Date: January 2016

OBJECT MANAGEMENT GROUP®

Automated Source Code Reliability Measure™ (ASCRM™)

V1.0

OMG Document Number: formal/2016-01-03

Standard document URL: http://www.omg.org/spec/ASCRM/1.0

Normative Machine Consumable File(s):

http://www.omg.org/spec/ASCRM/20141211/AutomatedSourceCodeReliabilityMeasureSPMS.xmi

http://www.omg.org/spec/ASCRM/20141111/AutomatedSourceCodeReliabilityMeasureSMM.smm

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

LICENSES

PATENTS

GENERAL USE RESTRICTIONS

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 109 Highland Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

CORBA®, CORBA logos®, FIBO®, Financial Industry Business Ontology®, FINANCIAL INSTRUMENT GLOBAL IDENTIFIER®, IIOP®, IMM®, Model Driven Architecture®, MDA®, Object Management Group®, OMG®, OMG Logo®, SoaML®, SOAML®, SysML®, UAF®, Unified Modeling Language®, UML®, UML Cube Logo®, VSIPL®, and XMI® are registered trademarks of the Object Management Group, Inc.

For a complete list of trademarks, see: http://www.omg.org/legal/tm_list.htm. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG’s Issue Reporting

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue (http://www.omg.org/report_issue.htm).

Table of Contents

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language®); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel™); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:

http://www.omg.org/spec

Specifications are organized by the following categories:

Business Modeling Specifications

Middleware Specifications

- **CORBA/IIOP**
- **Data Distribution Services**
- **Specialized CORBA**

IDL/Language Mapping Specifications

Modeling and Metadata Specifications

- **UML, MOF, CWM, XMI**
- **UML Profile**

Modernization Specifications

Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

- **CORBAServices**
- **CORBAFacilities**

OMG Domain Specifications

CORBA Embedded Intelligence Specifications

CORBA Security Specifications

Signal and Image Processing Specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
109 Highland Avenue
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text, table text, bullets

Helvetica/Arial – 9 or **10 pt. Bold:** OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier new/Courier – 10 pt. Bold`: Programming Languages

Helvetica/Arial – 10 pt.: Exceptions

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/report_issue.htm*

1 Scope

1.1 Overview

The purpose of this specification is to establish a standard measure of reliability based on detecting violations of good architectural and coding practices that could result in unreliable operation such as outages, data corruption, and lengthy recovery from system failures. Establishing a standard for this measure is important because such measures are being used in outsourcing and system development contracts without having an approved international standard to reference. They are also critical to other software-intensive OMG initiatives such as The Internet of Things. The Consortium for IT Software Quality (CISQ) was formed as a special interest group of OMG to create specifications for automating standard measures of software quality attributes and submit them to OMG for approval.

1.2 CISQ Background

This specification defines a method for automating the measurement of Reliability from violations of architectural and coding practice that affect an application’s reliability, robustness, and resilience. The violations included in the CISQ measure were selected from a large set of candidate violations related to reliability issues. The final set of violations were chosen through a voting process among CISQ member organizations that resulted in a limited set of violations that member organizations believed were sufficiently severe that they had to be remediated. This process will be described more fully in subsequent sub clauses.

1.3 Overview of Software Quality Characteristic Measurement

Measurement of the internal or structural quality aspects of software has a long history in software engineering (Curtis, 1980). Software quality characteristics are increasingly being incorporated into development and outsourcing contracts as the equivalent of service level agreements. That is, target thresholds based on quality characteristic measures are being set in contracts for delivered software. Currently there are no standards for most of the software quality characteristic measures being used in contracts. ISO/IEC 25023 purports to address these measures, but only provides measures of external behavior and does not define measures that can be developed from source code during development. Consequently, providers are subject to different interpretations and calculations of common quality characteristics in each contract. This specification addresses one aspect of this problem by providing a specification for measuring one quality characteristic, Reliability, from the source code. This specification is one of four specifying source code level measures of quality characteristics. The other three specify quality characteristic measures for Security, Performance Efficiency, and Maintainability.

Violations of Good Architectural and Coding Practice—The most recent advance in measuring the structural quality of software is based on the analysis and measurement of violations of good architectural and coding practice that can be detected by statically analyzing the source code. The CWE/SANS 25 and OWASP Top Ten lists of security weaknesses are examples of this approach. These lists are drawn from the Common Weakness Enumeration (CWE) repository maintained by MITRE Corporation. CWE contains descriptions of over 800 weaknesses that represent violations of good architectural and coding practice in software that can be exploited to gain unauthorized entry into a system. The Software Assurance community has been a leader in this area of measurement by championing the detection of code weaknesses as a way of improving one aspect of software quality—software security.

Unfortunately there are no equivalent repositories of weaknesses for Reliability, Performance Efficiency, or Maintainability. Knowledge of these weaknesses is spread across software engineering textbooks, expert blogs, and information sharing sites such as github. The CISQ measure for Reliability can fill the void for a consensus body of knowledge about the most egregious Reliability problems that should be detected and remediated in source code. Currently, no standards or guidelines have been developed for calculating component or application-level reliability measures that aggregate weaknesses detected through static code analysis into application-level Reliability measures. CISQ will be providing recommendations for these aggregation and scaling techniques. However, these techniques are not part of this standard since different measurement objectives are best served by different scoring techniques.

Using violations of good architectural and coding practices in software quality metrics presents several challenges for establishing baselines. Growth in the number of unique violations to be detected could continually raise the bar for measuring quality, reducing the validity of baseline comparisons. Further, different vendors will detect different sets of violations, making comparisons difficult across commercial software quality measurement offerings. One solution to this problem is to create a stable list of violations that are used for computing a baseline for each quality characteristic. The CISQ Automated Source Code Reliability Measure was developed by a team of industry experts to form the basis for a stable baseline measure.

1.4 Development of the Automated Source Code Reliability Measure

The original 24 CISQ member companies provided experts to working groups whose charter was to define CISQ measures. Violations of good architectural and coding practice that a high probability of causing reliability problems were selected by an international team of experts drawn from the 24 organizations that joined CISQ in 2010. These organizations included IT departments in Fortune 200 companies, system integrators/outsourcers, and vendors that provide quality-related products and services to the IT market. The experts met several times per year for two years in the US, France, and India to develop a broad list of candidate reliability weaknesses and then pare it down to a set they felt had to be remediated to avoid serious operational problems.

The work group began by defining reliability issues, quality rules for avoiding these issues, and measures based on counting violations of these rules. They developed lists of issues and quality rules by drawing information from company defect logs, their career experience in different environments, and industry sources such as books and blogs. In order to reduce the work group’s initial list to a critical set of reliability violations, work group members individually evaluated the severity of each violation. High severity violations were judged to be those that must be fixed in a future release because of their operational risk or cost impact. The work group went through several rounds of eliminating lower severity violations and re-rating the severity of remaining violations until a final list was established as the quality measure elements to be incorporated into this specification.

1.5 Structure of the Automated Source Code Reliability Measure

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 1.1 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.

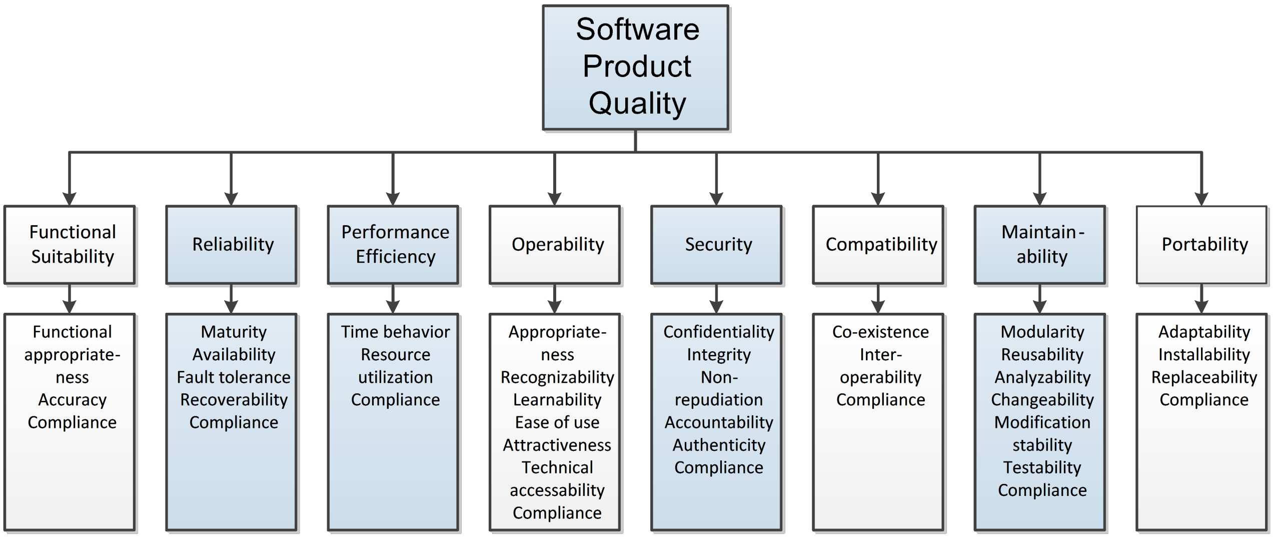
Software Product Quality
Functional Suitability
Reliability
Performance Efficiency
Operability
Security
Compatibility
Maintain-ability
Portability
Functional appropriate-ness Accuracy Compliance
Maturity Availability Fault tolerance Recoverability Compliance
Time behavior Resource utilization Compliance
Appropriate-ness Recognizability Learnability Ease of use Attractiveness Technical accessability Compliance
Confidentiality Integrity Non-repudiation Accountability Authenticity Compliance
Co-existence Inter-operability Compliance
Modularity Reusability Analyzability Changeability Modification stability Testability Compliance
Adaptability Installability Replaceability Compliance

Figure 1.1 - Software Quality Characteristics from ISO/IEC 25010 with CISQ focal areas highlighted

ISO/IEC 25023 establishes a framework of software quality characteristic measures wherein each quality sub-characteristic consists of a collection of quality attributes that can be quantified as quality measure elements. A quality measure element quantifies a unitary measurable attribute of software, such as the violation of a quality rule. Figure 1.2 presents an example of the ISO/IEC 25023 quality measurement framework using a partial decomposition for the Automated Source Code Reliability Measure.

The non-normative portion of this specification begins by listing the reliability issues that can plague software developed with poor architectural and coding practices. Quality rules written as architectural or coding practices are conventions that avoided the problem described in the reliability issue. These quality rules were then transformed into software quality measure elements by counting violations of these architectural and coding practices and conventions.

