % defective
- 4σ — 6,210 weaknesses per million opportunities — .6210 % defective
- 5σ — 233 weaknesses per million opportunities — .0233 % defective
- 6σ — 3.4 weaknesses per million opportunities — .0003% defective

In general applications below the 3 Sigma level should be considered unacceptable and of high risk. In practice, it will be difficult for applications to achieve 5 or 6 Sigma scores, and this level of quality may be beyond the requirements for many applications, or beyond the cost-benefits of striving for this level of quality. However, there are violations such as SQL injection for which the tolerance level should be '0 occurrences' since the Security risks posed by this weakness can be disastrous. The appropriate quality range for most business applications will be a certification between 3 and 4 Sigma. Figure 2 presents an example of certificate reporting Sigma levels for an application that would be presented to customers.

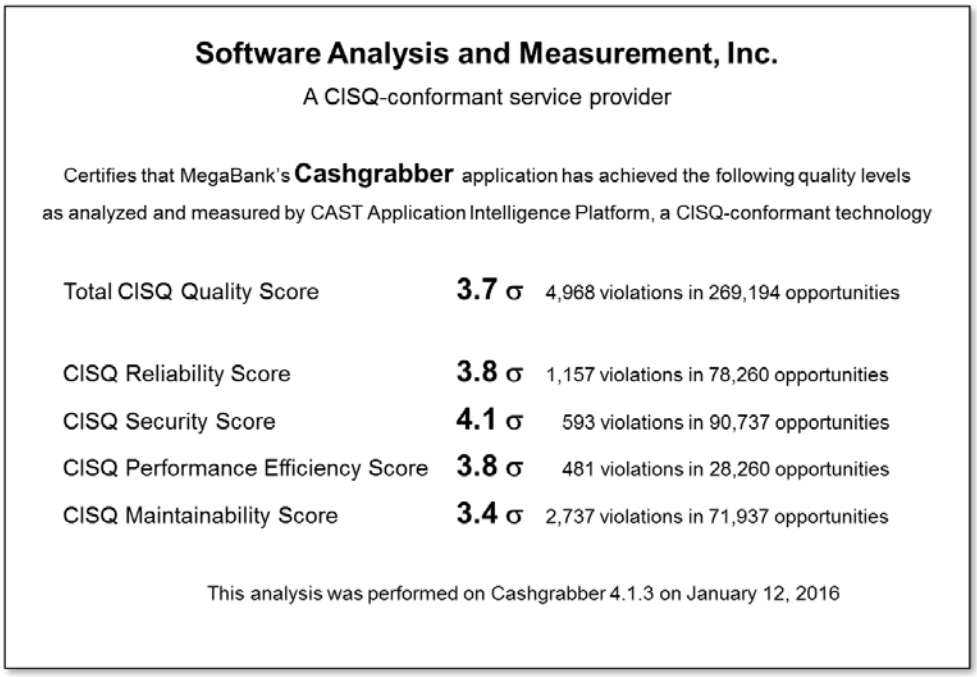

Figure 2. Example Certification Report

Other quality measures can be added to the certification such as the density of weaknesses based on the size of the application. In fact, density measures may be more appropriate than Sigma notations for small applications (<10,000 lines of source code) because of the constrained number of opportunities where architectural and coding rules can be applied.

2. The CISQ Conformance Assessment for Technology

2.1 What Is the CISQ Conformance Assessment Process?

An endorsement for CISQ conformance is awarded by a CISQ-authorized Lead Appraiser to a vendor's source code analysis and measurement technology. The Appraiser will evaluate the vendor's technology either at the vendor's site, or under agreement at another site, to assess the evidence of CISQ-conformance. The CISQ Conformance Assessment is conducted under a non-disclosure agreement with the vendor. The vendor may seek a conformance endorsement for all four CISQ Quality Measures or a subset of the measures.

Detection patterns can differ among programming languages and related technologies. Therefore, a conformance assessment must be performed separately for each programming language or related technology the vendor's technology supports that would be included in a CISQ-based certification. Vendors shall clearly indicate the programming languages and related technologies for which their technology has been endorsed as CISQ-conformant.

There are five steps in the CISQ Conformance Assessment process.

1. The Appraiser will review an evidence-based claim that they detect the weaknesses in a CISQ measure as demonstrated by the output from their analyses of customer applications that their technology can detect the weaknesses comprising the CISQ measures.
2. The Appraiser will review descriptions of how the technology detects and counts weaknesses. The vendor should not reveal any trade secrets in describing their analysis procedures. Nevertheless, the vendor should describe analytic procedures in sufficient detail that the Appraiser can determine whether they are effective approaches to detecting the weakness. If necessary, and both parties agree, the Appraiser can evaluate analytic techniques in the source code of the technology under evaluation.
3. The Appraiser will interview several members of the vendor's staff to affirm the information in the conformance case. Typical interviewees might include system designers, software developers, technology installers or operators, solution consultants, and a technology manager.
4. The Appraiser should observe a customer analysis to determine verify information in the case and to determine any limits on the analysis technology such as application size or multiple languages. If a customer analysis is not available, the Appraiser should observe an analysis on a vendor testbed. The applications analyzed should be of sufficient size to demonstrate the capability of the technology and validity of the conformance case.
5. The Appraiser will prepare a report of findings regarding conformance regarding the detection of each weakness for each measure in the scope of the assessment. This report will be submitted both to the vendor and the CISQ office. Optionally the Appraiser can help the vendor revise their conformance case for presentation to customers based on the evidence and limitations observed. The conformance assessment will normally take between one and three days.

The Appraiser will develop an agreement with the vendor on the scope and process steps of the CISQ Conformance Assessment. This agreement will be documented in an assessment plan which can be incorporated into an assessment contract or other formal agreement. At a minimum, the plan will include the following information.

- The CISQ Measures to be assessed for conformance

- The dates and estimated durations for each of the five tasks in the assessment process
- The analysis reports to be evaluated and the analyses to be observed
- The number and roles of employees to be interviewed
- The data for submitting a final conformance report

2.2 What Is Required to Demonstrate Conformance?

Vendors of technology wanting to be designated as 'CISQ Conformant' must present an evidence-based case to support their claim of conformance. The conformance claim and supporting arguments backed by evidence that justifies the validity of the claim will be evaluated by a CISQ-authorized assessor to determine if the case is sufficient to support the conformance claim. The assessor will also want to see the technology used in an analysis. The vendor must be able to demonstrate the following four capabilities.

1. *Capability to Load and Prepare Software for Analysis*—The vendor shall maintain an auditable description of the process for preparing the software for the analysis. The description shall include a list of all information and artifacts required to configure the software and support its analysis. The process description shall describe the tasks for obtaining the software and related information, ensuring the protection of customer confidential information and intellectual property, loading the software, and configuring the software for analysis.
2. *Capability to Analyze and Detect CISQ Measure Weaknesses*—The vendor shall maintain an auditable description of how its technology analyzes each of the violations in each CISQ Quality Characteristic measure for which conformance is being assessed. This description shall include methods for detecting violations both within and across code units, modules, and components where violations can involve multiple program elements.
3. *Capability to Accurately Compute CISQ Quality Characteristic Measures*—The vendor shall maintain an auditable description of how detected violations are counted and aggregated into a CISQ-conformant score for each CISQ Measure which is being assessed for conformance. If other measures such weakness density are being reported as part of the certification, the auditable description shall include them.
4. *Capability to Provide Auditable Reports of Results*—The vendor's analysis and measurement technology shall provide auditable reports that include the number and location of violations for each of the CISQ violations included in each CISQ Measure being assessed. These reports will include all measures computed from the detected weaknesses. These reports should be standard outputs that are presented to customers.

Claims of conformance will be made individually for each CISQ measure, and sub-claims of conformance will be made for each weakness included in the measure. In the case where a weakness may have several different instantiations each requiring a different mode of detection, a third tier of claims must be made regarding these different weakness modes.

Some vendor technologies may be unable to detect all the weaknesses comprising each CISQ measure at the initiation of the CISQ Conformance program. Consequently, CISQ has established a sliding scale of requirements for conformance during the period of 2017 to 2019. During these years the number of weaknesses that must be detected to achieve conformance will gradually increase as depicted in Table 1. By the end of 2019, vendor technologies must not fail to detect more than one of the weaknesses included in a measure in to be conformant with that measure.

CISQ Measures	# weaknesses for conformance			Total weaknesses per measure
	2017	2018	2019	
Reliability	20	24	28	29
Security	15	18	21	22
Perform. Eff.	12	13	14	15
Maintainability	15	17	19	20

Table 1. Yearly Requirement for Number of Weaknesses Detected to Achieve Conformance

The required percentages are more lenient for Reliability and Security, reflecting the greater analytic difficulty of detecting weaknesses in these measures. This graduated compliance scheme allows vendors sufficient time to develop the analytic and measurement capabilities required for conformance. In their evidence-based conformance case are required to indicate what weaknesses they are unable to detect, or any instantiations of a weakness they are unable to detect.

