Date: ~~Nov~~Dec~~ember~~ 7~~12~~, 2018

Automated Source Code Quality Measures

Request for Comments

OMG Document Number: admtf/2018-1~~2~~1-0~~2~~7
Standard document URL:
http://www.omg.org/spec/ASCQM/2018121~~1~~021~~1~~/AutomatedSourceCodeQualityMeasures

Machine consumable files:

http://www.omg.org/spec/ASCQM/20181102/AutomatedSourceCodeQualityMeasures.xmi

---

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group (OMG) specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Consortium for IT Software Quality and its parent, Object Management Group, Inc. (OMG), a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of 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. CISQ 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 CISDQ, 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 252.287-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.287-19 or as specified in 48 C.F.R. 287-7202-2 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, 250 First Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

IMM®, MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , MOF™ , OMG Interface Definition Language (IDL)™ , and OMG SysML™ are trademarks of the Object Management Group. 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 Procedure

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

About the Object Management Group

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 Meta-model); 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 Formal Specifications are available from this URL: *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

CORBA Embedded Intelligence Specifications

CORBA Security Specifications

OMG Domain 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
Suite 300
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/IEC standards. Please consult http://www.iso.org

Issues

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

0. Submission-Specific Material

0.1 Submission Preface

This submission is of a measure represented in compliance with OMG's Knowledge Discovery Metamodel (KDM), Structured Patterns Metamodel for Software (SPMS), and Structured Metrics Meta-Model (SMM). However, its submission is independent of KDM, SPMS, and SMM to establish it as a supported specification in its own right. This specification for four Structural Quality Measures builds on elements already developed in OMG's Automated Source Code Measures for Reliability, Security, Performance Efficiency, and Maintainability Measure standards. The measures described in this specification are an important component for achieving the mission of the Architecture Driven Modernization Task Force by qualifying the structural quality of modernized software and its architecture.

0.2 Copyright Waiver

IPR Mode: Non-Assertion Covenant

0.3 Submitter Representative

Bill Curtis
CAST Software, Inc.
b.curtis@castsoftware.com

Robert Martin
MITRE Corporation
ramartin@mitre.org

0.4 Author Team

Bill Curtis,
CAST Software, Inc.
b.curtis@castsoftware.com

Robert Martin
MITRE Corporation
ramartin@mitre.org

Paul Seay
Northrup Grumman
Paul.seay@ngc.com

Paul Beaudoin
Northrup Grumman
paul.beaudoin@ngc.com

Paul Rainey
CGI
paul.rainey@cgi.com

Kevin Doyle
CGI
kevin.m.doyle@cgifederal.com

Gordon Uchenick
Synopsys
Gordon Uchenick@synopsys.com

Joe Jarzombek
Synopsys
Joe.Jarzombek@synopsys.com

Philippe-Emmanuel Douziech
CAST
p.douziech@castsoftware.com

Guillaume Ragar
CAST
g.ragar@castsoftware.com

William Dickenson
CAST Software, Inc.
w.dickenson@castsoftware.com

Hariharan Mathrubutham
Cognizant
Hariharan.Mathrubutham@cognizant.com

Girish Seshagiri
ISHPI
girish.seshagiri@ishpi.net

Dan Plakosh
Software Engineering Institute - CERT
dplakosh@cert.org

S. Amitabh
Tech Mahindra
samitabh@TechMahindra.com

Sanjeev Chikodi
Tech Mahindra
sanjeevc@TechMahindra.com

0.5 Proof of Concept

Coverity and CAST among other static analysis vendors have implemented versions of these measures based on the set of weaknesses their technologies detect. Currently there are no industry-wide standards for which weaknesses to include in structural quality measures or how such measures should be calculated. Consequently, each vendor produces a unique version of these structural quality measures.