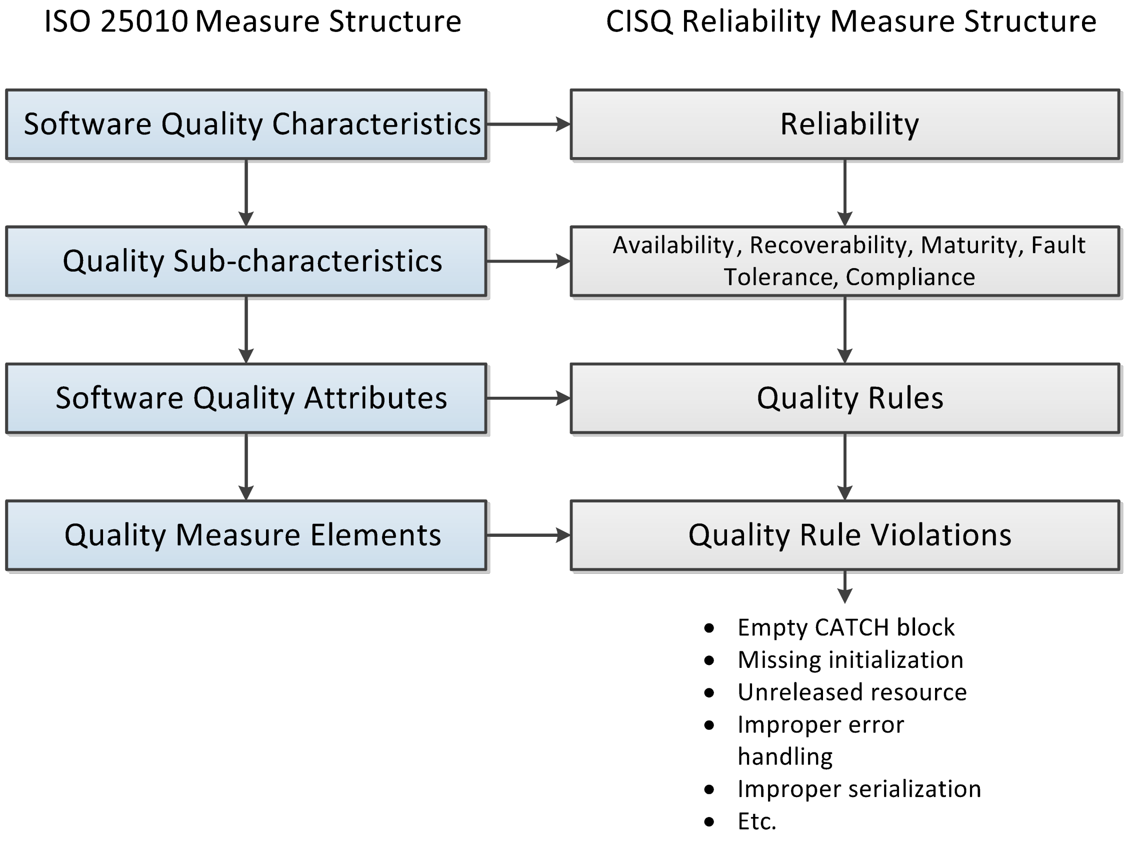
ISO 25010 Measure Structure
CISQ Reliability Measure Structure
Software Quality Characteristics
Reliability
Quality Sub-characteristics
Availability, Recoverability, Maturity, Fault Tolerance, Compliance
Software Quality Attributes
Quality Rules
Quality Measure Elements
Quality Rule Violations
Empty CATCH block
Missing initialization
Unreleased resource
Improper error handling
Improper serialization
Etc.

Figure 1.2 – ISO/IEC 25010 Framework for Software Quality Characteristics Measurement

The normative portion of this specification represents each quality measure element developed from a reliability rule using the Structured Patterns Metamodel Standard (SPMS). The code-based elements in these patterns are represented using the Knowledge Discovery Metamodel (KDM). The calculation of the Automated Source Code Reliability Measure from its quality measure elements is then represented in the Structured Metrics Metamodel (SMM). This calculation is presented as the simple sum of quality measure elements without being adjusted by a weighting scheme.

There are several weighting schemes that can be applied in aggregating violation counts into structural quality measures. The most effective weighting often depends on the measure's use such as assessing operational risk or estimating maintenance costs. The quality measure elements included in this specification were considered to be severe violations of secure architectural and coding practices that would need to be remediated. Therefore, weightings based on severity would add little useful information to the measure since the variance among weights would be small. In order to support benchmarking among applications, this specification includes a measure of the violation density. This measure is created by dividing the total number of violations detected by a count of Automated Function Points (Object Management Group, 2014).

1.6 Using and Improving This Measure

The Automated Source Code Reliability Measure is a correlated measure rather than an absolute measure. That is, since it does not measure all possible reliability-related weaknesses it does not provide an absolute measure of reliability. However, since it includes counts of what industry experts considered high severity reliability weaknesses, it provides a strong indicator of reliability that will be highly correlated with the absolute reliability of a software system and with the probability that it can experience outages, data corruption, and related problems.

Since the impact and frequency of specific violations in the Automated Source Code Reliability Measure could change over time, this approach allows specific violations to be included, excluded, amplified, or diminished over time in order to support the most effective benchmarking, diagnostic, and predictive use. This specification will be adjusted through controlled OMG specification revision processes to reflect changes in reliability engineering while retaining the ability to compare baselines. Vendors of static analysis and measurement technology can compute this standard baseline measure, as well as their own extended measures that include other reliability weaknesses not included as measure elements in this specification.

2 Conformance

2.1 Overview

Implementations of this specification should be able to demonstrate the following attributes in order to claim conformance—automated, objective, transparent, and verifiable.

- ***Automated*** **–** The analysis of the source code and the actual counting must be fully automated. The initial inputs required to prepare the source code for analysis include the source code of the application, the artifacts, and information needed to configure the application for operation, and any available description of the architectural layers in the application.
- ***Objective*** – After the source code has been prepared for analysis using the information provided as inputs, the analysis, calculation, and presentation of results must not require further human intervention. The analysis and calculation must be able to repeatedly produce the same results and outputs on the same body of software.
- ***Transparent*** – Implementations that conform to this specification must clearly list all source code (including versions), non-source code artifacts, and other information used to prepare the source code for submission to the analysis.
- ***Verifiable*** – Compliance with this specification requires that an implementation state the assumptions/heuristics it used with sufficient detail so that the calculations may be independently verified by third parties. In addition, all inputs used are required to be clearly described and itemized so that they can be audited by a third party.

3 References

3.1 Normative References

The following normative documents contain provisions, which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of any of these publications do not apply.

- Structured Patterns Metamodel Standard, formal/2015-10-01
- Knowledge Discovery Metamodel, version 1.3 (KDM), formal/2011-08-04

- Structured Metrics Metamodel, version 1.0 (SMM), formal/2012-01-05
- MOF/XMI Mapping, version 2.4.1 (XMI), formal/2011-08-09
- Automated Function Points (AFP), formal/2014-01-03
- ISO/IEC 25010 Systems and software engineering – System and software product Quality Requirements and Evaluation (SquaRE) – System and software quality models

4 Terms and Definitions

Automated Function Points - a specification for automating the counting of Function Points that mirrors as closely as possible the counting guidelines of the International Function Point User Group. (OMG, formal 2014-01-03)

Common Weakness Enumeration - a repository maintained by MITRE Corporation of known weaknesses in software that can be exploited to gain unauthorized entry into a software system. (cwe.mitre.org)

Cyclomatic Complexity - A measure of control flow complexity developed by Thomas McCabe based on a graph-theoretic analysis that reduces the control flow of a computer program to a set of edges, vertices, and their attributes that can be quantified. (McCabe, 1976)

Internal Software Quality - the degree to which a set of static attributes of a software product satisfy stated and implied needs for the software product to be used under specified conditions. This will be referred to as software structural quality, or simply structural quality in this specification. (ISO/IEC 25010)

Quality Measure Element - a measure defined in terms of a software quality attribute and the measurement method for quantifying it, including optionally the transformation by a mathematical function. (ISO/IEC 25010)

Reliability - the degree to which a system, product, or component performs specified functions under specified conditions for a specified period of time. (ISO/IEC 25010)

Software Product - a set of computer programs, procedures, and possibly associated documentation and data. (ISO/IEC 25010)

Software Product Quality Model - a model that categorizes product quality properties into eight characteristics (functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability). Each characteristic is composed of a set of related sub-characteristics. (ISO/IEC 25010)

Software Quality - degree to which a software product satisfies stated and implied needs when used under specified conditions. (ISO/IEC 25010)

Software Quality Attribute - an inherent property or characteristic of software that can be distinguished quantitatively or qualitatively by human or automated means. (derived from ISO/IEC 25010)

Software Quality Characteristic - a category of software quality attributes that bears on software quality. (ISO/IEC 25010)

Software Quality Characteristic Measure - a software quality measure derived from measuring the attributes related to a specific software quality characteristic.

Software Quality Issue - architectural or coding practices that are known to cause problems in software development, maintenance, or operations and for which software quality rules can be defined that help avoid problems created by the issue.

Software Quality Measure - a measure that is defined as a measurement function of two or more values of software quality measure elements. (ISO/IEC 25010)

Software Quality Measurement - (verb) a set of operations having the object of determining a value of a software quality measure. (ISO/IEC 25010)

Software Quality Model - a defined set of software characteristics, and of relationships between them, which provides a framework for specifying software quality requirements and evaluating the quality of a software product. (derived from ISO/IEC 25010)

Software Quality Property - measurable component of software quality. (derived from ISO/IEC 25010)

Software Quality Rule - an architectural or coding practice or convention that represents good software engineering practice and avoids problems in software development, maintenance, or operations. Violations of these quality rules produces software anti-patterns.

Software Quality Sub-characteristic - a sub-category of a software quality characteristic to which software quality attributes and their software quality measure elements are conceptually related. (derived from ISO/IEC 25010)

Software Reliability - the degree to which the software incorporated into a system, product, or component performs specified functions under specified conditions for a specified period of time. (ISO/IEC 25010

Software Reliability Measure Element - a measure defined in terms of a quality attribute of software that affects it reliability and the measurement method for quantifying it, including optionally the transformation by a mathematical function. (adapted from ISO/IEC 25023)

Structural Quality - the degree to which a set of static attributes of a software product satisfy stated and implied needs for the software product to be used under specified conditions—a component of software quality. This concept is referred to as internal software quality in ISO/IEC 25010

Violation – a pattern or structure in the code that is inconsistent with good architectural and coding practices and can lead to problems in operation or maintenance.

5 Symbols and Abbreviated Terms

- CWE – Common Weakness Enumeration
- CISQ – Consortium for IT Software Quality
- KDM – Knowledge Discovery Metamodel
- SPMS – Structured Patterns Metamodel Standard
- SMM – Structured Metrics Metamodel

6 Additional Information (Informative)

6.1 Software Product Inputs

The following inputs are needed by static code analyzers in order to interpret violations of the software quality rules that would be included in individual software quality measure elements.

- The entire source code for the application being analyzed.
- All materials and information required to prepare the application for production.
- A description of the architecture and layer boundaries of the application, including an assignment of modules to layers.

Static code analyzers will also need a list of the violations that constitute each quality element in the CISQ Automated Source Code Reliability Measure.

6.2 CISQ Automated Source Code Reliability Measure Elements

The violations of good architectural and coding practice incorporated into the CISQ Automated Source Code Reliability Measure are listed and described in Table 6.1. Some of the CWEs from the Common Weakness Enumeration repository that are included in the CISQ Security measure are also defects that can cause reliability problems. In order to retain consistency across measurement specifications, the original CWE numbers and titles have been retained for these reliability measure elements.