2.3 How Is an Evidence-Base Case Constructed?

Each claim is associated with an argument that the claim is true. Arguments are supported by evidence that demonstrate the validity of the claim. An argument states the kind of evidence required and why that demonstrates the truth of the claim. Thus, the evidence must support the argument and be sufficient to judge its truth. The hierarchy of conformance claims in presented in Figure 2.

Validated subclaims can be used to provide evidence for the validity of higher level claims. That is, the fact that the weaknesses comprising a measure can be detected and counted provides evidence that argues the technology is CISQ-conformant for that measure.

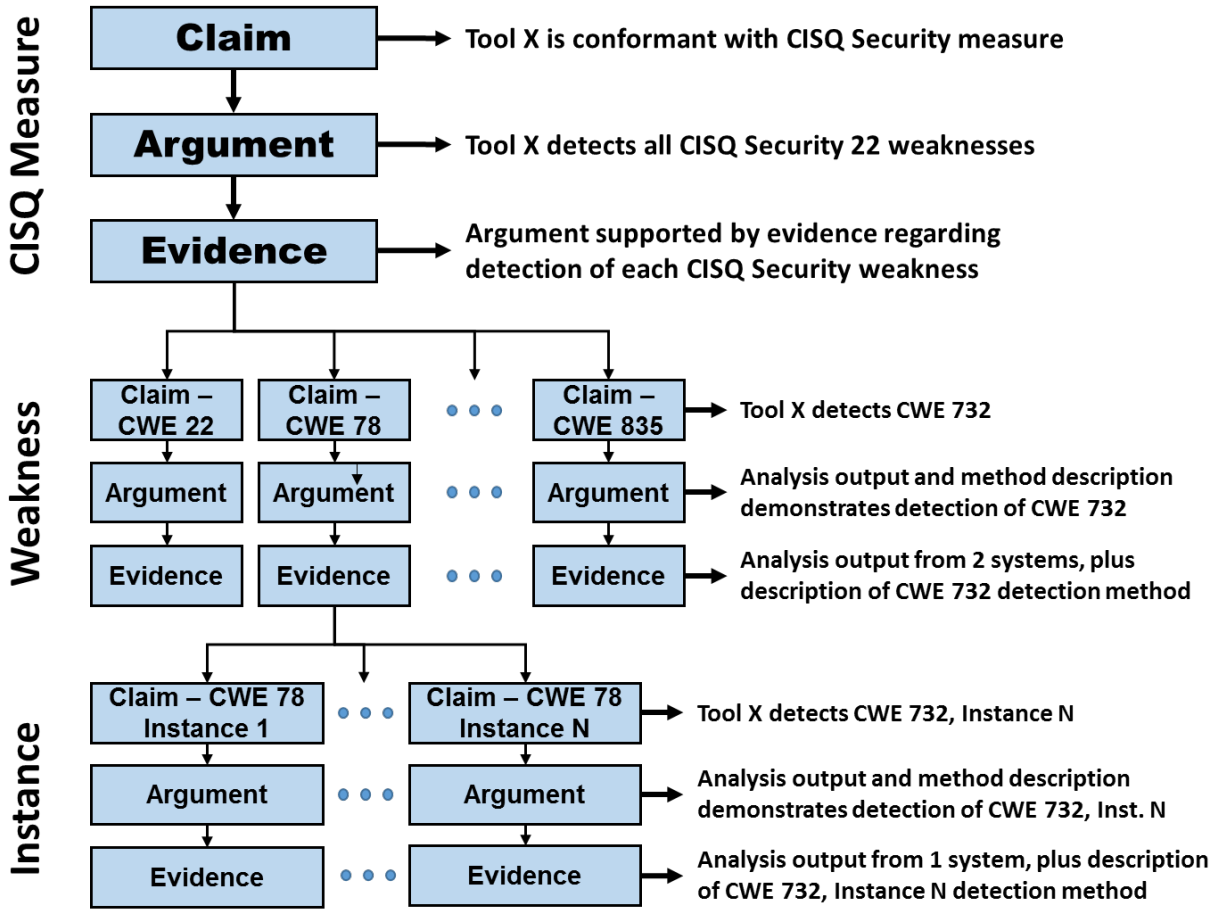

Figure 2. The Hierarchy of Claims for a CISQ Conformant Technology

The hierarchy of claims in a CISQ conformance case for technology compliance will include the first two and possibly the third level of claims as follows:

- *Measure Level*—a claim that the Technology achieves the level of detection required for conformance for a specific measure. The level of detection will be determined by verified claims regarding each of the weaknesses incorporated into a CISQ measure. During the initial three years of the CISQ Conformance Assessment program, the level of compliance may be different for each CISQ measure based on the difficulty of detecting the various weaknesses incorporated into it.
- *Weakness Level*—a claim that a weakness can be detected by the technology in one or more of its instantiations in code. The claim for a weakness may be verified for some instantiations, but not others. In which case the claim will be conditional and undetectable instantiations must be declared.
- *Instantiation Level*—a weakness may occur in several different forms, or instantiations, in the code. Each instantiation must be verified with evidence that supports the argument for that instantiation. Instantiations that cannot be accurately detected must be declared and represented in the evidence-based case presented to the CISQ Assessor. Undetected instantiations must also be declared in materials presented to customers so they know the limits of the certification assessments performed. If has only a single instantiation, then no third-tier claims need to be made for the weakness.

2.4 What Evidence Is Required?

Claims and arguments for CISQ conformance must be supported by several forms of evidence for each of the CISQ measures to which the vendor claims conformance. The primary types of evidence to be presented in a CISQ consist of the following.

- Output from one or more automated structural quality analyses
- Descriptions of how each CISQ weakness is detected by the technology
- Onsite interviews with developers and operators of the technology
- Observation of an actual structural quality analysis

Evidence of detection-—The most important form of evidence is output from the results of a structural quality analysis. This form of evidence is required and guidance for its content can be found in Section 3. The analysis output must provide the following information.

1. A list of CISQ weaknesses detected with their CISQ identifiers
2. The count of weaknesses detected for each type of CISQ weakness
3. The total of all CISQ weaknesses detected
4. The location in the code of each CISQ weakness detected

Evidence from detection method—Complementing evidence of detection from the output is a description of the analysis methods used to detect each of the CISQ weaknesses. Method descriptions are a requirement in the assessment process. The method used should be described in enough detail for a knowledgeable assessor to evaluate its reasonableness, but proprietary or trade secret information should not be revealed. A high level description of methods should also be included in the documented conformance case. This form of evidence should include the following information.

1. The method for loading and preparing the software for analysis
2. How the code pattern representing each potential weakness is identified
3. How the elements of each pattern relevant to the weakness are analyzed
4. In the case of system-level weaknesses, how patterns and elements are analyzed across applications layers
5. How the weakness is determined and distinguished from a false positive
6. In the case of weaknesses with multiple instantiations, how each instantiation is identified, analyzed, and distinguished from a false positive
7. A list of instantiations that cannot be detected

Evidence from interviews—Assertions from interviews with developers or operators can be used to verify the evidence collected from analysis outputs and method descriptions. Developers can be questioned about the methods used in identifying patterns and detecting weaknesses. Operators can be questioned about how application is prepared for analysis and the types of problems have been encountered during analysis. The purpose of interviewing the vendor's staff include the following.

- Verifying and affirming evidence from outputs and method descriptions
- Clearing up points of confusion
- Assisting the vendor in preparing their conformance case for presentation to customers
- Recommendations for improvement from the assessor's experience

In addition to the evidence regarding weaknesses enumerated above, the method for calculating the Sigma level and other measures provided in the certification must also be inspected. Evidence regarding the calculation of the Sigma level should include the following.

1. The method for counting weaknesses and aggregating scores
2. The transformation of the scores into weaknesses per million opportunities

3. The translation of weaknesses per million opportunities into Sigma notation based on a normal distribution
4. Methods for calculating any other measures provided in the certification such as weakness density.

The evidence should be organized in a conformance case that can be evaluated by a CISQ Assessor prior to an on-site visit. Forms for recording evidence are provided in Appendix A.

2.5 The Endorsement of CISQ Conformance

The CISQ-authorized Assessor will determine the level of conformance of the vendor's analysis and measurement technology. If the vendor's technology does not satisfy the required threshold of conformance for any of the CISQ measures within scope of the assessment, the Assessor will describe any necessary remediation actions to achieve the threshold required for endorsement as CISQ-conformant. For each CISQ Measure for which the threshold for conformance has been achieved, the Assessor will provide the vendor an Endorsement of CISQ Conformance. The vendor is then entitled to advertise conformance for those specific CISQ measures.