1. Scope

1.1 Purpose

This specification updates, expands, and combines four previous adopted specifications of the OMG:

- ~~formal 16-01-01~~ Automated Source Code Maintainability Measure (ASCMM) https://www.omg.org/spec/ASCMM/1.0/
- ~~formal 16-01-02~~ Automated Source Code Performance Efficiency Measure (ASCPEM) https://www.omg.org/spec/ASCPEM/1.0/
- ~~formal 16-01-03~~ Automated Source Code Reliability Measure (ASCRM) https://www.omg.org/spec/ASCRM/1.0/
- ~~formal 16-01-04~~ Automated Source Code Security Measure (ASCSM) https://www.omg.org/spec/ASCSM/1.0/

The measures in these standards were calculated from detecting and counting violations of good architectural and coding practices in the source code that could result in unacceptable operational risks or excessive costs. Establishing standards for these measures at the source code level is important because they have been used in outsourcing and system development contracts without having international standards to reference. For instance, the ISO/IEC 25000 series of standards that govern software product quality do not provide measures at the source code level.

A primary objective of updating these measures was to extend their applicability to embedded software, which is especially important for the growing implementation of embedded devices and the Internet of Things. Functionality that has traditionally been implemented in IT applications is now being moved to embedded chips. Since the weaknesses included in the measures specified in this document have been found to be applicable to all forms of software, embedded software is not treated specially in this specification.~~Consequently, all but a few of the weaknesses in these measures have been found applicable to both domains of software. Therefore, we do not segment the measures by software domain.~~

1.2 Overview of Structural Quality Measurement in Software

Measurement of the structural quality characteristics of software has a long history in software engineering (Curtis, 1980). These characteristics are also referred to as the structural, internal, technical, or engineering characteristics of software source code. Software quality characteristics are increasingly incorporated into development and outsourcing contracts as the equivalent of service level agreements. That is, target thresholds based on structural quality measures are being written into contracts as acceptance criteria for delivered software. Currently there are no standards for most of the software structural quality measures 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 structural quality characteristics in each contract. This specification addresses one aspect of this problem by providing a specification for measuring four structural quality characteristics from the source code—Reliability, Security, Performance Efficiency, and Maintainability.

Recent advances in measuring the structural quality of software involve detecting violations of good architectural and coding practice from statically analyzing source code. Violations of good architectural and design practice can also be detected from statically analyzing design specifications written in a design language with a formal syntax and semantics. Good architectural and coding practices can be stated as rules for engineering software products. Violations of these rules will be called weaknesses in this specification to be consistent with terms used in the Common Weakness Enumeration (Martin & Barnum, 2006) which lists many of the weaknesses used in several of these measures.

The Automated Source Code Quality Measures are correlated measures rather than absolute measures. That is, since they do not measure all possible weaknesses in each of the four areas, they do not provide absolute measures. However, since they include counts of what industry experts have determined to be most severe weaknesses, they provide strong indicators of the quality of a software system in each area. In most instances they will be highly correlated with the probability of operational or cost problems related to each measure's area.

Recent research in analyzing structural quality weaknesses has identified common patterns of code structures that can be used to detect weaknesses. Many of these 'Detection Patterns' are shared across different weaknesses. Detection Patterns will be used in this specification to organize and simplify the presentation of weaknesses underlying the four structural quality measures. Each weakness will be described as a quality measure element to remain consistent with ISO/IEC 25020. Each quality measure element will be represented as one or more Detection Patterns. Many quality measure elements (weaknesses) will share one of more Detection Patterns in common.

The normative portion of this specification represents each quality attribute (weakness) and quality measure element (detection pattern) 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 each of the four Automated Source Code Quality Measures from their quality measure elements is then represented in the Structured Metrics Metamodel (SMM). This calculation is developed by counting the number of detection patterns for each weakness, and then summing these numbers for all the weaknesses included in the specific quality characteristic measure.

~~Using violations of good architectural and coding practices in structural quality measures presents several challenges for establishing baselines. Growth in the number of unique rules that can be violated continually raises the bar for measuring structural quality, reducing the validity of baseline comparisons. Further, different vendors use different algorithms that detect different weaknesses or different instantiations of the same weakness, making comparisons difficult across commercial static analysis tools. One solution to this problem is to create a stable list of weaknesses that are used for computing a baseline for each structural quality characteristic. Baselines will be most accurate when compared~~

against analyses from the same tool. However, the measures in this specification provide a starting point for industry benchmarks.