In this sub clause and Clause 7 each reliability measure element from Table 6.1 will be labeled as ASCRM-#, where # can be replaced by either a CWE number or a unique REL number assigned to each reliability defect enumerated here. Weaknesses taken from the Common Weakness Enumeration will retain their original CWE number. Weaknesses appearing only in the Reliability list will be labeled with a unique REL number.

Table 6.1 - Reliability Issues, Rules, and Reliability Measure Elements

Reliability Pattern	Consequence	Objective	Measure Element
ASCRM-CWE-120: Buffer Copy without Checking Size of Input	Software featuring known weak coding practices results in unexpected and erroneous behaviors	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	Number of instances in which the content of a buffer can be moved into another buffer whose size is less than that of the sending buffer.
ASCRM-CWE-252-data: Unchecked Return Parameter Value of named Callable and Method Control Element with Read, Write, and Manage Access to Data Resource	Software without consistent and complete handling of errors and exceptions makes it impossible to accurately identify and adequately respond to unusual and unexpected situations	Avoid improper processing of the execution status of data handling operations	Number of instances where the named callable control element or method control element executes a CRUD SQL statement with an action, yet the value of the return parameter from the action is not used by any check control element.
ASCRM-CWE-252-resource: Unchecked Return Parameter Value of named Callable and Method Control Element with Read, Write, and Manage Access to Platform Resource	Software featuring known weak coding practices results in unexpected and erroneous behaviors	Avoid improper processing of the execution status of resource handling operations	Number of instances where the named callable control element or method control element executes a ‘Read,’ ‘Write,’ or ‘Manage Access’ action, yet the value of the return parameter from the action is not used by any check control element.
ASCRM-CWE-396: Declaration of Catch for Generic Exception	Software without consistent and complete handling of errors and exceptions makes it impossible to accurately identify and adequately respond to unusual and unexpected situations	Avoid failure to use dedicated exception types	Number of instances where the named callable control element or method control element contains a catch unit which declares to catch an exception parameter whose data type is part of a list of overly broad exception data types.
ASCRM-CWE-397: Declaration of Throws for Generic Exception	Software without consistent and complete handling of errors and exceptions makes it impossible to accurately identify and adequately respond to unusual and unexpected situations	Avoid failure to use dedicated exception types	Number of instances where the named callable control element or method control element throws an exception parameter whose data type is part of a list of overly broad exception data types.
ASCRM-CWE-456: Storable and Member Data Element Missing Initialization	Software featuring known weak coding practices results in unexpected and erroneous behaviors	Avoid failure to explicitly initialize software data elements in use	Number of instances where a storable data element or member data element is declared by the ‘Create’ action, then is evaluated in a ‘Read’ action without ever being initialized by a ‘Write’ action prior to the evaluation.

ASCRM-CWE-674: Uncontrolled Recursion	Software with uncontrolled recursion risks exceeding resource and capacity limits	Avoid recursion	Number of instances where a named callable control element or method control element initiates an execution path which contains itself
ASCRM-CWE-704: Incorrect Type Conversion or Cast	Software featuring known weak coding practices results in unexpected and erroneous behaviors	Avoid data corruption during incompatible mutation	Number of instances where a storable element or member element is declared with a data type in the 'Create' action, and then is updated with a value which is cast via a type cast action into a second data type, which is incompatible with the first data type.
ASCRM-CWE-772: Missing Release of Resource after Effective Lifetime	Software that is unaware of resource bounds or fails to monitor resources incurs the risk of exceeding resource and capacity limits	Avoid resource hoarding and consequently resource depletion	Number of instances where a platform resource is allocated and assigned a unique resource handler value via a manage resource action, and its unique resource handler value is used throughout the application along a transformation sequence composed of action elements with data relations, some of which are part of named callable and method control elements, but none of which is a resource release statement.
ASCRM-CWE-788: Memory Location Access After End of Buffer	Software that is unaware of resource bounds or fails to monitor resources incurs the risk of exceeding resource and capacity limits	Avoid resource out-of-bound access	Number of instances where none of the callable or method control elements of a transformation sequence perform a range check on a buffer (whose maximum size was defined in a buffer creation action) when a value element that was transformed by the callable or method control element, is used as an index to access a storable or member data element in a buffer 'Read' or 'Write.'
ASCRM-RLB-1: Empty Exception Block	Software without consistent and complete handling of errors and exceptions makes it impossible to accurately identify and adequately respond to unusual and unexpected situations.	Avoid improper responses to unusual and unexpected situations	Number of instances where an exception handling block (such as Catch and Finally blocks) of the named callable and method control elements does not contain any other control element.
ASCRM-RLB-2: Serializable Storable Data Element without Serialization Control Element	Software featuring known weak coding practices results in unexpected and erroneous behaviors	Avoid failure to implement serialization capabilities	Number of instances where the serializable storable element has no serialization control element in its list of control elements (in the case of technologies with class and interface elements, this

			means situations where the serializable storable data element is a class that implements a serializable interface element but does not implement a serialization method element as part of its list composed of method elements). 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: Serializable Storable Data Element with non-Serializable Item Elements	Software featuring known weak coding practices results in unexpected and erroneous behaviors	Avoid incomplete implementation of serialization capabilities	Number of instances where a serializable storable element is composed of a non-serializable item element (in case of technologies with class and interface elements, this means situations where the serializable storable data element is a class that is serializable but owns the element that is a non-serializable member element). The serializable nature of an element is technology dependent; for example, serializable capabilities come from a serializable attribute in .NET and the inheritance from the java.io.Serializable interface in Java.
ASCRM-RLB-4: Persistant Storable Data Element without Proper Comparison Control Element	Software featuring known weak coding practices results in unexpected and erroneous behaviors	Avoid improper comparison capabilities of persistent data	Number of instances where the persistent storable element has no dedicated control element handling comparison action elements from the required comparison control element list. (In case of technologies with class elements, this means situations where a persistent storable data element is a class that is made persistent while it does not implement method elements from the required comparison control element list now composed of method elements: for example, with JAVA, a required comparison control element list is {'hashCode()','equals()'} method elements).

ASCRM-RLB-5: Runtime Resource Management Control Element in a Component Built to Run on Application Servers	Software featuring known weak coding practices results in unexpected and erroneous behaviors	Avoid unproven platform capabilities	Number of instances where the application uses deployed components from application servers, yet uses control elements from the list of low-level resource management API.
ASCRM-RLB-6: Storable or Member Data Element containing Pointer Item Element without Proper Copy Control Element	Software featuring known weak coding practices results in unexpected and erroneous behaviors	Avoid improper copy capabilities when handling data pointers	Number of instances where a storable data element or member data element contains a pointer data element but no dedicated copy operation or copy constructor element.
ASCRM-RLB-7: Class Instance Self Destruction Control Element	Software featuring known weak coding practices results in unexpected and erroneous behaviors	Avoid self-destruction capabilities	Number of instances where a class element can execute a self-destruction control element to destroy itself (an example of a self-destruction control element in C++ is the 'delete this' control).
ASCRM-RLB-8: Named Callable and Method Control Elements with Variadic Parameter Element	Software featuring known weak coding practices results in unexpected and erroneous behaviors	Avoid using variadic parameter elements	Number of instances where the named callable control element or method control element has a variable number of parameters because of the variadic parameter in its signature.
ASCRM-RLB-9: Float Type Storable and Member Data Element Comparison with Equality Operator	Software featuring known weak coding practices results in unexpected and erroneous behaviors	Avoid the use of fault-prone comparison operations between numerical values	Number of instances where the values of storable or member data elements of float type are compared for equality using regular comparison operators in the comparison control element.
ASCRM-RLB-10: Data Access Control Element from Outside Designated Data Manager Component	Software without consistently-enforced approach to data integrity management incurs the risk of behaving unexpectedly	Avoid circumventing dedicated and specialized data manager component(s)	Number of instances where a callable or method control element that is not enumerated on the list of dedicated data access components performs a data action, thus circumventing the intended design for data access.
ASCRM-RLB-11: Named Callable and Method Control Element in Multi-Threaded Context with non-Final Static Storable or Member Element	Software deployed in multi-threaded environments that does not protect its state can experience deadlock or livelock	Avoid unsafe implementations in multi-threaded environments that fail to protect state	Number of instances where a callable control element or method control element owns an unsafe non-final static storable or member data element while it operates in a multi-threaded environment.

ASCRM-RLB-12: Singleton Class Instance Creation without Proper Lock Element Management	Software deployed in multi-threaded environments that does not protect their state can experience deadlock or livelock	Avoid incorrect implementation of singleton patterns caused by improperly-locked instantiations	Number of instances where a singleton class element, (that is, a class element that can be used only once in the 'to' association of a 'Create' action) is instantiated with the 'Creates' action element without any prior locking mechanism activation.
ASCRM-RLB-13: Inter-Module Dependency Cycles	Software deployed in multi-threaded environments that does not protect their state can experience deadlock or livelock	Avoid circular dependencies between modules	Number of instances where a module has references that cycle back to itself via the module callable or data relations cycle (for example, with JAVA this pattern means cycles between packages).
ASCRM-RLB-14: Parent Class Element with References to Child Class Element	Software that does not follow the principles of inheritance and polymorphism results in unexpected behaviors	Avoid parent class references to child class(es)	Number of instances where a parent class element that is used in the 'to' association of an Extends class relation, references the child class element used in the 'from' association of an Extends class relation, directly or indirectly through a parent and child class element, using a callable or data relation (the reference statement is made directly to the child class element or to any one of its own method or member elements).
ASCRM-RLB-15: Class Element with Virtual Method Element wihout Virtual Destructor	Software that does not follow the principles of inheritance and polymorphism results in unexpected behaviors	Avoid failing to include a virtual destructor in a class that includes a virtual method(s)	Number of instances where a class element contains a virtual method element yet does not declare any virtual destructor.
ASCRM-RLB-16: Parent Class Element without Virtual Destructor Method Element	Software that does not follow the principles of inheritance and polymorphism results in unexpected behaviors	Avoid failing to include a virtual destructor in a parent class	Number of instances where, for languages in which custom destructors can be written, the parent class of a child class element via an Extends class relation has no virtual destructor.
ASCRM-RLB-17: Child Class Element without Virtual Destructor unlike its Parent Class Element	Software that does not follow the principles of inheritance and polymorphism results in unexpected behaviors	Avoid failing to include a virtual destructor in a child class despite the existence of a virtual destructor in the parent class	Number of instances where, for languages in which custom destructors can be written, the child class element used in the 'from' association of an Extends class relation does not have its own virtual destructor, while its parent class element that is used in the 'to' association of the Extends class relation has a virtual destructor.

ASCRM-RLB-18: Storable and Member Data Element Initialization with Hard-Coded Network Resource Configuration Data	Software featuring network configuration within its own code incurs the risk of failure when the remote resource change	Avoid the existence of hard-coded values corresponding to network resource identifications	Number of instances where a storable data element or member data element is initialized by a 'Write' action with a hard-coded value corresponding to network resource identifications.
ASCRM-RLB-19: Synchronous Call Time-Out Absence	Software featuring blocking calls to remote systems incurs the risk of own failure when the remote systems fails to process the call correctly	Avoid synchronous remote resource access without handling time-out capabilities	Number of instances where a synchronous call instruction is initiated but the time-out argument is not set or is set to infinite time.