3. What Is Required for Each Structural Quality Characteristic

3.1 Assessment Guidance for Evaluating Weakness Detection

CISQ-authorized Assessors will evaluate the capability of a technology to detect and analyze patterns in software where rules of good architectural and coding practice can be violated. A technology must be able to identify the program elements incorporated into the pattern and detect the anomaly in their structure, organization, relationships, or interaction that creates the CISQ weakness. The following tables present the structures that should be detected for each weakness in each CISQ measure in order to argue the claim that a weakness can be detected. How these elements are detected would be provided in the evidence regarding the analysis method. The columns in these tables provide the following information.

- *Weakness identifier*—the descriptive name of the weakness
- *Detection Level*—indicated in yellow lettering directly below the weakness identifier, the detection level indicates program levels that may contain structural elements of the weakness, and which therefore must be analyzed to detect the weakness. A weakness can be limited to only one level, or it can incorporate all three. The three levels include
    - *Unit*—a self-contained group of computer instructions such as a class, method, module, sub-routine, etc., often separately compiled
    - *Technology*—a collection of code units that form a sub-system, layer, or some other significant grouping, and are written in the same language
    - *System*—the full collection of technology groupings across the various layers of an application that form its architecture and software content
- *Descriptor*—a short name for the weakness, chosen from common usage where possible.
- *Detection pattern*—a description of the weakness pattern that includes the source code elements that must be detected to identify the weakness during automated code analysis.

3.2 Reliability

Reliability measures of the extent to which software contains weaknesses that cause outages, unexpected behavior, instability, data corruption, long recovery times, or other related problems.

CISQ identifier	Descriptor	Remediation
ASCRM-CWE-120 *system, technology, unit*	Buffer overflow	Detect instances where the content of the first buffer is moved into the content of the second buffer while their allocated sizes are incompatible
ASCRM-CWE-252-data *unit*	Unchecked return parameter from data handling operations	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. executes a CRUD SQL statement, yet the return code value of the action is not checked anywhere
ASCRM-CWE-252-resource *unit*	Unchecked return parameter from resource handling operations	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. reads, writes, or manages an external resource, yet the return code value of the action is not checked anywhere
ASCRM-CWE-396 *unit*	Catch of overly broad exception types	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. contains a catch of an exception whose type is part of a list of overly broad exception types
ASCRM-CWE-397 *unit*	Throw of overly broad exception types	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. throws an exception whose type is part of a list of overly broad exception types
ASCRM-CWE-456 *unit*	Uninitialized data element	Detect instances where a variable, field, member, etc. is declared, then is evaluated without ever being initialized prior to the evaluation
ASCRM-CWE-674 *system, technology, unit*	Recursion	Detect instances in which a control element initiates an execution path that contains itself
ASCRM-CWE-704 *technology, unit*	Incompatible data type conversion	Detect instances where a variable, field, member, etc. is declared with a data type, and then is updated with a value from a second data type that is incompatible with the first data type
ASCRM-CWE-772 *technology, unit*	Unreleased resource	Detect instances where a platform resource (messaging, lock, file, stream, directory, etc.) is allocated and assigned a unique resource handler that is used throughout the application, but is never released
ASCRM-CWE-788 *technology, unit*	Memory access after end of buffer	Detect instances where a value is used as an index in a 'Read' or 'Write' access to a buffer; yet none of the operations performed prior the buffer access check the value with regards to the buffer's maximum size
ASCRM-RLB-1 *unit*	Empty exception block	Detect instances where an exception handling block (such as catch and finally blocks) of a function, method, procedure, stored procedure, sub-routine, etc. does not contain any instruction

ASCRM-RLB-2 *technology, unit*	Missing serialization control element	Detect instances where a serializable field, member, etc. has no serialization operation. Notes: * in the case of technologies with classes and interfaces, this means situations where the serializable field is from a class that implements a serializable interface but does not implement a serialization method as part of its list of methods * the serializable nature of an element is technology dependent, for example, serializable capabilities come from sources such as a serializable attribute in .NET and inheritance from the java.io.Serializable interface in Java
ASCRM-RLB-3 *technology, unit*	Serialized data element containing non-serialized items	Detect instances where a serializable field, member, etc. is composed of a non-serializable data element. Notes: * in case of technologies with classes and interfaces, this means situations where the serializable field is from a class that is serializable but owns the non-serializable field * the serializable nature of an element is technology dependent; for example, serializable capabilities come from sources such as a serializable attribute in .NET and inheritance from the java.io.Serializable interface in Java
ASCRM-RLB-4 *technology, unit*	Persistent data without proper comparison controls	Detect instances where the persistent variable, field, member, etc. has no dedicated operation handling comparison operations. Note: • In case of technologies with classes, this means situations where a persistent field is from a class that is made persistent while it does not implement methods from the list of required comparison operations (a JAVA example is the list composed of {'hashCode()','equals()'} methods)
ASCRM-RLB-5 *technology, unit*	Improper runtime resource management	Detect instances where an application is running on an application server, yet uses a low-level resource management API (I/O, sockets, class loaders, etc.) and not the resource management API offered by the application server
ASCRM-RLB-6 *unit*	Improper copy capabilities for data pointers	Detect instances where a variable, field, member, etc. contains a pointer but no dedicated copy operation or copy constructor
ASCRM-RLB-7 *unit*	Self-destruction	Detect instances where a class can self-destruct (an example of a self-destruction in C++ is 'delete this')
ASCRM-RLB-8 *unit*	Variadic parameters	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. has a variable number of parameters, thanks to the variadic parameter in its signature
ASCRM-RLB-9 *unit*	Improper equality comparisons of float-type numerical data	Detect instances where the float values of a variable, field, member, etc. are compared for equality using regular comparison operators (an example in JAVA, is the use of '= =' or '!=')

ASCRM-RLB-10 *system, technology*	Circumventing data access routines	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. executes a data access outside of a dedicated data access component, thus circumventing the intended design for data access. Notes: • the dedicated data access component can be either client-side or server-side, which means that data access components can be developed using non-SQL languages • if there is no dedicated data access component, every data access is a violation
ASCRM-RLB-11 *technology, unit*	Non-final static data in a multi-threaded environment	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. owns a non-final static variable, field, member, etc. while it operates in a multi-threaded environment
ASCRM-RLB-12 *technology, unit*	Improper locking of singleton classes	Detect instances where a singleton class is instantiated without any prior locking mechanism being activated
ASCRM-RLB-13 *technology*	Cyclic dependencies	Detect instances where a module has references that cycle back to itself, e.g., the existence of cycles between packages in JAVA
ASCRM-RLB-14 *technology, unit*	Parent class referencing child class	Detect instances where a parent class has a reference to one of its child classes, directly or indirectly via its methods and fields.
ASCRM-RLB-15 *unit*	Class with virtual method missing destructor	Detect instances where a class contains a virtual method, yet the class does not declare any virtual destructor
ASCRM-RLB-16 *unit*	Parent class missing virtual destructor	Detect instances where, for languages in which custom destructors can be written, the parent has no virtual destructor
ASCRM-RLB-17 *unit*	Child class missing virtual destructor	Detect instances where, for languages in which custom destructors can be written, the child class does not have its own virtual destructor, while its parent class has a virtual destructor
ASCRM-RLB-18 *system, technology, unit*	Hard-coded network resource information	Detect instances where a variable, field, member, etc. is initialized with hard-coded network resource identification information
ASCRM-RLB-19 *unit*	Synchronous call missing timeout	Detect instances where a synchronous call is initiated but the time-out argument is not set or is set to infinite time

3.3 Security

Security measures the extent to which software contains weaknesses that can be exploited to gain unauthorized access to a system to steal data, cause damage, or other malicious acts.