1.3 Development of the Automated Source Code Quality Measures

The Consortium for IT Software Quality (CISQ), a consortium managed by OMG, was formed in 2010 to create international standards for automating measures of size and structural quality characteristics from source code. These measures are intended for use by IT organizations, IT service providers, and software vendors in contracting, developing, testing, accepting, and deploying software systems. Executives from the member companies that joined CISQ prioritized Reliability, Security, Performance Efficiency, and Maintainability as the initial structural quality measures to be specified.

An international team of experts drawn from CISQ's 24 original companies formed into working groups to define CISQ measures. Weaknesses that had a high probability of causing reliability, security, performance efficiency, or maintainability problems were selected for inclusion in the four measures. The original CISQ members 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 weaknesses. This list was pared down to a set of weaknesses they believed had to be remediated to avoid serious operational or cost problems. These 86 weaknesses became the foundation of the original specifications of the automated source code measures for Reliability, Security, Performance Efficiency, and Maintainability.

The work groups began their work on developing specifications for reliability, security, performance, and maintainability measures by identifying serious problems in each area and quality rules for avoiding them. The measures were then developed from counting violations of these rules, i.e., counting weaknesses. They developed lists of candidate issues and quality rules by drawing information from company defect logs, their career experience in different environments, and industry sources such as books, developer-focused discussion sites, and blogs. To reduce the work group's initial list to a critical set of weaknesses for each quality characteristic, 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 groups 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 the measurement specifications.

CISQ decided to base its Security measure on exploitable weaknesses contained in the Common Weakness Enumeration (CWE) repository maintained by MITRE Corporation. CWE is a cyber-security community repository of over 800 known weaknesses in software that can be exploited for unauthorized intrusion into a system. The original CISQ Security specification was developed from 22 of the CWE/SANS Institute Top 25 Most Dangerous Software Errors, a list of the most widespread and commonly exploited security weaknesses. This revision of the CISQ Security measure includes additional weaknesses from the CWE repository that have been judged to create unacceptable security risks.

~~The original CISQ structural quality measures focused on weaknesses primarily found in IT business applications. In 2017 the CISQ governing board directed CISQ to extend the structural quality measures to include weaknesses related to embedded systems. This specification was developed by a working group formed in 2018 from CISQ sponsors as well as the Software Engineering Institute and MITRE to define additional weaknesses that satisfied one of two criteria:~~

1. ~~a severe weakness specific to embedded systems, or~~
2. ~~a severe weakness applicable to all systems that had not been included in the earlier measures.~~

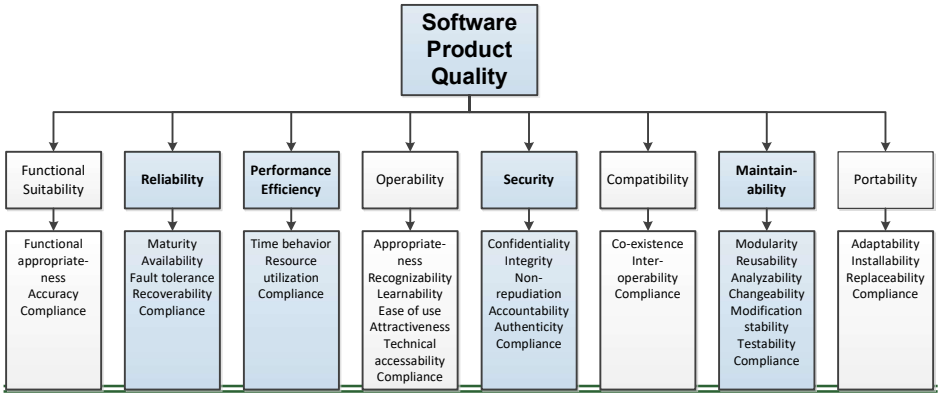
~~With the trend to embed greater functionality on chips, the embedded extensions working group found fewer weaknesses that were unique to embedded systems, except those related to timing in real-time applications. With few exceptions, weaknesses in the existing CISQ measures were applicable to many embedded systems.~~