This page intentionally left blank.

7 SPMS Representation of the Quality Measure Elements (Normative)

7.1 Introduction

This clause displays in a human readable format the content of the machine readable XMI format file attached to the current specification.

The content of the machine readable XMI format file is the representations of the CISQ Quality Measure Elements

- according to the Structured Patterns Metamodel Standard (SPMS), and
- relating to the Knowledge Discovery Metamodel (KDM) within their description as frequently as possible, so as to be as generic as possible yet as accurate as possible.

SPMS

More specifically, the machine readable XMI format file attached to the current specification uses the SPMS Definitions Classes:

- PatternDefinition (SPMS:PatternDefinition): the pattern specification. In the context of this document, each CISQ Quality Measure Element is basically the count of occurrences of the described patterns.
- Role (SPMS:Role): "A pattern is informally defined as a set of relationships between a set of entities. Roles describe the set of entities within a pattern, between which those relationships will be described. As such the Role is a required association in a PatternDefinition. {…}. Semantically, a Role is a 'slot' that is required to be fulfilled for an instance of its parent PatternDefinition to exist."
- PatternSection (SPMS:PatternSection): "A PatternSection is a free-form prose textual description of a portion of a PatternDefinition." In the context of this document, there are several different PatternSections in use:
    - "Descriptor" to provide pattern signature, a visible interface of the pattern.
    - "Measure Element" to provide a human readable explanation of the measure.
    - "Description" to provide a human readable explanation of the pattern that is sought after, identifying "Roles" and KDM modeling information.
    - "Objective" to provide a human readable explanation of the intent to get rid of the occurrences of the pattern that is sought after.
    - "Consequence" to provide a human readable explanation of the issue the detection of the pattern is designed to solve.
    - "Input" to provide a human readable of the parameters that are needed to fine-tune the behavior of the pattern detection (e.g., the target application architectural blueprint to comply with).
    - "Comment" to provide some additional information (until now, used to inform about situations where the same measure element is useful for another one of the categories).

As well as some of the SPMS Relationships Classes:

- MemberOf (SPMS:MemberOf): “An InterpatternRelationship specialized to indicate inclusion in a Category.”
- Category (SPMS:Category): “A Category is a simple grouping element for gathering related PatternDefinitions into clusters.” In the context of this document, the SPMS Categories are used to represent the 4 CISQ Quality Characteristics:
    - “CISQ Reliability”
    - “CISQ Security”
    - “CISQ Performance Efficiency”
    - “CISQ Maintainability”

KDM

More specifically, the machine readable XMI format file attached to the current specification uses KDM entities in the “Description” section of the pattern definitions.

Descriptions try to remain as generic yet accurate as possible so that the pattern can be applicable and applied to as many situations as possible: different technologies, different programming languages, etc.

This means:

1 The descriptions include information such as (code:MethodUnit), (action:Reads), (platform:ManagesResource), … to identify the KDM entities the pattern definition involves.

2 The descriptions only detail the salient aspects of the pattern as the specifics can be technology- or language-dependent.

KDM is helpful for reading this clause. However, for readers not familiar with KDM, Table 7.1 presents a primer which translates standard source code element terms into the KDM wording in this specification.

Table 7.1 - Software elements translated into KDM wording

Software element	KDM wording
function, method, procedure, stored procedure, sub-routine etc.	named callable control element (code:CallableUnit with code:CallableKind 'regular,' 'external,' or 'stored') or method control element (code:MethodUnit)
variable, field, member, etc.	storable data element (code:StorableUnit) or member data element (code:MemberUnit)
class	class element (code:StorableUnit with code:DataType code:ClassUnit)
interface	interface element (code:StorableUnit of code:DataType code:InterfaceUnit)
method	method element (code:MethodUnit)
field, member	member element (code:MemberUnit)
SQL stored procedures	stored callable control elements (code:CallableUnit with code:CallableKind 'stored') in a data manager resource (platform:DataManager)
return code value	value (code:Value) of the return parameter (code:ParameterUnit of

	code:ParameterKind 'return')
exception	exception parameter (code:ParameterUnit with code:ParameterKind 'exception')
user input data flow	an external value is entered into the application through the 'ReadsUI' user interface ReadsUI action (ui:ReadsUI), transformed throughout the application along the 'TransformationSequence' sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external,' or 'stored'), and ultimately used as
execution path	execution path (action:BlockUnit composed of action:ActionElements with action:CallableRelations to code:ControlElements)
Libraries, etc.	deployed component (platform:DeployedComponent)
RDBMS	data manager resource (platform:DataManager)
loop body	loop body block (action:BlockUnit starting as the action:TrueFlow of the loop action:GuardedFlow and ending with an action:Flow back to the loop action:GuardedFlow)
loop condition	loop condition (action:BlockUnit used in the action:GuardedFlow)
singleton	class element (code:StorableUnit with code:DataType code:ClassUnit) that can be used only once in the 'to' association of a Create action (action:Creates)
checked	used by a check control element (code:ControlElement containing action:ActionElement with a kind from micro KDM list of comparison actions)

Reading guide

For each numbered sub clause of this clause

- Sub clause 7.2 represents the SPMS Category covered by the current specification.
- Starting with sub clause 7.3 represents a new SPMS PatternDefinition member of this SPMS Category.

SPMS PatternDefinition sub clauses are:

- Pattern category: the “SPMS:Category” category the pattern is related to through a “SPMS:MemberOf” relationship.
- Pattern sections: the list of "SPMS:PatternSection" sections from the pattern:
    - “Descriptor”
    - “Description”
    - “Objective”
    - “Consequence”

  and when applicable,

    - “Input”

- “Comment”

- Pattern roles: the list of “SPMS:Role” roles used in the “Descriptor” and “Description” sub clauses above.

In the following sub-clauses

- Data between square brackets (e.g., [key CISQ_Reliability]) identifies “xmi:id” that are unique and used to reference entities. They are machine-generated to ensure unicity.

- Data between parentheses (e.g., (code:MethodUnit)) identifies KDM modeling information.

- Data between angle brackets (e.g., <ControlElement>) identifies SPMS Roles in Description and Input sub clauses.

7.2 Category definition of CISQ Reliability

[key ASCRM_Reliability] CISQ Reliability

7.3 Pattern definition of ASCRM-CWE-120: Buffer Copy without Checking Size of Input

Pattern Category

[key ASCRM-CWE-120-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-CWE-120-objective]

Avoid buffer operations among buffers with incompatible sizes

Consequence

[key ASCRM-CWE-120-consequence]

Software featuring known weak coding practices results in unexpected and erroneous behaviors

Measure Element

[key ASCRM-CWE-120-measure-element]

Number of instances in which the content of the first buffer is moved into the content of the second buffer while the size of the first buffer is greater than the size of the second buffer.

Description

[key ASCRM-CWE-120-description]

This pattern identifies situations where two buffer storable elements (code:StorableUnit) or member elements (code:MemberUnit) are allocated with specific sizes in <SourceBufferAllocationStatement> and

<TargetBufferAllocationStatement> Create actions (action:Creates), transformed within the application via the <SourceTransformationSequence> and <TargetTransformationSequence> sequences (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), then ultimately used by the application to move the content of the first buffer (action:Reads) onto the content of the second buffer (action:Writes) through the <MoveBufferStatement> statement, while the size of the first buffer is greater than the size of the second buffer.

Descriptor

[key ASCRM-CWE-120-descriptor]

ASCRM-CWE-120(SourceBufferAllocationStatement: sourceBufferAllocationStatement,TargetBufferAllocationStatement: targetBufferAllocationStatement, SourceTransformationSequence: sourceTransformationSequence, TargetTransformationSequence: targetTransformationSequence, MoveBufferStatement: moveBufferStatement)

Variable input

(none applicable)

Comment

[key ASCRM-CWE-120-comment] Measure element contributes to Reliability and Security

List of Roles

[key ASCRM-CWE-120-roles-sourceBufferAllocationStatement] SourceBufferAllocationStatement

[key ASCRM-CWE-120-roles-targetBufferAllocationStatement] TargetBufferAllocationStatement

[key ASCRM-CWE-120-roles-sourceTransformationSequence] SourceTransformationSequence

[key ASCRM-CWE-120-roles-targetTransformationSequence] TargetTransformationSequence

[key ASCRM-CWE-120-roles-moveBufferStatement] MoveBufferStatement

7.4 Pattern definition of ASCRM-CWE-252-data: Unchecked Return Parameter Value of named Callable and Method Control Element with Read, Write, and Manage Access to Data Resource

Pattern Category

[key ASCRM-CWE-252-data-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-CWE-252-data-objective]

Avoid improper processing of the execution status of data handling operations

Consequence

[key ASCRM-CWE-252-data-consequence]

Software without consistent and complete handling of errors and exceptions makes it impossible to accurately identify and adequately respond to unusual and unexpected situations.

Measure Element

[key ASCRM-CWE-252-data-measure-element]

Number of instances where the named callable control element or method control element executes a CRUD SQL statement with an action, yet the value of the return parameter from the action is not used by any check control element.

Description

[key ASCRM-CWE-252-data-description]

This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular,' 'external,' or 'stored') or method control element (code:MethodUnit) executes the <SQLStatement> CRUD SQL statement (data:ReadsColumnSet or data:WritesColumnSet) with the <ExecuteSQLStatement> action yet the value (code:Value) of the return parameter (code:ParameterUnit of code:ParameterKind 'return') from the action is not used by any check control element (code:ControlElement containing action:ActionElement with a kind from micro KDM list of comparison actions).

Descriptor

[key ASCRM-CWE-252-data-descriptor]

ASCRM-CWE-252-data(ControlElement: controlElement,SQLStatement: sQLStatement, ExecuteSQLStatement: executeSQLStatement).

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-CWE-252-data-roles-controlElement] ControlElement

[key ASCRM-CWE-252-data-roles-sQLStatement] SQLStatement

[key ASCRM-CWE-252-data-roles-executeSQLStatement] ExecuteSQLStatement

7.5 Pattern definition of ASCRM-CWE-252-resource: Unchecked Return Parameter Value of named Callable and Method Control Element with Read, Write, and Manage Access to Platform Resource

Pattern Category

[key ASCRM-CWE-252-resource-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-CWE-252-resource-objective]

Avoid improper processing of the execution status of resource handling operations.

Consequence

[key ASCRM-CWE-252-resource-consequence]

Software featuring known weak coding practices results in unexpected and erroneous behaviors.

Measure Element

[key ASCRM-CWE-252-resource-measure-element]

Number of instances where the named callable control element or method control element executes a ‘Read,’ ‘Write,’ or ‘Manage Access’ action, yet the value of the return parameter from the action is not used by any check control element.

Description

[key ASCRM-CWE-252-resource-description]

This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular,' 'external,' or 'stored') or method control element (code:MethodUnit) executes the <ResourceAccessStatement> Read, Write, and Manage Access action (platform:ReadsResource, platform:WritesResource, and platform:ManagesResource) yet the value (code:Value) of the return parameter (code:ParameterUnit of code:ParameterKind 'return') from the action is not used by any check control element (code:ControlElement containing action:ActionElement with a kind from micro KDM list of comparison actions).