CISQ identifier	Descriptor	Remediation
ASCSM-CWE-22 *system, technology, unit*	Improper path traversal	Detect instances where a user input is ultimately used in a file path creation statement, without any sanitization (based on a list of vetted sanitization functions, methods, procedures, stored procedures, sub-routines, etc.) of the user input value between the user input and the statement.
ASCSM-CWE-78 *system, technology, unit*	OS command injection	Detect instances where a user input is ultimately used to execute an OS command, without any sanitization (based on a list of vetted sanitization functions, methods, procedures, stored procedures, sub-routines, etc.) of the user input value between the user input and the statement.
ASCSM-CWE-79 *system, technology, unit*	Cross-site scripting	Detect instances where a user input is ultimately displayed to the user, without any sanitization (based on a list of vetted sanitization functions, methods, procedures, stored procedures, sub-routines, etc.) of the user input value between the user input and the statement.
ASCSM-CWE-89 *system, technology, unit*	SQL injection	Detect instances where a user input is ultimately used in a SQL statement, without any sanitization (based on a list of vetted sanitization functions, methods, procedures, stored procedures, sub-routines, etc.) of the user input value between the user input and the statement.
ASCSM-CWE-99 *system, technology, unit*	Unsanitized user input used to access a named resource	Detect instances where a user input is ultimately used to access a resource by name, without any sanitization (based on a list of vetted sanitization functions, methods, procedures, stored procedures, sub-routines, etc.) of the user input value between the user input and the statement.
ASCSM-CWE-120 *system, technology, unit*	Buffer overflow	Detect instances where the content of the first buffer is moved into the content of the second buffer while their allocated sizes are incompatible
ASCSM-CWE-129 *system, technology, unit*	Unchecked array index range	Detect instances where a user input is ultimately used in a ‘Read’ or ‘Write’ access to an array, without any range check between the user input and the array access.

ASCSM-CWE-134 *system, technology, unit*	Improper format string neutralization	Detect instances where a user input is ultimately used in a formatting statement, without any sanitization (based on a list of vetted sanitization functions, methods, procedures, stored procedures, sub-routines, etc.) of the user input value between the user input and the statement.
ASCSM-CWE-252-resource *unit*	Unchecked return parameter from resource handling operations	Detect instances where the function, method, procedure, stored procedure, sub-routine, etc. reads, writes, or manages an external resource, yet the value of the return code is not checked anywhere
ASCSM-CWE-327 *unit*	Unvetted cryptographic algorithms	Detect instances where the application uses a cryptographic list which is not part of the list of vetted cryptographic libraries.
ASCSM-CWE-396 *unit*	Catch of overly broad exception types	Detect instances where the function, method, procedure, stored procedure, sub-routine, etc. contains a catch which declares to catch an exception whose type is part of a list of overly broad exception types
ASCSM-CWE-397 *unit*	Throw of overly broad exception types	Detect instances where the function, method, procedure, stored procedure, sub-routine, etc. throws an exception whose type is part of a list of overly broad exception types
ASCSM-CWE-434 *system, technology, unit*	Unsanitized user input in file upload statement	Detect instances where a user input is ultimately used in a file upload statement, without any sanitization (based on a list of vetted sanitization functions, methods, procedures, stored procedures, sub-routines, etc.) of the user input value between the user input and the statement.
ASCSM-CWE-456 *unit*	Uninitialized data element	Detect instances where a variable, field, member, etc. is declared, then is evaluated without ever being initialized prior to the evaluation.
ASCSM-CWE-606 *system, technology, unit*	Unchecked input in loop condition	Detect instances where a user input is ultimately used in the loop condition statement, without any range check between the user input and the loop statement.
ASCSM-CWE-667 *technology*	Improper locking of shared resources	Detect instances where the shared variable, field, member, etc., is accessed outside a critical section of the application.
ASCSM-CWE-672 *technology, unit*	Access to released or expired resources	Detect instances where the platform resource (messaging, lock, file, stream, etc.) is deallocated using its unique resource handler which is used later within the application to try and access the resource.
ASCSM-CWE-681 *technology, unit*	Incompatible numeric type conversion	Detect instances where a variable, field, member, etc. is declared with a numeric type, and then is updated with a value from a second numeric type that is incompatible with the first numeric type

ASCSM-CWE-772 *technology, unit*	Unreleased resource	Detect instances where a platform resource (CPU, messaging, lock, file, stream, etc.) is allocated and assigned a unique resource handler, and its unique resource handler is used throughout the application along a sequence of operations, but none of which is a release statement
ASCSM-CWE-789 *system, technology, unit*	Unchecked range of user input to a buffer	Detect instances where a user input is ultimately used in a 'Read' or 'Write' access to a buffer, without any range check between the user input and the buffer access.
ASCSM-CWE-798 *system, technology, unit*	Hard-coded credentials for remote resources	Detect instances where a variable, field, member, etc., is initialized with a hard-coded literal value, and ultimately used to access a remote resource.
ASCSM-CWE-835 *system, technology, unit*	Infinite recursion	Detect instances where a recursive function, method, procedure, stored procedure, sub-routine, etc., has no execution path to exit the recursion

3.4 Performance Efficiency

Performance Efficiency measures the extent to which software contains weaknesses that can degrade a system's performance or cause excessive use of processor, memory, or other resources.

CISQ identifier	Descriptor	Remediation
ASCPEM-PRF-1 *unit*	Static block initialization	Detect instances where a variable, field, member, etc. is initialized in a static block of code
ASCPEM-PRF-2 *unit*	Immutable text data	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc., creates immutable text data via a string concatenation (which could be avoided by using text buffer instead)
ASCPEM-PRF-3 *unit*	Static data outside of singleton class	Detect instances where a static field or member is declared as static but its parent class is not a singleton class; it does not account for final static fields or members
ASCPEM-PRF-4 *system, technology, unit*	Complex read/write access	Detect instances where a very large table, that is, whose number of rows exceeds a threshold value (default is 1,000,000 rows), is accessed by a SQL statement with too many joins (default threshold value for the maximum number of joins is 5), and too many sub-queries (default threshold value for the maximum number of sub-queries is 3).
ASCPEM-PRF-5 *system, technology, unit*	Incorrect indices	Detect instances where the syntax of the SQL SELECT statement and the index configuration of the SQL table or SQL view causes the DBMS to run sequential searches
ASCPEM-PRF-6 *unit*	Excessive number of indices on large tables	Detect instance where a very large table, that is, whose number of rows exceeds a threshold value (default is 1,000,000 rows), has too many indices (default threshold value for the maximum number of indices is 3)
ASCPEM-PRF-7 *unit*	Excessively large indices on large tables	Detect instances where a very large table, that is, whose number of rows exceeds a threshold value (default is 1,000,000 rows), has an index whose size is too large (default threshold value for the index range is 10)
ASCPEM-PRF-8 *system, technology, unit*	Resource-consuming operation in loop	Detect instances where an operation causing consumption of platform resource (messaging, lock, file, stream, directory, etc.) is directly or indirectly called within a loop body or within a loop condition
ASCPEM-PRF-9 *system, technology, unit*	Excessive data queries in non-stored procedure	Detect instances where a server-side non-stored procedure contains too many data queries (default value for the maximum number of data queries is 5)
ASCPEM-PRF-10 *system, technology, unit*	Excessive data queries in client-side code	Detect instances where a client-side function, method, sub-routine, etc., contains too many data queries (default value for the maximum number of data queries is 2).

ASCPEM-PRF-11 *system, technology, unit*	Data access circumventing data manager	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. accesses data without going a dedicated data manager component (as identified in the vetted data access component list), thus circumventing the authorized data access procedures
ASCPEM-PRF-12 *technology, unit*	Excessively large data element	Detect instances where a variable, field, member, etc., is an aggregate of too many (non-primitive) data types (default value for the maximum number of aggregated non-primitive types is 5)
ASCPEM-PRF-13 *unit*	Data access not using connection pool	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. executes a data resource management action without using a connection pooling capability (the usage of a connection pooling capability is technology dependent; for example, connection pooling is disabled with the addition of 'Pooling=false' to the connection string with ADO.NET or the value of a 'com.sun.jndi.ldap.connect.pool' environment parameter in Java)
ASCPEM-PRF-14 *technology, unit*	Unreleased memory	Detect instances where a memory resource is explicitly allocated to a variable, field, member, etc. which is used throughout the application, but is never released.
ASCPEM-PRF-15 *technology, unit*	Unreleased data	Detect instances where a method references an object, without ever de-referencing it

3.5 Maintainability

Maintainability measures the extent to which software contains weaknesses that make software hard to understand or change, resulting in excessive maintenance time and cost as well as higher defect injection rates.