~~The Automated Source Code Quality Measures are correlated measures rather than absolute measures. That is, since they do not measure all possible weaknesses in each of the four areas, they do not provide absolute measures. However, since they include counts of what industry experts have determined to be most severe weaknesses, they provide strong indicators of the quality of a software system in each area. In most instances they will be highly correlated with the probability of operational or cost problems related to each measure's area.~~

1.4 ~~Relationship of the CISQ Structural Quality Measures to ISO Standards~~

~~**ISO/IEC 25010 defines the product quality model for software-intensive systems (Figure 1). This model is composed of 8 quality characteristics, four of which are the subject of CISQ structural quality measures (indicated in blue). Each of ISO/IEC 25010's eight quality characteristics consists of several quality sub-characteristics that define the domain of issues covered by their parent quality characteristic. CISQ structural quality measures conform to the definitions in ISO/IEC 25010. The sub-characteristics of each quality characteristic were used to ensure the CISQ measures covered the domain of issues in each of the four areas. ISO/IEC 25010 is currently undergoing revision with CISQ participation. The CISQ measures will conform with definitions in the revised ISO/IEC 25010-2 when published.**~~

Formatted: Heading 2



~~Figure 1. Software Quality Characteristics from ISO/IEC 25010 with CISQ measure 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 2 presents an example of the ISO/IEC 25023 quality measurement framework using a partial decomposition for the Automated Source Code Security Measure.~~

~~1.5 Organization of the Specification~~

In Clause 6 of this specification lists weaknesses grouped by quality characteristic that correspond to ISO/IEC 25020's quality attributes. A weakness is detected by identifying patterns of code elements in the software (called detection patterns) that instantiate the weakness. Each detection pattern equates to a quality measure element used in calculating the CISQ quality measures. In Clause 7, quality attributes (weaknesses) are transformed into the KDM and SPMS-based detection patterns that represent them. The CISQ quality measures are then calculated by detecting and counting occurrences of detection patterns, each of which indicates the existence of a weakness in the software. These calculations are represented in the Structured Metrics Metamodel (SMM).

Figure 2 displays the hierarchical relationships indicating how CISQ conforms to the reference measurement structure established in ISO/IEC 25020 that governs software quality measures in ISO/IEC 25023. This structure is presented using the CISQ Security measure as an example. The CISQ measures only use ISO's quality subcharacteristics for ensuring that the CISQ weaknesses covered the measurable domain of an ISO quality characteristic as defined in ISO/IEC 25010. CISQ's weaknesses (CWEs) correspond to ISO's quality attributes. CISQ weaknesses are represented as one or more detection patterns among structural code elements in the software. Variations in how a weakness may be instantiated are represented by its association with several different detection patterns. Each occurrence of a detection pattern represents an occurrence of a weakness in the software. Occurrences of these detection patterns in the software correspond to ISO's quality measure elements and are the elements calculated in the CISQ measures.

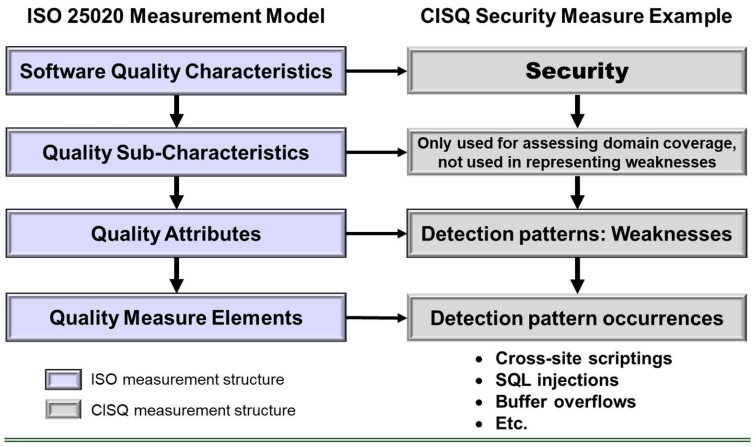


Figure 2. ISO/IEC 25010 Framework for Software Quality Characteristics Measurement

The normative portion of this specification represents each quality attribute (weakness) and quality measure element (detection pattern) 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 each of the four Automated Source Code Quality Measures from their quality measure elements is then represented in the Structured Metrics Metamodel (SMM). This calculation is developed by counting the number of detection patterns for each weakness, and then summing these numbers for all the weaknesses included in the specific quality characteristic measure.