Descriptor

[key ASCRM-CWE-252-resource-descriptor]

ASCRM-CWE-252-resource(ControlElement: controlElement,ResourceAccessStatement: resourceAccessStatement).

Variable input

(none applicable)

Comment

[key ASCRM-CWE-252-resource-comment] Measure element contributes to Reliability and Security.

List of Roles

[key ASCRM-CWE-252-resource-roles-controlElement] ControlElement

[key ASCRM-CWE-252-resource-roles-resourceAccessStatement] ResourceAccessStatement

7.6 Pattern definition of ASCRM-CWE-396: Declaration of Catch for Generic Exception

Pattern Category

[key ASCRM-CWE-396-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-CWE-396-objective]

Avoid failure to use dedicated exception types

Consequence

[key ASCRM-CWE-396-consequence]

Software without consistent and complete handling of errors and exceptions makes it impossible to accurately identify and adequately respond to unusual and unexpected situations.

Measure Element

[key ASCRM-CWE-396-measure-element]

Number of instances where the named callable control element or method control element contains a catch unit which declares to catch an exception parameter whose data type is part of a list of overly broad exception data types.

Description

[key ASCRM-CWE-396-description]

This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular,' 'external,' or 'stored') or method control element (code:MethodUnit) contains the <CatchElement> catch unit (action:CatchUnit) which declares to catch the <CaughtExceptionParameter> exception parameter (code:ParameterUnit with code:ParameterKind 'exception') whose datatype (code:DataType) is part of the <OverlyBroadExceptionTypeList> list of overly broad exception datatypes.

As an example, with JAVA, <OverlyBroadExceptionTypeList> is {'java.lang.Exception'}.

Descriptor

[key ASCRM-CWE-396-descriptor]

ASCRM-CWE-396(ControlElement: controlElement,CatchElement: catchElement, CaughtExceptionParameter: caughtExceptionParameter, OverlyBroadExceptionTypeList: overlyBroadExceptionTypeList)

Variable input

[key ASCRM-CWE-396-input]

<OverlyBroadExceptionTypeList> list of overly broad exception datatypes

Comment

[key ASCRM-CWE-396-comment] Measure element contributes to Reliability and Security

List of Roles

[key ASCRM-CWE-396-roles-controlElement] ControlElement

[key ASCRM-CWE-396-roles-catchElement] CatchElement

[key ASCRM-CWE-396-roles-caughtExceptionParameter] CaughtExceptionParameter

[key ASCRM-CWE-396-roles-overlyBroadExceptionTypeList] OverlyBroadExceptionTypeList

7.7 Pattern definition of ASCRM-CWE-397: Declaration of Throws for Generic Exception

Pattern Category

[key ASCRM-CWE-397-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-CWE-397-objective]

Avoid failure to use dedicated exception types

Consequence

[key ASCRM-CWE-397-consequence]

Software without consistent and complete handling of errors and exceptions makes it impossible to accurately identify and adequately respond to unusual and unexpected situations.

Measure Element

[key ASCRM-CWE-397-measure-element]

Number of instances where the named callable control element or method control element throws an exception parameter whose data type is part of a list of overly broad exception data types.

Description

[key ASCRM-CWE-397-description]

This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular', 'external,' or 'stored') or method control element (code:MethodUnit) throws with the <ThrowsAction> Throws action (action:Throws) the <ThrownExceptionParameter> exception parameter (code:ParameterUnit with code:ParameterKind 'exception') whose datatype (code:Datatype) is part of the <OverlyBroadExceptionTypeList> list of overly broad exception datatypes.

As an example, with JAVA, <OverlyBroadExceptionTypeList> is {'java.lang.Exception'}.

Descriptor

[key ASCRM-CWE-397-descriptor]

ASCRM-CWE-397(ControlElement: controlElement,ThrowsAction: throwsAction, ThrownExceptionParameter: thrownExceptionParameter, OverlyBroadExceptionTypeList: overlyBroadExceptionTypeList)

Variable input

[key ASCRM-CWE-397-input]

<OverlyBroadExceptionTypeList> list of overly broad exception datatypes

Comment

[key ASCRM-CWE-397-comment] Measure element contributes to Reliability and Security

List of Roles

[key ASCRM-CWE-397-roles-controlElement] ControlElement

[key ASCRM-CWE-397-roles-throwsAction] ThrowsAction

[key ASCRM-CWE-397-roles-thrownExceptionParameter] ThrownExceptionParameter [key ASCRM-CWE-397-roles-overlyBroadExceptionTypeList] OverlyBroadExceptionTypeList

7.8 Pattern definition of ASCRM-CWE-456: Storable and Member Data Element Missing Initialization

Pattern Category

[key ASCRM-CWE-456-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-CWE-456-objective]

Avoid failure to explicitly initialize software data elements in use

Consequence

[key ASCRM-CWE-456-consequence]

Software featuring known weak coding practices results in unexpected and erroneous behaviors

Measure Element

[key ASCRM-CWE-456-measure-element]

Number of instances where a storable data element or member data element is declared by the ‘Create’ action, then is evaluated in a ‘Read’ action without ever being initialized by a ‘Write’ action prior to the evaluation.

Description

[key ASCRM-CWE-456-description]

This pattern identifies situations where the <DataElement> storable data element (code:StorableUnit) or member data element (code:MemberUnit) is declared by the <DeclarationStatement> Create action (action:Creates), then evaluated in the <EvaluationStatement> Read action (action:Reads) without ever being initialized by a Write action (action:Writes) prior to the evaluation.

Descriptor

[key ASCRM-CWE-456-descriptor]

ASCRM-CWE-456(DataElement: dataElement,DeclarationStatement: declarationStatement, EvaluationStatement: evaluationStatement)

Variable input

(none applicable)

Comment

[key ASCRM-CWE-456-comment] Measure element contributes to Reliability and Security

List of Roles

[key ASCRM-CWE-456-roles-dataElement] DataElement

[key ASCRM-CWE-456-roles-declarationStatement] DeclarationStatement

[key ASCRM-CWE-456-roles-evaluationStatement] EvaluationStatement

7.9 Pattern definition of ASCRM-CWE-674: Uncontrolled Recursion

Pattern Category

[key ASCRM-CWE-674-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-CWE-674-objective]

Avoid infinite recursions

Consequence

[key ASCRM-CWE-674-consequence]

Software that is unaware of recursion incurs the risk of exceeding resource and capacity limits.

Measure Element

[key ASCRM-CWE-674-measure-element]

Number of instances where the named callable control element or method control element starts the execution path which constraints itself.

Description

[key ASCRM-CWE-674-description]

This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular,' 'external,' or 'stored') or method control element (code:MethodUnit) features the <RecursiveExecutionPath> execution path (action:BlockUnit composed of action:ActionElements with action:CallableRelations to code:ControlElements) which contains itself.

Descriptor

[key ASCRM-CWE-674-descriptor]

ASCRM-CWE-674(ControlElement: controlElement,RecursiveExecutionPath: recursiveExecutionPath)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-CWE-674-roles-controlElement] ControlElement

[key ASCRM-CWE-674-roles-recursiveExecutionPath] RecursiveExecutionPath

7.10 Pattern definition of ASCRM-CWE-704: Incorrect Type Conversion or Cast

Pattern Category

[key ASCRM-CWE-704-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-CWE-704-objective]

Avoid data corruption during incompatible mutation

Consequence

[key ASCRM-CWE-704-consequence]

Software featuring known weak coding practices results in unexpected and erroneous behaviors.

Measure Element

[key ASCRM-CWE-704-measure-element]

Number of instances where a storable element or member element is declared with a data type in the ‘Create’ action, and then is updated with a value which is cast via a type cast action into a second data type, which is incompatible with the first data type.

Description

[key ASCRM-CWE-704-description]

This pattern identifies situations where the <DataElement> storable element (code:StorableElement) or member element (code:MemberUnit) is declared with the <DataType> datatype (code:DataType) in the <DataElementDeclarationStatement> Create action (action:Creates), then updated with a value which is cast via the <TypeCastExpression> type cast action (action:ActionElement with micro KDM kind 'TypeCast' or 'DynCast') into the <TargetDataType> second datatype, which is incompatible with the first one.

Descriptor

[key ASCRM-CWE-704-descriptor]

ASCRM-CWE-704(DataElement: dataElement,DataElementDeclarationStatement: dataElementDeclarationStatement, DataType: dataType, TypeCastExpression: typeCastExpression, TargetDataType: targetDataType)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-CWE-704-roles-dataElement] DataElement

[key ASCRM-CWE-704-roles-dataElementDeclarationStatement] DataElementDeclarationStatement

[key ASCRM-CWE-704-roles-dataType] DataType [key ASCRM-CWE-704-roles-typeCastExpression] TypeCastExpression

[key ASCRM-CWE-704-roles-targetDataType] TargetDataType

7.11 Pattern definition of ASCRM-CWE-772: Missing Release of Resource after Effective Lifetime

Pattern Category

[key ASCRM-CWE-772-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-CWE-772-objective]

Avoid resource hoarding and consequently resource depletion

Consequence

[key ASCRM-CWE-772-consequence]

Software that is unaware of resource bounds or fails to monitor resources incurs the risk of exceeding resource and capacity limits.

Measure Element

[key ASCRM-CWE-772-measure-element]

Number of instances where a platform resource is allocated and assigned a unique resource handler value via a manage resource action, and its unique resource handler value is used throughout the application along a transformation sequence composed of action elements with data relations, some of which are part of named callable and method control elements, but none of which is a resource release statement.

Description

[key ASCRM-CWE-772-description]

This pattern identifies situations where the <PlatformResource> platform resource (platform:ResourceType) is allocated and assigned a unique resource handler value via the <ResourceAllocationStatement> ManagesResource action (platform:ManagesResources), its unique resource handler value is used throughout the application, along the <TransformationSequence> sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular,' 'external,' or 'stored'), none of which being a resource release statement (platform:ManagesResource).

Descriptor

[key ASCRM-CWE-772-descriptor]

ASCRM-CWE-772(PlatformResource: platformResource,ResourceAllocationStatement: resourceAllocationStatement, TransformationSequence: transformationSequence)

Variable input

(none applicable)

Comment

[key ASCRM-CWE-772-comment] Measure element contributes to Reliability and Security

List of Roles

[key ASCRM-CWE-772-roles-platformResource] PlatformResource

[key ASCRM-CWE-772-roles-resourceAllocationStatement] ResourceAllocationStatement

[key ASCRM-CWE-772-roles-transformationSequence] TransformationSequence

7.12 Pattern definition of ASCRM-CWE-788: Memory Location Access After End of Buffer

Pattern Category

[key ASCRM-CWE-788-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-CWE-788-objective]

Avoid resource out-of-bound access

Consequence

[key ASCRM-CWE-788-consequence]

Software that is unaware of resource bounds or fails to monitor resources incurs the risk of exceeding resource and capacity limits.