CISQ identifier	Descriptor	Remediation
ASCMM-MNT-1 *unit*	Control transferred outside of switch statement	Detect instances where the control flow is transferred outside a switch statement (e.g., depending on the technology, by using 'go to', 'continue', or 'break' statements)
ASCMM-MNT-2 *technology*	Excessive inheritance from concrete classes	Detect instances where a class inherits from too many concrete classes (default threshold value for the maximum number of concrete class Inheritances is 1).
ASCMM-MNT-3 *unit*	Hard-coded literals	Detect instances where a literal value is used to initialize a variable, field, member, etc. (exceptions are simple integers and a static constant variable, field, member, etc.)
ASCMM-MNT-4 *System, technology, unit*	Excessive coupling	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. has a Fan-Out value that is too large, that is, with too many references to other objects within the application. (default threshold value for the maximum number of references to other objects within the application is 5)
ASCMM-MNT-5 *unit*	Condition value update within loop	Detect instances where a value of a local variable, field, member, etc. used in the condition of a loop is updated within the loop body
ASCMM-MNT-6 *unit*	Excessive commented-out code	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. contains too much commented-out code (default threshold value for the maximum percentage of commented-out instructions is 2%).
ASCMM-MNT-7 *technology*	Circular dependencies among modules	Detect instances where a module has references that cycle back to itself (for example, in JAVA this pattern means cycles between packages)
ASCMM-MNT-8 *unit*	Excessively large file	Detect instances where a file has too many lines of code (default threshold value for the maximum number of lines of code is 1000)

ASCMM-MNT-9: *system*	Too many/few horizontal layers	Detect instances where a model of the architectural layers of an application contains too may or too few horizontal layers (excluding the vertical utility layers) based on comparison to a threshold value. The default threshold value for the minimal number of horizontal layers is 4, and the default value for maximal number of horizontal layers is 8. Note: • This is a 'control' on the architectural model used in ASCMM-MNT-10 and ASCMM-MNT-12 to avoid defining a model designed to 'pass' these other weaknesses
ASCMM-MNT-10 *system*	Layer-spanning component	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. is part of two architectural layers as defined in a model of the application's architectural layers
ASCMM-MNT-11 *unit*	Excessive cyclomatic complexity	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. has a Cyclomatic Complexity that is too large (default threshold value for maximum Cyclomatic Complexity is 20)
ASCMM-MNT-12 *system*	Layer-skipping calls	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. from a higher horizontal layer directly calls a function, method, procedure, stored procedure, sub-routine, etc. in a lower horizontal layer that is not adjacent to the upper layer making the call, as defined in a model of the application's architectural layers (this excludes the vertical utility layers that can be referenced from any horizontal layer)
ASCMM-MNT-13 *unit*	Excessive parameterization	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. has too many parameters in its signature (default threshold value for the maximum number of parameters is 7)
ASCMM-MNT-14 *unit*	Control element with excessive data operations	Detect instances where a function, method, procedure, stored procedure, sub-routine, etc. has too many SQL or file operations (default threshold value for the maximum number of SQL or file operations is 7)
ASCMM-MNT-15 *unit*	Public data element	Detect instances where a variable, field, member, etc. is declared as public
ASCMM-MNT-16 *technology, unit*	Cross element data access	Detect instances where a method from a class accesses a field, or member from another class
ASCMM-MNT-17 *technology, unit*	Excessive inheritance levels	Detect instances where a class inheritance level is too large (default threshold value for maximum Inheritance levels is 7)
ASCMM-MNT-18 *technology*	Excessive child classes	Detect instances where the class number of children of a class is too large (default threshold value for maximum number of children of a class is 10)

ASCMM-MNT-19 *unit*	Element redundancy	Detect instances of copy-paste between functions, methods, procedures, stored procedures, sub-routines, etc.
ASCMM-MNT-20 *system*	Dead code	Number of instances where a function or method is unreferenced by any other code item in the application (the application determines the scope of the search for code items that could call a function or method element)

4. Appendix A: Certification Measurement Reports

Software Quality Certification

Based on CISQ Quality Characteristic Measures

This report describes the analysis results for <<application>> <<release>> that was analyzed and measured on <<date>> by <<service provider>> using the <<tool>> developed by <<vendor>>. This analysis certifies the level of quality measured in this application when measured against the CISQ Quality Characteristic Measures developed by the Consortium for IT Software Quality and adopted as standards by the Object Management Group (OMG), an international IT standards organization. These measures are developed from counting the number of times critical rules of good architectural and coding practice for each characteristic have been violated. Since structural quality analysis tools differ in the violations of good architectural and coding practices they can detect, the tables will only include results for practices that were evaluated and are the basis for this certification. For each architectural or coding practice within each quality characteristic, the tables below present both the number of times each practice was violated and the number of opportunities for the practice to have been violated within the application. When aggregated over the all violations, these numbers provide the basis for a 6-sigma ranking for each quality characteristic and the aggregated characteristics. That is, the σ level representing the number of violations per million opportunities. This certification provides an evidence-based assessment of the risk this application poses to the business operations it supports or its cost of ownership.

4.1 Summary Results

Quality Characteristics	Violations	Opportunities	6-sigma level
Total Application Quality			
Reliability			
Security			
Performance Efficiency			
Maintainability			

4.2 Reliability

ASCRM #	Architectural or Coding Practice	Vios	Opps	6σ
CWE-120	Avoid buffer operations which fail to ensure that the size of the buffer receiving content is at least as large as the size of the buffer sending content			
CWE-252-data	Avoid improper processing of errors and exceptions from data handling operations			
CWE-252-resource	Avoid improper processing of the execution status of resource handling operations			
CWE-396	Avoid catch units that catch overly broad exception data types			
CWE-397	Avoid throwing overly broad exception data types			
CWE-456	Avoid failure to explicitly initialize software data elements in use			
CWE-674	Avoid recursion			
CWE-704	Avoid data corruption during incompatible mutation			
CWE-772	Avoid failure to release resources after their usage lifetime ends			
CWE-788	Avoid accessing memory locations that are outside a buffer			
RLB-1	Avoid empty exception blocks			
RLB-2	Avoid storable data elements without serialization controls			
RLB-3	Avoid serializable data elements with non-serializable items			
RLB-4	Avoid persistant data elements without proper comparison controls			
RLB-5	Avoid using deployed components from application servers while using control elements from the list of low-level resource management APIs			
RLB-6	Avoid storable data elements containing pointers without proper copy controls			
RLB-7	Avoid class instances that can self-destruct			
RLB-8	Avoid using variadic parameter elements			
RLB-9	Avoid comparing equality of float-type data using regular comparison operators			
RLB-10	Avoid actions circumventing dedicated and specialized data manager components			
RLB-11	Avoid actions that own an unsafe non-final static data element while operating in a multi-threaded environment			
RLB-12	Avoid singleton classes without proper locking mechanisms			
RLB-13	Avoid circular dependencies between modules			
RLB-14	Avoid parent class references to child classes			
RLB-15	Avoid failing to include a virtual destructor in a class that includes a virtual method			
RLB-16	Avoid failing to include a virtual destructor in a parent class			
RLB-17	Avoid failing to include a virtual destructor in a child class despite the existence of a virtual destructor in the parent class			
RLB-18	Avoid the existence of hard-coded values corresponding to network resource identifications			
RLB-19	Avoid synchronous remote resource access without handling time-out capabilities			
Total aggregated Reliability violation results				

4.3 Security

ASCSM #	Architectural or Coding Practice	Vios	Opps	6σ
CWE-22	Avoid failure to sanitize user input in path manipulation operations			
CWE-78	Avoid failure to sanitize user input used as operating system commands			
CWE-79	Avoid failure to sanitize user input used in output generation operations (cross-site scripting)			
CWE-89	Avoid failure to sanitize user input used in SQL compilation operations			
CWE-99	Avoid failure to sanitize user input in use as resource names or references			
CWE-120	Avoid buffer operations among buffers with incompatible sizes			
CWE-129	Avoid failure to check range of user input used as array index			
CWE-134	Avoid failure to sanitize user input used in formatting operations			
CWE-252-resource	Avoid improper processing of the execution status of resource handling operations			
CWE-327	Avoid failure to use vetted cryptographic libraries			
CWE-396	Avoid catch units that catch overly broad exception data types			
CWE-397	Avoid throwing overly broad exception data types			
CWE-434	Avoid failure to sanitize user input used in file upload operations			
CWE-456	Avoid failure to explicitly initialize software data elements in use			
CWE-606	Avoid failure to check range of user input used in iteration control			
CWE-667	Avoid improper locking of shared data			
CWE-672	Avoid access to a released, revoked, or expired resource			
CWE-681	Avoid conversions between incompatible numerical data types			
CWE-772	Avoid failure to release resources after their usage lifetime ends			
CWE-789	Avoid failure to relase a platform resource after its useful lifetime			
CWE-798	Avoid using hard-coded credentials for remote authentication			
CWE-835	Avoid infinite loops resulting from unreachable exit conditions			
Total aggregated Security violation results				