Several weighting schemes 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 severe enough violations of quality rules that they should be remediated. Consequently, weightings based on severity would add little useful information to the measures 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 These Measures

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

Since the impact and frequency of specific violations in the Automated Source Code Security 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 security 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 security weaknesses not included as measure elements in this specification.

2. Conformance

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 counting of weaknesses 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 uses 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. Normative References

~~3.1 Normative~~

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, https://www.omg.org/spec/SPMS/1.2/ ~~formal/17-11-01~~
- Knowledge Discovery Metamodel, version 1.3 (KDM), https://www.omg.org/spec/KDM/1.4/ ~~formal/2011-08-04~~
- Structured Metrics Metamodel, version 1.0 (SMM), formal/2012-01-05
- MOF/XMI Mapping, version 2.4.1 (XMI), https://www.omg.org/spec/XMI/2.5.1/ ~~formal/2011-08-09~~
- Automated Function Points (AFP), https://www.omg.org/spec/AFP/1.0/ ~~formal/2014-01-03~~
- Automated Source Code Reliability Measure, version 1.0 (ASCRM), https://www.omg.org/spec/ASCRM/1.0/ ~~formal/2016-01-03~~
- Automated Source Code Security Measure, version 1.0 (ASCSM), https://www.omg.org/spec/ASCSM/1.0/ ~~formal/2016-01-04~~

- Automated Source Code Performance Efficiency Measure, version 1.0 (ASCPEM), https://www.omg.org/spec/ASCPEM/1.0/ ~~formal/2016-01-02~~
- Automated Source Code Maintainability Measure, version 1.0 (ASCMM), https://www.omg.org/spec/ASCMM/1.0/ ~~formal/2016-01-01~~
- ISO/IEC 25010 Systems and software engineering – System and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models
- ISO/IEC 25020:2007 Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Measurement reference model and guide

4. Terms and Definitions

For the purposes of this specification, the following terms and definitions apply.

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)

Contributing Weakness—a weakness that is represented as a child of a parent weakness in the Common Weakness Enumeration, that is, a variant instantiation of the parent weakness (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)

Detection Pattern—a collection of parsed program elements and their relations that constitute a weakness in the software.

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)

Maintainability—capability of a product to be modified by the intended maintainers with effectiveness and efficiency (ISO/IEC 25010)

Parent Weakness—a weakness in the Common Weakness Enumeration that has numerous possible instantiations in software that are represented by its relation to child CWEs (cwe.mitre.org)

Performance Efficiency—capability of a product to use an appropriate amount of resources under stated conditions (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—capability a product, to perform specified functions under specified conditions for a specified period of time (ISO/IEC 25010)

Security— capability of a product to protect information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization, and to defend against attack patterns by malicious actors (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 set of software quality attributes that affect a specific category of software quality outcomes. (similar to but more specific than 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 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 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)

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 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)

Structural Element—a component of software code that can be uniquely identified and counted such as a token, decision, variable, etc.

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.

Weakness— sometimes referred to as a software anti-pattern, is a pattern or structure in the code (Detection Pattern) that is inconsistent with good architectural or coding practice, violates a software quality rule, and can lead to operational or cost problems.

5. Symbols (and Abbreviated Terms)

AFP — Automated Function Points
ASCMM — Automated Source Code Maintainability Measure
ASCPEM — Automated Source Code Performance Efficiency Measure
ASCQM — Automated Source Code Quality Measure
ASCRM — Automated Source Code Reliability Measure
ASCSM — Automated Source Code Security Measure
CWE — Common Weakness Enumeration
CISQ — Consortium for IT Software Quality
KDM — Knowledge Discovery Metamodel
SPMS — Structured Pattern 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 list of vetted libraries that are being used to sanitize data against potential attacks~~"neutralize" input data~~
- What routines/API calls are being used for remote authentication, to any custom initialization and cleanup routines, to synchronize resources, or to neutralize accepted file types or the names of resources