Measure Element

[key ASCRM-CWE-788-measure-element]

Number of instances where a value element is transformed throughout the application along a sequence composed of action elements with data relations, some of which are part of named callable and method control elements, and ultimately used as an index element to access a storable or member data element in a buffer ‘Read’ or ‘Write’ access action; yet none of the callable or method control elements of the transformation sequence perform a range check on the buffer whose maximum size was defined in the buffer creation action.

Description

[key ASCRM-CWE-788-description]

This pattern identifies situations where the <ValueElement> value element (code:Value) is transformed throughout the application along the <TransformationSequence> sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular,' 'external,' or 'stored'), and ultimately used as an index element (code:IndexUnit) to access a storable or member data element (code:StorableUnit or code:MemberUnit) in the <BufferAccessStatement> buffer Read or Write access action (action:Reads, action:Writes, action:Addresses); none of the callable or method control element of the transformation sequence being a range check with regards to the <Buffer> buffer whose maximum size was defined in the <BufferAllocationStatement> buffer creation action (action:Creates).

Descriptor

[key ASCRM-CWE-788-descriptor]

ASCRM-CWE-788(ValueElement: valueElement,Buffer: buffer, BufferReferenceStatement: bufferReferenceStatement, BufferAllocationStatement: bufferAllocationStatement, TransformationSequence: transformationSequence)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-CWE-788-roles-valueElement] ValueElement

[key ASCRM-CWE-788-roles-buffer] Buffer

[key ASCRM-CWE-788-roles-bufferReferenceStatement] BufferReferenceStatement

[key ASCRM-CWE-788-roles-bufferAllocationStatement] BufferAllocationStatement

[key ASCRM-CWE-788-roles-transformationSequence] TransformationSequence

7.13 Pattern definition of ASCRM-RLB-1: Empty Exception Block

Pattern Category

[key ASCRM-RLB-1-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-1-objective]

Avoid improper responses to unusual and unexpected situations

Consequence

[key ASCRM-RLB-1-consequence]

Software without consistent and complete handling of errors and exceptions makes it impossible to accurately identify and adequately respond to unusual and unexpected situations.

Measure Element

[key ASCRM-RLB-1-measure-element]

Number of instances where an exception handling block (such as Catch and Finally blocks) of the named callable and method control elements does not contain any other control element.

Description

[key ASCRM-RLB-1-description]

This pattern identifies situations where the <ExceptionHandlingBlock> exception handling block - such as Catch and Finally blocks - (action:FinallyUnit or action:CatchUnit) of the <ControlElement> named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular,' 'external,' or 'stored') does not contain any other control element (code:ControlElement).

Descriptor

[key ASCRM-RLB-1-descriptor]

ASCRM-RLB-1(ControlElement: controlElement,ExceptionHandlingBlock: exceptionHandlingBlock)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-1-roles-controlElement] ControlElement

[key ASCRM-RLB-1-roles-exceptionHandlingBlock] ExceptionHandlingBlock

7.14 Pattern definition of ASCRM-RLB-2: Serializable Storable Data Element without Serialization Control Element

Pattern Category

[key ASCRM-RLB-2-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-2-objective]

Avoid failure to implement serialization capabilities

Consequence

[key ASCRM-RLB-2-consequence]

Software featuring known weak coding practices results in unexpected and erroneous behaviors.

Measure Element

[key ASCRM-RLB-2-measure-element]

Number of instances where the serializable storable element has no serialization control element in its list of control elements (in the case of technologies with class and interface elements, this means situations where the serializable storable data element is a class that implements a serializable interface element but does not implement a serialization method element as part of its list composed of method elements) (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).

Description

[key ASCRM-RLB-2-description]

This pattern identifies situations where the <SerializableStorableDataElement> serializable storable element (code:StorableUnit) has no serialization control element (code:ControlElement) in its <ControlElementList> list of control elements.

In case of technologies with class and interface elements, this means situations where the <SerializableStorableDataElement> is a class (code:StorableUnit of code:DataType code:ClassUnit) that implements a serializable interface element (code:StorableUnit of code:DataType code:InterfaceUnit) but does not implement a serialization method element (code:MethodUnit) as part of it <ControlElementList> list composed of method elements (code:MethodUnit).

The serializable nature of the element is technology dependent. As examples, serializable nature comes from a serializable SerializableAttribute attribute in .NET and the inheritance from the java.io.Serializable interface in Java.

Descriptor

[key ASCRM-RLB-2-descriptor]

ASCRM-RLB-2(SerializableStorableDataElement: serializableStorableDataElement,ControlElementList: controlElementList)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-2-roles-serializableStorableDataElement] SerializableStorableDataElement

[key ASCRM-RLB-2-roles-controlElementList] ControlElementList

7.15 Pattern definition of ASCRM-RLB-3: Serializable Storable Data Element with non-Serializable Item Elements

Pattern Category

[key ASCRM-RLB-3-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-3-objective]

Avoid incomplete implementation of serialization capabilities

Consequence

[key ASCRM-RLB-3-consequence]

Software featuring known weak coding practices results in unexpected and erroneous behaviors.

Measure Element

[key ASCRM-RLB-3-measure-element]

Number of instances where a serializable storable element is composed of a non-serializable item element (in case of technologies with class and interface elements, this means situations where the serializable storable data element is a class that is serializable but owns the element that is a non-serializable member element) (the serializable nature of an element is technology dependent; for example, serializable capabilities come from a serializable attribute in .NET and the inheritance from the java.io.Serializable interface in Java).

Description

[key ASCRM-RLB-3-description]

This pattern identifies situations where the <SerializableStorableDataElement> serializable storable element (code:StorableElement) is composed of the <NonSerializableItem> non-serializable item element (code:Item).

In case of technologies with class and interface elements, this means situations where the <SerializableStorableDataElement> is a class (code:StorableUnit of code:DataType code:ClassUnit) that is serializable but owns the <NotSerializableItem> that is a non-Serializable member element (code:MemberUnit).

The serializable nature of the element is technology dependent. As examples, serializable nature comes from a serializable SerializableAttribute attribute in .NET and the inheritance from the java.io.Serializable interface in Java.

Descriptor

[key ASCRM-RLB-3-descriptor]

ASCRM-RLB-3(SerializableStorableDataElement: serializableStorableDataElement,NonSerializableItem: nonSerializableItem)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-3-roles-serializableStorableDataElement] SerializableStorableDataElement

[key ASCRM-RLB-3-roles-nonSerializableItem] NonSerializableItem

7.16 Pattern definition of ASCRM-RLB-4: Persistent Storable Data Element without Proper Comparison Control Element

Pattern Category

[key ASCRM-RLB-4-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-4-objective]

Avoid improper comparison capabilities of persistent data

Consequence

[key ASCRM-RLB-4-consequence]

Software featuring known weak coding practices results in unexpected and erroneous behaviors.

Measure Element

[key ASCRM-RLB-4-measure-element]

Number of instances where the persistent storable element has no dedicated control element handling comparison action elements from the required comparison control element list. (in case of technologies with class elements, this means situations where a persistent storable data element is a class that is made persistent while it does not implement method elements from the required comparison control element list now composed of method elements: for example, with JAVA, a required comparison control element list is {'hashCode()','equals()'} method elements).

Description

[key ASCRM-RLB-4-description]

This pattern identifies situations where the <PersistentStorableDataElement> persistent storable element (code:StorableElement with data:ReadsColumnSet and data:WritesColumnSet relations to data:DataContainer) has no dedicated control element (code:ControlElement) aiming at handling comparison action elements (action:ActionElements with 'kind' from micro KDM list of Comparison Actions) from the <RequiredComparisonControlElementList> list.

In case of technologies with class elements, this means situations where the <PersistentStorableDataElement> is a class (code:StorableUnit of code:DataType code:ClassUnit) that is made persistent while it does not implement method elements (code:MethodUnit) from the <RequiredComparisonControlElementList> list now composed of method elements (code:MethodUnit).

As an example, with JAVA, <RequiredComparisonControlElementList> list is {'hashCode()','equals()'} method elements.

Descriptor

[key ASCRM-RLB-4-descriptor]

ASCRM-RLB-4(PersistentStorableDataElement: persistentStorableDataElement,RequiredComparisonControlElementList: requiredComparisonControlElementList)

Variable input

[key ASCRM-RLB-4-input]

<RequiredComparisonControlElementList> list of control elements required for proper comparison.

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-4-roles-persistentStorableDataElement] PersistentStorableDataElement

[key ASCRM-RLB-4-roles-requiredComparisonControlElementList] RequiredComparisonControlElementList

7.17 Pattern definition of ASCRM-RLB-5: Runtime Resource Management Control Element in a Component Built to Run on Application Servers

Pattern Category

[key ASCRM-RLB-5-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-5-objective]

Avoid unproven platform capabilities

Consequence

[key ASCRM-RLB-5-consequence]

Software featuring known weak coding practices results in unexpected and erroneous behaviors.

Measure Element

[key ASCRM-RLB-5-measure-element]

Number of instances where the application uses deployed components from application servers, yet uses control elements from the list of low-level resource management API.

Description

[key ASCRM-RLB-5-description]

This pattern identifies situations where the <Application> application uses deployed component (platform:DeployedComponent) from the <PlatformDeployedComponentList> list, yet uses control elements from the <LowLevelResourceManagementAPIList> list of low-level resource management API (platform:ManagesResource).

Descriptor

[key ASCRM-RLB-5-descriptor]

ASCRM-RLB-5(Application: application,LowLevelResourceManagementAPIList: lowLevelResourceManagementAPIList, PlatformDeployedComponentList: platformDeployedComponentList)

Variable input

[key ASCRM-RLB-5-input]

<PlatformDeployedComponentList> list of application server resource management deployed components

<LowLevelResourceManagementAPIList> list of low-level resource management API

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-5-roles-application] Application

[key ASCRM-RLB-5-roles-lowLevelResourceManagementAPIList] LowLevelResourceManagementAPIList

[key ASCRM-RLB-5-roles-platformDeployedComponentList] PlatformDeployedComponentList

7.18 Pattern definition of ASCRM-RLB-6: Storable or Member Data Element containing Pointer Item Element without Proper Copy Control Element

Pattern Category

[key ASCRM-RLB-6-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-6-objective]

Avoid improper copy capabilities when handling data pointers

Consequence

[key ASCRM-RLB-6-consequence]

Software featuring known weak coding practices results in unexpected and erroneous behaviors.

Measure Element

[key ASCRM-RLB-6-measure-element]

Number of instances where a storable data element or member data element contains a pointer data element but no dedicated copy operation or copy constructor element.

Description

[key ASCRM-RLB-6-description]

This pattern identifies situations where the <DataElement> storable data element (code:StorableUnit) or member data element (code:MemberUnit) contains the <ChildPointerIDataElement> pointer data element (code:DataElement with code:DataType code:PointerType) but has no dedicated copy operation (code:CallableUnit) or copy constructor element (code:MethodUnit with code:MethodKind 'constructor').