4.4 Performance Efficiency

ASCPEM #	Architectural or Coding Practice	Vios	Opps	6σ
PRF-1	Avoid initializing a data element in a static block			
PRF-2	Avoid unnecessary creation of immutable data elements			
PRF-3	Avoid data elements declared static whose parent is not a singleton class			
PRF-4	Avoid queries on tables of >1M rows that involve >5 joins or >3 sub-queries			
PRF-5	Avoid unnecessary full scans of data tables			
PRF-6	Avoid >3 indices on data tables of >1M rows			
PRF-7	Avoid index ranges >10 on data tables of >1M rows			
PRF-8	Avoid resource consuming operations found directly or indirectly within loops			
PRF-9	Avoid failure to use dedicated stored procedures when multiple data accesses are needed			
PRF-10	Avoid client-side actions that embed >2 queries performed outside of a data manager			
PRF-11	Avoid circumventing data access managers			
PRF-12	Avoid creating data elements that aggregate >5 non-primitive data types			
PRF-13	Avoid failure to share database connections via a connection pool			
PRF-14	Avoid failure to release used memory			
PRF-15	Avoid failure to release used data elements			
Total aggregated Performance Efficiency violation results				

4.5 Maintainability

ASCMM #	Architectural or Coding Practice	Vios	Opps	6σ
MNT-1	Avoid the unconditional transfer of control flow outside of switch structures			
MNT-2	Avoid multiple inheritance of classes with concrete implementations			
MNT-3	Avoid hard-coded non-trivial values in the code			
MNT-4	Avoid code units with >5 references to other objects within the application			
MNT-5	Avoid updating loop values within the loop			
MNT-6	Avoid having >2% commented-out instructions in the application			
MNT-7	Avoid circular dependencies between code units			
MNT-8	Avoid files with >1000 lines of code			
MNT-9	Avoid having <4 or >8 horizontal layers in an application			
MNT-10	Avoid spreading code units across two or more architectural layers			
MNT-11	Avoid code units with Cyclomatic Complexity >20			
MNT-12	Avoid references to code elements that are not in adjacent architectural layers			
MNT-13	Avoid code elements with >7 parameters in its signature			
MNT-14	Avoid code elements with >7 data operations			
MNT-15	Avoid openly available data elements circumventing encapsulation			
MNT-16	Avoid direct access to data elements of another entity			
MNT-17	Avoid class elements with >7 levels of inheritance			
MNT-18	Avoid class elements with >10 child elements			
MNT-19	Avoid code elements that contain multiple computational objects that are identical to computational objects in another code element			
MNT-20	Avoid inactive code blocks that are unreferenced by any other code element			
Total aggregated Maintainability violation results				

5. Appendix B: Java-EE Weakness Pattern Examples

These tables present examples of weakness patterns in Java-EE to provide a guideance on the patterns a conformant technology should be able to detect. If a cell is empty, then it is assumed there are no structures unique to Java EE that are required beyond the general capability for detecting the elements of the pattern. An 'N/A' indicates the weakness cannot occur in Java EE.

5.1 Reliability Examples

CISQ identifier	JEE detection aspects	JEE-specific example

ASCRM-CWE-120	• mostly N/A,
	• except for JNI (JAVA Native Interface) and for system calls
		• OWASP unsafe JNI
		
		class Echo {
		public native void runEcho();
		static {
		
		System.loadLibrary("echo");
		}
		public static void main(String[] args) {
		new Echo().runEcho();
		}
		}
		with
		#include <jni.h>
		#include "Echo.h"//the java class above compiled with javah
		#include <stdio.h>
		
		JNIEXPORT void JNICALL
		Java_Echo_runEcho(JNIEnv *env, jobject obj)
		{
		char buf[64];
		gets(buf);
		printf(buf);
		}

ASCRM-CWE-252-data	• The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved. (Src. CWE-252)	

ASCRM-CWE-252-resource	• The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved. (Src. CWE-252)	• CWE-252 sample
		•
		FileInputStream fis;
		
		byte[] byteArray = new byte[1024];
		for (Iterator i=users.iterator(); i.hasNext();) {
		String userName = (String) i.next();
		String pFileName = PFILE_ROOT + "/" + userName;
		FileInputStream fis = new FileInputStream(pFileName);
		fis.read(byteArray); // the file is always 1k bytes
		fis.close();
		processPFile(userName, byteArray);
		The code loops through a set of users, reading a private data file for each user. The programmer assumes that the files are always 1 kilobyte in size and therefore ignores the return value from Read(). If an attacker can create a smaller file, the program will recycle the remainder of the data from the previous user and treat it as though it belongs to the attacker.

ASCRM-CWE-396	• (none so far)	• QR-7962 try { ,,, } catch (Exception /*e*/) // <= VIOLATION { ,,, }

ASCRM-CWE-397	• (none so far)	• QR-7824
		
		void f()
		{
		,,,
		throw new Exception("My Message"); // <= VIOLATION
		,,,
		}

ASCRM-CWE-456	• (none so far)
	• not N/A (cf. samples)
		• CWE-456 samples
		• private User user;
		• public void someMethod() {
		• // Do something interesting.
		• ...
		• // Throws NPE if user hasn't been properly initialized.
		• String username = user.getName();
		}
		public class BankManager {
		// user allowed to perform bank manager tasks
		private User user = null;
		private boolean isUserAuthentic = false;
		// constructor for BankManager class
		public BankManager() {
		...
		}
		// retrieve user from database of users
		public User getUserFromUserDatabase(String username){
		...
		}
		// set user variable using username
		public void setUser(String username) {
		this.user = getUserFromUserDatabase(username);
		}
		// authenticate user
		public boolean authenticateUser(String username, String password) {
		if (username.equals(user.getUsername()) && password.equals(user.getPassword())) {
		isUserAuthentic = true;
		}
		return isUserAuthentic;
		}
		// methods for performing bank manager tasks
		...
		}

ASCRM-CWE-674	• (none so far)	
ASCRM-CWE-704	• (none so far)	

ASCRM-CWE-772	• Check resource types of the pattern search: o stream o file o lock o thread (not so simple as the thread itself must be wired to cooperate so the pattern is not enough) o ... • Check usage of data flow link	
ASCRM-CWE-788	• N/A	
ASCRM-RLB-1	• comment does not count as non-empty	
ASCRM-RLB-2	• Check depth of inheritance used to find serializable elements	

ASCRM-RLB-3	• Check depth of inheritance used to find serializable elements	• QR-7650
		
		public static final class SomeType {}
		
		public class SerializableField implements Serializable {
		private String str;
		private SomeType field; // VIOLATION
		}

ASCRM-RLB-4	• Check list of supported persistance frameworks:
	o homemade (thanks to data accesses),
	o Session Bean and JDBC,
	o JDBC,
	o hibernate,
	o iBATIS,
	o JPA,
	o JCA with legacy systems
	...
		• QR-7504
		
		public class B {
		private long id;
		...
		
		private void setId(long id) {
		this.id = id;
		}
		
		public long getId() {
		return id;
		}
		...
		// VIOLATION: equals and hashCode are not defined
		}
		with
		<hibernate-mapping >
		
		<class name="A" table ="A">
		<id name="id">
		<generator class="increment"/>
		</id>
		
		<set name="b_items" lazy="true" table ="B">
		<key column="B_ID"/>
		<one-to-many class="B"/>
		</array>
		</class>
		
		<class name="B" table="B" lazy="true">
		<id name="id" column="B_ID">
		<generator class="increment"/>
		</id>
		</class>
		
		</hibernate-mapping>

ASCRM-RLB-5	• Check list of supported resource types of the pattern search: threads, sockets, ...
	• Check list of supported Application Servers
		• QR-7728
		
		class BasicThread implements Runnable {
		public void run() {
		[...]
		}
		}
		
		class SimpleThread extends Thread {
		public SimpleThread(String str) {
		super(str);
		}
		public void run() {
		[...]
		}
		}
		
		class ThreadLaunch {
		public static void main (String args[]) {
		Runnable runnable = new BasicThread();
		new Thread(runnable).start(); // VIOLATION
		new SimpleThread("Thread").start(); // VIOLATION
		}
		}
		with
		ejb-jar.xml
		web.xml
		...
		in application

ASCRM-RLB-6		
ASCRM-RLB-7		
ASCRM-RLB-8		

ASCRM-RLB-9	• (none so far)	• QR-8096
		
		float a = 0.15f + 0.15f;
		float b = 0.1f + 0.2f;
		if(a == b) {...} // VIOLATION
		if(a != b) {...} // VIOLATION

ASCRM-RLB-10	• Check the ability to defined target architecture
	• Check linking capabilities regarding
	o reflection
	o overloading
	o overriding through inheritance
	o overriding though interface
	o cross-technology
		• QR-7782
		try { ... }
		catch (MyException e)
		{ ,,, }
		finally
		{
		// DO NOTHING <= VIOLATION
		}
		• QR-7788
		
		try { ... }
		catch (MyException e)
		{
		// DO NOTHING <= VIOLATION
		}

ASCRM-RLB-11	• Check the detection of the multi-thread context: struts, ...	• QR-7154
		
		Public class BaseApplicationAction extends Action
		{
		static int StaticField; // VIOLATION
		long Id; // VIOLATION
		
		public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) throws Exception
		[...]
		}

ASCRM-RLB-12	• Check list of supported locking mechanisms
	o synchronized
	o java.util.concurrent.*
		• QR-7438
		
		class Singleton
		{
		private static Singleton instance;
		private Singleton() {}
		public static Singleton getInstance() {
		if (instance == null) // VIOLATION
		instance = new Singleton();
		return instance;
		}
		}

ASCRM-RLB-13	• Check ability to find indirect cycles (A->B->C->A) • Check limitiations used to find indirect cycles • Check linking capabilities regarding o reflection o overloading o overriding through inheritance o overriding though interface	

ASCRM-RLB-14	• Check depth of inheritance used to find references • Check linking capabilities regarding o reflection o overloading o overriding through inheritance o overriding though interface	
ASCRM-RLB-15	• N/A	
ASCRM-RLB-16	• N/A	
ASCRM-RLB-17	• N/A	
ASCRM-RLB-18	• Check list of supported network resource type: o IP v4 o IP v6 o UNC / path o url o ...	
ASCRM-RLB-19	• Check list of supported synchronous call types: o java.net.Socket o java.util.concurrent.Future<V> o javax.jms.MessageConsumer o ...	.

5.2 Security Examples

CISQ identifier	JEE detection aspects	JEE-specific example

ASCSM-CWE-22	• Check usage of dataflow links
	• Check list of supported user input operations
	• Check list of supported path creation operations
	• Check coverage of JEE languages and frameworks
	• Check ability to input list of vetted sanitization operations
		• CWE-73
		
		String rName = request.getParameter("reportName");
		File rFile = new File("/usr/local/apfr/reports/" + rName);
		...
		rFile.delete();
		fis = new FileInputStream(cfg.getProperty("sub")+".txt");
		amt = fis.read(arr);
		out.println(arr);

ASCSM-CWE-78	• Check usage of dataflow links
	• Check list of supported user input operations
	• Check list of supported OS command execution operations
	• Check coverage of JEE languages and frameworks
	• Check ability to input list of vetted sanitization operations
		• CWE-78
		
		String script = System.getProperty("SCRIPTNAME");
		if (script != null)
		System.exec(script);
		public String coordinateTransformLatLonToUTM(String coordinates)
		{
		String utmCoords = null;
		try {
		String latlonCoords = coordinates;
		Runtime rt = Runtime.getRuntime();
		Process exec = rt.exec("cmd.exe /C latlon2utm.exe -" + latlonCoords);
		// process results of coordinate transform
		// ...
		
		}
		catch(Exception e) {...}
		return utmCoords;
		
		}

ASCSM-CWE-79	• Check usage of dataflow links • Check list of supported user input operations • Check list of supported user display operations • Check coverage of JEE languages and frameworks • Check ability to input list of vetted sanitization operations	

ASCSM-CWE-89	• Check usage of dataflow links • Check list of supported user input operations • Check list of supported sql compilation statements • Check coverage of JEE languages and frameworks • Check ability to input list of vetted sanitization operations	
ASCSM-CWE-99	• Check usage of dataflow links • Check list of supported user input operations • Check list of supported access by name statements • Check coverage of JEE languages and frameworks • Check ability to input list of vetted sanitization operations	

ASCSM-CWE-120	• mostly N/A,
	• except for JNI (JAVA Native Interface) and for system calls
		• OWASP unsafe JNI
		
		class Echo {
		public native void runEcho();
		
		static {
		
		System.loadLibrary("echo");
		}
		
		public static void main(String[] args) {
		new Echo().runEcho();
		}
		}
		
		with
		
		#include <jni.h>
		#include "Echo.h"//the java class above compiled with javah
		#include <stdio.h>
		
		JNIEXPORT void JNICALL
		Java_Echo_runEcho(JNIEnv *env, jobject obj)
		{
		char buf[64];
		gets(buf);
		printf(buf);
		}

ASCSM-CWE-129	• Check usage of dataflow links • Check list of supported user input operations • Check list of supported array access syntaxes • Check coverage of JEE languages and frameworks • Check ability to input list of vetted sanitization operations	.

ASCSM-CWE-134	• Check usage of dataflow links • Check list of supported user input operations • Check list of supported format operations • Check coverage of JEE languages and frameworks • Check ability to input list of vetted sanitization operations	

ASCSM-CWE-252-resource	• The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved. (Src. CWE-252)	• CWE-252 sample
		• FileInputStream fis;
		• byte[] byteArray = new byte[1024];
		• for (Iterator i=users.iterator(); i.hasNext();) {
		• String userName = (String) i.next();
		• String pFileName = PFILE_ROOT + "/" + userName;
		• FileInputStream fis = new FileInputStream(pFileName);
		• fis.read(byteArray); // the file is always 1k bytes
		• fis.close(); processPFile(userName, byteArray);
		
		The code loops through a set of users, reading a private data file for each user. The programmer assumes that the files are always 1 kilobyte in size and therefore ignores the return value from Read(). If an attacker can create a smaller file, the program will recycle the remainder of the data from the previous user and treat it as though it belongs to the attacker.

ASCSM-CWE-327	• (none so far)	
ASCSM-CWE-396	• (none so far)	• QR-7962 try { ,,, } catch (Exception /*e*/) // <= VIOLATION { ,,, }

ASCSM-CWE-397	• (none so far)	• QR-7824
		
		void f()
		{
		,,,
		throw new Exception("My Message"); // <= VIOLATION
		,,,
		}

ASCSM-CWE-434	• Check usage of dataflow links • Check list of supported user input operations • Check list of supported file upload operations • Check coverage of JEE languages and frameworks • Check ability to input list of vetted sanitization operations	.

ASCSM-CWE-456	• (none so far)
	• not N/A (cf. samples)
		• CWE-456 samples
		• private User user;
		•
		• public void someMethod() {
		•
		• // Do something interesting.
		• ...
		• // Throws NPE if user hasn't been properly initialized.
		• String username = user.getName();
		
		}
		public class BankManager {
		// user allowed to perform bank manager tasks
		private User user = null;
		private boolean isUserAuthentic = false;
		// constructor for BankManager class
		public BankManager() {
		...
		}
		// retrieve user from database of users
		public User getUserFromUserDatabase(String username){
		...
		}
		// set user variable using username
		public void setUser(String username) {
		this.user = getUserFromUserDatabase(username);
		}
		// authenticate user
		public boolean authenticateUser(String username, String password) {
		if (username.equals(user.getUsername()) && password.equals(user.getPassword())) {
		isUserAuthentic = true;
		}
		return isUserAuthentic;
		
		}
		// methods for performing bank manager tasks
		...
		
		}

ASCSM-CWE-606	• Check usage of dataflow links • Check list of supported user input operations • Check list of supported loop syntaxes • Check coverage of JEE languages and frameworks • Check ability to input list of vetted sanitization operations	

ASCSM-CWE-667	• Check list of supported locking mechanisms
	o synchronized
	o java.util.concurrent.*
		• CWE-667
		
		private long someLongValue;
		public long getLongValue() {
		return someLongValue;
		}
		
		public void setLongValue(long l) {
		someLongValue = l;
		}

ASCSM-CWE-672	• Check resource types of the pattern search: o stream o file o lock o thread (not so simple as the thread itself must be wired to cooperate so the pattern is not enough) o ... • Check usage of data flow links	
ASCSM-CWE-681	• (none so far)	• CWE-681 int i = (int) 33457.8f;

ASCSM-CWE-772	• Check resource types of the pattern search:
	o stream
	o file
	o lock
	o thread (not so simple as the thread itself must be wired to cooperate so the pattern is not enough)
	o ...
	• Check usage of data flow links
		• CWE-772
		
		private void processFile(string fName)
		{
		BufferReader in = new BufferReader(new FileReader(fName));
		String line;
		while ((line = in.ReadLine()) != null)
		{
		processLine(line);
		}
		}
		
		with a call to Close() only in Finalize()

ASCSM-CWE-789	• Check usage of dataflow links • Check list of supported user input operations • Check coverage of JEE languages and frameworks	• CWE-789 unsigned int size = GetUntrustedInt(); HashMap list = new HashMap(size);
ASCSM-CWE-798	• Check usage of dataflow links • Check list of supported authentication operations • Check coverage of JEE languages and frameworks • check list of supported initialization syntax o hard-coded in the JAVA code o hard-coded in Resource Bundle o ...	• CWE-798 DriverManager.getConnection(url, "scott", "tiger"); # Java Web App ResourceBundle properties file ... webapp.ldap.username=secretUsername webapp.ldap.password=secretPassword ...
ASCSM-CWE-835	• Check ability to find indirect cycles (A->B->C->A) • Check limitations used to find indirect cycles • Check linking capabilities regarding o reflection o overloading o overriding through inheritance o overriding though interface • Check linking capabilities regarding cross-technology	