Descriptor

[key ASCRM-RLB-6-descriptor]

ASCRM-RLB-6(DataElement: dataElement,ChildPointerDataElement: childPointerDataElement)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-6-roles-dataElement] DataElement

[key ASCRM-RLB-6-roles-childPointerDataElement] ChildPointerDataElement

7.19 Pattern definition of ASCRM-RLB-7: Class Instance Self Destruction Control Element

Pattern Category

[key ASCRM-RLB-7-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-7-objective]

Avoid self-destruction capabilities

Consequence

[key ASCRM-RLB-7-consequence]

Software featuring known weak coding practices results in unexpected and erroneous behaviors.

Measure Element

[key ASCRM-RLB-7-measure-element]

Number of instances where a class element can execute a self-destruction control element to destroy itself (an example of a self-destruction control element in C++ is the 'delete this' control).

Description

[key ASCRM-RLB-7-description]

This pattern identifies situations where the <Class> class element (code:StorableUnit with code:DataType code:ClassUnit) executes the <SelfDestructionControlElement> control element (code:ControlElement) to destroy itself.

As an example of self-destruction control element in C++, the 'delete this' control element.

Descriptor

[key ASCRM-RLB-7-descriptor]

ASCRM-RLB-7(Class: class,SelfDestructionControlElement: selfDestructionControlElement)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-7-roles-class] Class

[key ASCRM-RLB-7-roles-selfDestructionControlElement] SelfDestructionControlElement

7.20 Pattern definition of ASCRM-RLB-8: Named Callable and Method Control Elements with Variadic Parameter Element

Pattern Category

[key ASCRM-RLB-8-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-8-objective]

Avoid using variadic parameter elements

Consequence

[key ASCRM-RLB-8-consequence]

Software featuring known weak coding practices results in unexpected and erroneous behaviors.

Measure Element

[key ASCRM-RLB-8-measure-element]

Number of instances where the named callable control element or method control element has a variable number of parameters because of the variadic parameter in its signature.

Description

[key ASCRM-RLB-8-description]

This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular,' 'external,' or 'stored') or method control element (code:MethodUnit) has a variable number of parameters thanks to the <VariableNumberOfParametersSyntax> variadic parameter (code:ParameterUnit with code:ParameterKind 'variadic') in its signature (code:Signature).

Descriptor

[key ASCRM-RLB-8-descriptor]

ASCRM-RLB-8(ControlElement: controlElement,VariableNumberOfParametersSyntax: variableNumberOfParametersSyntax)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-8-roles-controlElement] ControlElement

[key ASCRM-RLB-8-roles-variableNumberOfParametersSyntax] VariableNumberOfParametersSyntax

7.21 Pattern definition of ASCRM-RLB-9: Float Type Storable and Member Data Element Comparison with Equality Operator

Pattern Category

[key ASCRM-RLB-9-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-9-objective]

Avoid the use of fault-prone comparison operations between numerical values

Consequence

[key ASCRM-RLB-9-consequence]

Software featuring known weak coding practices results in unexpected and erroneous behaviors.

Measure Element

[key ASCRM-RLB-9-measure-element]

Number of instances where the values of storable or member data elements of float type are compared for equality using regular comparison operators in the comparison control element.

Description

[key ASCRM-RLB-9-description]

This pattern identifies situations where the <FloatingValue1> and <FloatingValue2> values (code:Value) of storable or member data element (code:StorableUnit or code:MemberUnit) of float type (code:DataType code:FloatType), are tested for equality with regular comparison operators in the <ComparisonStatement> comparison control element (code:ControlElement containing action:ActionElement with a kind from micro KDM list of comparison actions).

Descriptor

[key ASCRM-RLB-9-descriptor]

ASCRM-RLB-9(FloatingValue1: floatingValue1,FloatingValue2: floatingValue2, ComparisonStatement: comparisonStatement)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-9-roles-floatingValue1] FloatingValue1

[key ASCRM-RLB-9-roles-floatingValue2] FloatingValue2

[key ASCRM-RLB-9-roles-comparisonStatement] ComparisonStatement

7.22 Pattern definition of ASCRM-RLB-10: Data Access Control Element from Outside Designated Data Manager Component

Pattern Category

[key ASCRM-RLB-10-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-10-objective]

Avoid circumventing dedicated and specialized data manager component(s)

Consequence

[key ASCRM-RLB-10-consequence]

Software without consistently-enforced approach to data integrity management incurs the risk of behaving unexpectedly.

Measure Element

[key ASCRM-RLB-10-measure-element]

Number of instances where named callable control element or method control element executes a data action which is not part of a component on the dedicated data access component list thus circumventing the intended design for data access (the unlisted data access component can be either client-side or server-side, which means that data access components can be developed using non-SQL languages).

Description

[key ASCRM-RLB-10-description]

This pattern identifies situations where named callable control element (code:CallableUnit with code:CallableKind 'regular,' 'external,' or 'stored') or method control element (code:MethodUnit) executes the <DataAccessStatement> data action (data:DataActions including data:ReadsColumnSet or data:WritesColumnSet) although it is not part of a component (structure:Component) identified as one of the dedicated data access component from the <DataAccessComponentList> list. The data access component can be either client-side either server-side, which means that data access components can be developed using non-SQL languages. The pattern simply identifies situations where the implementation does not follow the intended design, regardless of the design.

Descriptor

[key ASCRM-RLB-10-descriptor]

ASCRM-RLB-10(ControlElement: controlElement,DataAccessStatement: dataAccessStatement, CentralDataManagerComponent: centralDataManagerComponent, DataAccessComponentList: dataAccessComponentList)

Variable input

[key ASCRM-RLB-10-input]

<DataAccessComponentList> list of components designated to manage data accesses

Comment

[key ASCRM-RLB-10-comment] Measure element contributes to Reliability and Performance Efficiency (as PRF-1)

List of Roles

[key ASCRM-RLB-10-roles-controlElement] ControlElement

[key ASCRM-RLB-10-roles-dataAccessStatement] DataAccessStatement

[key ASCRM-RLB-10-roles-centralDataManagerComponent] CentralDataManagerComponent

[key ASCRM-RLB-10-roles-dataAccessComponentList] DataAccessComponentList

7.23 Pattern definition of ASCRM-RLB-11: Named Callable and Method Control Element in Multi-Thread Context with non-Final Static Storable or Member Element

Pattern Category

[key ASCRM-RLB-11-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-11-objective]

Avoid unsafe implementations in multi-thread environments that fail to protect state

Consequence

[key ASCRM-RLB-11-consequence]

Software deployed in multi-thread environments that does not protect their state can experience deadlock or livelock.

Measure Element

[key ASCRM-RLB-11-measure-element]

Number of instances where a named callable control element or method control element owns an unsafe non-final static storable or member data element while it operates in a multi-threaded environment.

Description

[key ASCRM-RLB-11-description]

This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular,' 'external,' or 'stored') or method control element (code:MethodUnit) owns unsafe <NonFinalStaticField> non-final static storable or member data element (code:StorableUnit or code:MemberUnit with code:StorableKind 'static' and without code:ExportKind 'final') while it operates in a multi-threaded environment (when platform:DeployedResource owns platform:Thread).

Descriptor

[key ASCRM-RLB-11-descriptor]

ASCRM-RLB-11(ControlElement: controlElement,NonFinalStaticField: nonFinalStaticField)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-11-roles-controlElement] ControlElement

[key ASCRM-RLB-11-roles-nonFinalStaticField] NonFinalStaticField

7.24 Pattern definition of ASCRM-RLB-12: Singleton Class Instance Creation without Proper Lock Element Management

Pattern Category

[key ASCRM-RLB-12-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-12-objective]

Avoid incorrect implementation of singleton patterns caused by improperly-locked instantiations

Consequence

[key ASCRM-RLB-12-consequence]

Software deployed in multi-thread environments that does not protect their state can experience deadlock or livelock.

Measure Element

[key ASCRM-RLB-12-measure-element]

Number of instances where a singleton class element, (that is, a class element that can be used only once in the 'to' association of a ‘Create’ action) is instantiated with the ‘Creates’ action element without any prior locking mechanism activation.

Description

[key ASCRM-RLB-12-description]

This pattern identifies situations where the <SingletonClass> singleton class element (code:StorableUnit with code:DataType code:ClassUnit), that is, a class element that can be used only once in the 'to' association of a Create action (action:Creates), is instantiated with the <InstantiationStatement> Creates action element (action:Creates) without any prior locking mechanism activation (platform:PlatformActions with a platform:LockResource).

Descriptor

[key ASCRM-RLB-12-descriptor]

ASCRM-RLB-12(SingletonClass: singletonClass,InstantiationStatement: instantiationStatement)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-12-roles-singletonClass] SingletonClass

[key ASCRM-RLB-12-roles-instantiationStatement] InstantiationStatement

7.25 Pattern definition of ASCRM-RLB-13: Inter-Module Dependency Cycles

Pattern Category

[key ASCRM-RLB-13-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-13-objective]

Avoid circular dependencies between modules

Consequence

[key ASCRM-RLB-13-consequence]

Software deployed in multi-thread environments that does not protect their state can experience deadlock or livelock.

Measure Element

[key ASCRM-RLB-13-measure-element]

Number of instances where a module has references that cycle back to itself via the module callable or data relations cycle (for example, with JAVA this pattern means cycles between packages).

Description

[key ASCRM-RLB-13-description]

This pattern identifies situations where the <Module> module (code:Module) has references that cycle back to itself via the <ModuleDependencyCycle> module callable or data relations cycle (action:BlockUnit composed of action:CallableActions or action:DataActions).

As an example, with JAVA, this pattern means cycles between packages (code:Package).

Descriptor

[key ASCRM-RLB-13-descriptor]

ASCRM-RLB-13(Module: module,ModuleDependencyCycle: moduleDependencyCycle)

Variable input

(none applicable)

Comment

[key ASCRM-RLB-13-comment] Measure element contributes to Reliability and Maintainability

List of Roles

[key ASCRM-RLB-13-roles-module] Module

[key ASCRM-RLB-13-roles-moduleDependencyCycle] ModuleDependencyCycle

7.26 Pattern definition of ASCRM-RLB-14: Parent Class Element with References to Child Class Element

Pattern Category

[key ASCRM-RLB-14-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-14-objective]

Avoid parent class references to child class(es)

Consequence

[key ASCRM-RLB-14-consequence]

Software that does not follow the principles of inheritance and polymorphism results in unexpected behaviors.

Measure Element

[key ASCRM-RLB-14-measure-element]

Number of instances where a parent class element that is used in the 'to' association of an Extends class relation, references the child class element used in the 'from' association of an Extends class relation, directly or indirectly through a parent and child class element, using a callable or data relation (the reference statement is made directly to the child class element or to any one of its own method or member elements).

Description

[key ASCRM-RLB-14-description]

This pattern identifies situations where the <ParentClass> parent class class element (code:StorableUnit of code:DataType code:ClassUnit) that is used in the 'to' association of the Extends class relation, references the <ChildClass> child class element (code:StorableUnit of code:DataType code:ClassUnit) used in the 'from' association of the Extends class relation (code:Extends), directly or indirectly through parent and child class element, with the <ReferenceStatement> callable or data relations (action:CallableRelations or action:DataRelations). The reference statement is made directly to the child class element or to any one of its own method or member elements (code:MethodUnit and code:MemberUnit).