5.3 Performance Efficiency

CISQ identifier	JEE detection aspects	JEE-specific example
ASCPEM-PRF-1	• (none so far)	

ASCPEM-PRF-2	• check list of supported concatenation syntaxes	• QR-7200
		
		String result = "hello"; for (int i = 0; i < 1500; i++) {
		result += "hello"; // VIOLATION
		}

ASCPEM-PRF-3	• (none so far)	• QR-7562
		class Sample {
		// VIOLATION
		static HashMap my_map;
		...
		}
		
		• QR-7704
		@Stateful
		public class MyBean implements MyRemoteBean {
		private static String myName = "MyBean"; //
		VIOLATION
		...
		public int aMethod() {
		...
		}
		...
		}

ASCPEM-PRF-4	• check ability to process embedded SQL • check ability to process dynamic SQL	
ASCPEM-PRF-5	• check ability to process embedded SQL • check ability to process dynamic SQL • check ability to process SQL side	
ASCPEM-PRF-6	• N/A	
ASCPEM-PRF-7	• N/A	

ASCPEM-PRF-8	• check list of supported expensive operations
	o string concatenation
	o SQL query
	o instantiation
	o remote call
	o ...
		• QR-7200
		String result = "hello"; for (int i = 0; i < 1500; i++) {
		result += "hello"; // VIOLATION
		}
		
		• QR-7424
		PreparedStatement updateSales;
		String updateString = "update COFFEES " +
		"set SALES = ? where COF_NAME like ?";
		updateSales = con.prepareStatement(updateString);
		
		int len = coffees.length;
		for(int i = 0; i < len; i++) {
		updateSales.setInt(1, salesForWeek[i]);
		updateSales.setString(2, coffees[i]);
		updateSales.executeUpdate(); // VIOLATION
		}
		
		• QR-7210
		public class MyLoop {
		public void printCount() {
		for (int i = 0; i < 100; i++) {
		StringBuffer sb = new StringBuffer(); //
		VIOLATION
		sb.append("count = ");
		sb.append(i);
		System.out.println(sb);
		}
		}
		}
		
		• QR-7204
		for (int i=0; i < tab.length(); i++) { // VIOLATION
		tab[i] = i;
		}
		
		• QR-7206
		class BaseExample { ... }
		class Example1 extends BaseExample { ... }
		class Example2 extends BaseExample { ... }
		
		class Test {
		BaseExample[] exList;
		void method () {
		for (int i = exList.length-1; i >= 0; i--) {
		if (exList[i] instanceof Example1) { // VIOLATION
		((Example1) exList[i]).aMethod1();
		} elseif (exList[i] instanceof Example2) { //
		VIOLATION
		((Example2) exList[i]).aMethod2();
		}
		}
		}

ASCPEM-PRF-9	• N/A	

ASCPEM-PRF-10	• check ability to process embedded SQL
	• check ability to process dynamic SQL
		• QR-8110
		
		/**
		* @param resultat
		* @throws SQLException
		*/
		public void meth_1() throws SQLException
		{
		Statement statement = connection.createStatement();
		String sql1 = "INSERT INTO STUDENTS VALUES" + "(100,'TOTO','TITI', {d '2001-12-16'})";
		String sql2 = "INSERT INTO STUDENTS VALUES" + "(100,'MR','T', {d '2002-10-1'})";
		String sql3 = "INSERT INTO STUDENTS VALUES" + "(10,'MR','T', {d '2002-10-1'})";
		
		statement.executeUpdate(sql1);
		statement.executeUpdate(sql2);
		statement.executeUpdate(sql3);
		}
		
		/**
		* @param resultat
		* @throws SQLException
		*/
		public void meth_2() throws SQLException
		{
		Statement statement = connection.createStatement();
		String sql1 = "INSERT INTO STUDENTS VALUES" + "(100,'TOTO','TITI', {d '2001-12-16'})";
		String sql2 = "INSERT INTO STUDENTS VALUES" + "(100,'MR','T', {d '2002-10-1'})";
		
		statement.executeUpdate(sql1);
		statement.executeUpdate(sql2);
		boolean rs = statement.execute("SELECT * FROM STUDENTS");
		if (rs)
		{
		// print result
		}
		}

ASCPEM-PRF-11	• check ability to input data access component	

ASCPEM-PRF-12	• (none so far)	
ASCPEM-PRF-13	• check list of supported data resource access operations	• QR-7638 Connection con = DriverManager.getConnection(URL, user, password); // VIOLATION
ASCPEM-PRF-14	• N/A	

ASCPEM-PRF-15	• check list of covered cases: collection, ...	• QR-7562
		
		class Sample {
		// VIOLATION
		static HashMap my_map;
		...
		}

5.4 Maintainability Examples

CISQ identifier	JEE detection aspects	JEE-specific example

ASCMM-MNT-1	• check list of transfer syntaxes
	• check ability to generate control flow (as opposed to simple "grep" method)
		• QR-7910
		
		`try {`
		`  ...`
		`  throw IllegalArgumentException();`
		`}`
		`finally {`
		`  // VIOLATION: the IllegalArgumentException will never be delivered to the caller. The finally block will cause the exception to be discarded.`
		`  return r;`
		`}`
		
		• QR-8032
		
		`for (int i = 0; i < 10; i++)`
		`{`
		`  if (true)`
		`  {`
		`  break; // Violation`
		`  }`
		`  // ...`
		`}`

ASCMM-MNT-2	• (none so far)	
ASCMM-MNT-3	• check list of trivial acceptable hard-coded literals	
ASCMM-MNT-4	• Check linking capabilities regarding o reflection o overloading o overriding through inheritance o overriding though interface o cross-technology	
ASCMM-MNT-5		
ASCMM-MNT-6	• check list of supported loop syntaxes	
ASCMM-MNT-7	• (none so far)	

ASCMM-MNT-8	• Check ability to find indirect cycles (A->B->C->A) • Check limitiations used to find indirect cycles • Check linking capabilities regarding o reflection o overloading o overriding through inheritance o overriding though interface	
ASCMM-MNT-9:	• (none so far)	
ASCMM-MNT-10	• check ability to import target architecture model layers	
ASCMM-MNT-11	• check ability to import target architecture model layers	
ASCMM-MNT-12	• check ability to generate control flow (as opposed to simple "grep" method)	
ASCMM-MNT-13	• check ability to import target architecture model layers and calls	
ASCMM-MNT-14	• (none so far)	
ASCMM-MNT-15	• (none so far)	
ASCMM-MNT-16	• (none so far)	
ASCMM-MNT-17	• Check linking capabilities regarding o reflection o overloading o overriding through inheritance o overriding though interface	
ASCMM-MNT-18	• check depth of inheritance tree for the pattern search	
ASCMM-MN-19	• (none so far)	
ASCMM-MNT-20	• check ability to generate elementary tokens (as opposed to hash map copy paste detection)	

6. Appendix C: CISQ

The purpose of the Consortium for IT Software Quality (CISQ) is to develop specifications for automated measures of software quality characteristics taken on source code. These measures were designed to provide international standards for measuring software structural quality that can be used by IT organizations, IT service providers, and software vendors in contracting, developing, testing, accepting, and deploying IT software applications. Executives from the member companies that joined CISQ prioritized the quality characteristics of Reliability, Security, Performance Efficiency, and Maintainability to be developed as measurement specifications.

CISQ strives to maintain consistency with ISO/IEC standards to the extent possible, and in particular with the ISO/IEC 25000 series that replaces ISO/IEC 9126 and defines quality measures for software systems. In order to maintain consistency with the quality model presented in ISO/IEC 25010, software quality characteristics are defined for the purpose of this specification as attributes that can be measured from the static properties of software, and can be related to the dynamic properties of a computer system as affected by its software. However, the 25000 series, and in particular ISO/IEC 25023 which elaborates quality characteristic measures, does not define these measures at the source code level. Thus, this and other CISQ quality characteristic specifications supplement ISO/IEC 25023 by providing a deeper level of software measurement, one that is rooted in measuring software attributes in the source code.

Companies interested in joining CISQ held executive forums in Frankfurt, Germany; Arlington, VA; and Bangalore, India to set strategy and direction for the consortium. In these forums four quality characteristics were selected as the most important targets for automation—reliability, security, performance efficiency, and maintainability. These attributes cover four of the eight quality characteristics described in ISO/IEC 25010. Figure 1 displays the ISO/IEC 25010 software product quality model with the four software quality characteristics selected for automation by CISQ highlighted in orange. Each software quality characteristic is shown with the sub-characteristics that compose it.