Descriptor

[key ASCRM-RLB-14-descriptor]

ASCRM-RLB-14(ParentClass: parentClass,ChildClass: childClass, ReferenceStatement: referenceStatement)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-14-roles-parentClass] ParentClass

[key ASCRM-RLB-14-roles-childClass] ChildClass

[key ASCRM-RLB-14-roles-referenceStatement] ReferenceStatement

7.27 Pattern definition of ASCRM-RLB-15: Class Element with Virtual Method Element without Virtual Destructor

Pattern Category

[key ASCRM-RLB-15-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-15-objective]

Avoid failing to include a virtual destructor in a class that includes a virtual method(s)

Consequence

[key ASCRM-RLB-15-consequence]

Software that does not follow the principles of inheritance and polymorphism results in unexpected behaviors.

Measure Element

[key ASCRM-RLB-15-measure-element]

Number of instances where a class element contains a virtual method element yet does not declare any virtual destructor.

Description

[key ASCRM-RLB-15-description]

This pattern identifies situations where the <Class> class element (code:StorableUnit of code:DataType code:ClassUnit) contains the <VirtualMethod> virtual method element (code:MethodUnit with code:MethodKind 'virtual') yet without declaring any virtual destructor (code:MethodUnit with code:MethodKind 'virtual' and 'destructor').

Descriptor

[key ASCRM-RLB-15-descriptor]

ASCRM-RLB-15(Class: class,VirtualMethod: virtualMethod)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-15-roles-class] Class

[key ASCRM-RLB-15-roles-virtualMethod] VirtualMethod

7.28 Pattern definition of ASCRM-RLB-16: Parent Class Element without Virtual Destructor Method Element

Pattern Category

[key ASCRM-RLB-16-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-16-objective]

Avoid failing to include a virtual destructor in a parent class

Consequence

[key ASCRM-RLB-16-consequence]

Software that does not follow the principles of inheritance and polymorphism results in unexpected behaviors.

Measure Element

[key ASCRM-RLB-16-measure-element]

Number of instances where, for languages in which custom destructors can be written, the parent class of a child class element via an Extends class relation has no virtual destructor.

Description

[key ASCRM-RLB-16-description]

This pattern identifies situations where, with languages where custom destructors can be written, the <ParentClass> class element (code:StorableUnit of code:DataType code:ClassUnit) parent of the <ChildClass> class element via an Extends class relation (code:Extends) has no virtual destructor (code:MethodUnit with code:MethodKind 'virtual' and 'destructor').

Descriptor

[key ASCRM-RLB-16-descriptor]

ASCRM-RLB-16(ParentClass: parentClass,ChildClass: childClass)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-16-roles-parentClass] ParentClass

[key ASCRM-RLB-16-roles-childClass] ChildClass

7.29 Pattern definition of ASCRM-RLB-17: Child Class Element without Virtual Destructor unlike its Parent Class Element

Pattern Category

[key ASCRM-RLB-17-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-17-objective]

Avoid failing to include a virtual destructor in a child class despite the existence of a virtual destructor in the parent class.

Consequence

[key ASCRM-RLB-17-consequence]

Software that does not follow the principles of inheritance and polymorphism results in unexpected behaviors.

Measure Element

[key ASCRM-RLB-17-measure-element]

Number of instances where, for languages in which custom destructors can be written, the child class element used in the 'from' association of an Extends class relation does not have its own virtual destructor, while its parent class element that is used in the 'to' association of the Extends class relation has a virtual destructor.

Description

[key ASCRM-RLB-17-description]

This pattern identifies situations where, with languages where custom destructors can be written, the <ChildClass> class element (code:StorableUnit of code:DataType code:ClassUnit) used in the 'from' association of an Extends class relation (code:Extends) whose <ParentClass> parent class class element (code:StorableUnit of code:DataType code:ClassUnit) that is used in the 'to' association of the Extends class relation, directly or indirectly through parent and child class element, has the <ParentVirtualDestructor> virtual destructor (code:MethodUnit with code:MethodKind 'virtual' and 'destructor'), that lack its own virtual destructor (code:MethodUnit with code:MethodKind 'virtual' and 'destructor').

Descriptor

[key ASCRM-RLB-17-descriptor]

ASCRM-RLB-17(ParentClass: parentClass,ChildClass: childClass, ParentVirtualDestructor: parentVirtualDestructor)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-17-roles-parentClass] ParentClass

[key ASCRM-RLB-17-roles-childClass] ChildClass

[key ASCRM-RLB-17-roles-parentVirtualDestructor] ParentVirtualDestructor

7.30 Pattern definition of ASCRM-RLB-18: Storable and Member Data Element Initialization with Hard-Coded Network Resource Configuration Data

Pattern Category

[key ASCRM-RLB-18-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-18-objective]

Avoid the existence of hard-coded network configuration elements.

Consequence

[key ASCRM-RLB-18-consequence]

Software featuring network configuration within its own code incurs the risk of failure when the remote resource change.

Measure Element

[key ASCRM-RLB-18-measure-element]

Number of instances where a storable data element or member data element is initialized by a ‘Write’ action with a hard-coded value corresponding to network resource identifications.

Description

[key ASCRM-RLB-18-description]

This pattern identifies situations where the <DataElement> storable data element (code:StorableUnit) or member data element (code:MemberUnit) is initialized by the <InitializationStatement> Write action (action:Writes) with the <NetworkResourceIdentificationValue> hard-coded value (code:Value) corresponding to network resource identifications (platform:MarshalledResource or platform:MessagingResource or platform:Machine).

Descriptor

[key ASCRM-RLB-18-descriptor]

ASCRM-RLB-18(DataElement: dataElement,InitializationStatement: initializationStatement, NetworkResourceIdentificationValue: networkResourceIdentificationValue)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-18-roles-dataElement] DataElement

[key ASCRM-RLB-18-roles-initializationStatement] InitializationStatement

[key ASCRM-RLB-18-roles-networkResourceIdentificationValue] NetworkResourceIdentificationValue

7.31 Pattern definition of ASCRM-RLB-19: Synchronous Call Time-Out Absence

Pattern Category

[key ASCRM-RLB-19-relatedPatts-reliability] ASCRM_Reliability

Pattern Sections

Objective

[key ASCRM-RLB-19-objective]

Avoid synchronous remote resource access without handling time-out capabilities.

Consequence

[key ASCRM-RLB-19-consequence]

Software featuring blocking calls to remote systems incurs the risk of own failure when the remote systems fails to process the call correctly.

Measure Element

[key ASCRM-RLB-19-measure-element]

Number of instances where a synchronous call instruction is initiated but the time-out argument is not set or is set to infinite time.

Description

[key ASCRM-RLB-19-description]

This pattern identifies situations where the <SynchronousCallInstruction> synchronous call instruction is initiated but the <TimeOutArgument> time-out argument is not set or set to infinite time.

Descriptor

[key ASCRM-RLB-19-descriptor]

ASCRM-RLB-19(SynchronousCallInstruction: synchronousCallInstruction,TimeOutArgumentValue: timeOutArgumentValue)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCRM-RLB-19-roles-synchronousCallInstruction] SynchronousCallInstruction

[key ASCRM-RLB-19-roles-timeOutArgumentValue] TimeOutArgumentValue

8 Calculation of Reliability and Functional Density Measures (Normative)

8.1 Calculation of the Base Measures

A count of total violations of quality rules was selected as the best alternative for measurement. Software quality measures have frequently been scored at the component level and then aggregated to develop an overall score for the application. However, scoring at the component level was rejected because many critical violations of reliability quality rules cannot be isolated to a single component, but rather involve interactions among several components. Therefore, the Automated Source Code Reliability Measure is computed as the sum of its 29 quality measure elements computed across the entire application.

The calculation of the Automated Source Code Reliability Measure begins with determining the value of each of the 29 reliability measure elements. Each reliability measure element is measured as the total number of violations of its associated quality rule that are detected through automated analysis. Thus the value of each of the 29 reliability measure elements is represented as CISQ-RelME$_i$ where the range for i runs from 1 to 29.

$$\text{CISQ-RelME}_i = \Sigma\ \text{(all violations of type CISQ-RelME}_i\text{ detected through automated analysis)}$$

The value of the un-weighted and un-normalized Automated Source Code Reliability Measure (CISQ-Rel) is the sum of the values of the 29 reliability measure elements.

$$\text{CISQ-Rel} = \sum_{i=1}^{22} \text{CISQ-RelME}_i$$

Higher values of CISQ-Rel indicate a larger number of reliability-related defects in the application.

8.2 Functional Density of Reliability Violations

In order to better compare reliability results among different applications, the Automated Source Code Reliability Measure can be normalized by size to create a density measure. There are several size measures with which the density of reliability violations can be normalized, such as lines of code and function points. These size measures, if properly standardized, can be used for creating a density measure for use in benchmarking applications. However, the OMG Automated Function Points measure offers an automatable size measure that, as an OMG Supported Specification, is standardized, adapted from the International Function Point User Group's (IFPUG) counting guidelines, and commercially supported. Although other size measures can be legitimately used to evaluate the density of reliability violations, the following density measure for reliability violations is derived from OMG supported specifications for Automated Function Points and the Automated Source Code Reliability Measure. Thus, the functional density of Reliability violations is a simple division expressed as follows.

$$\text{CISQ-Rel-density} = \frac{\text{CISQ-Rel}}{\text{AFP}}$$

This page intentionally left blank.

9 Alternative Weighted Measures and Uses (Informative)

9.1 Additional Derived Measures

There are many additional weighting schemes that can be applied to the Automated Source Code Reliability Measure or to the reliability measure elements that compose it. Table 9.1 presents several candidate weighted measures and their potential uses. However, these weighting schemes are not derived from any existing standards and are therefore not normative.

Table 9.1 - Informative Weighting Schemes for Reliability Measurement

Weighting scheme	**Potential uses**
Weight each Reliability measure by its severity	Measuring risk of reliability problems such as outages, slow recovery times, or data corruption.
Weight each Reliability measure element by its effort to fix	Measuring cost of ownership, estimative future corrective maintenance effort and costs.
Weight each module or application component by its density of Reliability violations	Prioritizing modules or application components for corrective maintenance or replacement.

This page intentionally left blank.

10 References (Informative)

Consortium for IT Software Quality (2010). http://www.it-cisq.org

Curtis, B. (1980). Measurement and experimentation in software engineering. *Proceedings of the IEEE*, 68 (9), 1103-1119.

International Organization for Standards. ISO/IEC 25010 Systems and software engineering – System and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models

International Organization for Standards (2012). *ISO/IEC 25023 (in development) Systems and software engineering: Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of system and software product quality.*

International Organization for Standards (2012). *ISO/IEC TR 9126-3:2003, Software engineering — Product quality — Part 3: Internal metrics.*

McCabe, T.J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, *2* (4), 308–320.

Object Management Group (2014). Automated Function Points. formal 2014-01-03 http://www.omg.org/spec/AFP/.

This page intentionally left blank.

Annex A: CISQ

(informative)

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 1.1 displays the ISO/IEC 25010 software product quality model with the four software quality characteristics selected for automation by CISQ highlighted in blue. Each software quality characteristic is shown with the sub-characteristics that compose it.

This page intentionally left blank.