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

6.2 Automated Source Code Quality Measure Elements

The weaknesses violating software quality rules that compose the CISQ Automated Source Code Quality Measures are grouped by measure in the clauses 6 and 7. Some of the weaknesses are included in more than one quality measure because they can cause several types of problems. The Common Weakness Enumeration repository (CWE, Appendix B) has recently been expanded to include weaknesses from quality characteristics beyond security. All weaknesses included in these measures are identified by their CWE number from the repository. ~~All weaknesses included in these measures are identified by their CWE number from the Common Weakness Enumeration repository.~~ In most cases the description of CWEs is taken from information in the online repository (cwe.mitre.org). The mappings of the weaknesses from the previous CISQ measures to the current measures are presented in Appendix C.

Some weaknesses drawn from the CWE repository (parent weaknesses) have related weaknesses listed as 'contributing weaknesses' ('children' in the CWE). Contributing weaknesses represent variants of how the parent weakness can be instantiated in software. In the following tables the cells containing CWE IDs for parents are presented in a darker blue than the cells containing contributing weaknesses. Based on their severity, not all children were included. Compliance to the CISQ measures is assessed at the level of the parent weakness. A technology must be able to detect at least one of the contributing weaknesses to be assessed compliant on the parent weakness.

6.3 Automated Source Code Maintainability Measure Element Descriptions

The quality measure elements (weaknesses violating software quality rules) that compose the CISQ Automated Source Code Maintainability Measure are presented in Table 1. This measure contains 28 parent weaknesses and 3 contributing weaknesses.

Table 1. Quality Measure Elements for Automated Source Code Maintainability Measure

CWE #	Descriptor	Weakness Description
CWE-1075	**Control transferred outside switch statement**	The software transfers control flow outside a switch statement (e.g., depending on the technology, by using a 'go to' statement)
CWE-1055	**Class Element Excessive Inheritance of Class Elements with Concrete Implementation**	A class inherits from too many concrete classes (default threshold for the maximum number of concrete class Inheritances is 1, *alternate threshold can be set prior to analysis*).
CWE-1052	**Storable and Member Data Element Initialization with Hard-Coded Literals**	The software uses a literal value to initialize a variable, field, member, etc. (exceptions are simple integers and a static constant variable, field, member, etc.)
CWE-1048	**Callable and Method Control Element Number of Outward Calls**	A function, method, procedure, stored procedure, or sub-routine references too many other objects within the application (default threshold for the maximum number of references is 5, *alternate threshold can be set prior to analysis*)
CWE-1095	**Loop Value Update within the Loop**	Within the body of a loop, the software updates the value of a local variable, field, member, etc. used in the loop condition.
CWE-1085	**Commented-out Code Element Excessive Volume**	A software component within the application contains too many commented-out instructions (default threshold for the maximum percent of commented-out instructions is 2%, *alternate threshold can be set prior to analysis*).
CWE-1047	**Inter-Module Dependency Cycles**	A software component within the application contains references that cycle back to itself (for example, in JAVA this pattern means cycles between packages).
CWE-1080	**Source Element Excessive Size**	A file within the application has too many logical source lines of code (default threshold for the maximum lines of code is 1000, *alternate threshold can be set prior to analysis*).
CWE-1054	**Named Callable and Method Control Element with Layer-skipping Call**	A function, method, procedure, stored procedure, or sub-routine calls a function, method, procedure, stored procedure, or sub-routine in a different architectural layer that violates the allowable connections as defined in a model of the application's architectural layers.

CWE-1093	**Callable and Method Control Element Excessive Cyclomatic Complexity Value**	A function, method, procedure, stored procedure, sub-routine, etc. has a Cyclomatic Complexity that is too large compared to a threshold value (default threshold for Cyclomatic Complexity is 20, *alternate threshold can be set prior to analysis*).
CWE-1064	**Callable and Method Control Element Excessive Number of Parameters**	A function, method, procedure, stored procedure, or sub-routine has too many parameters in its signature (default threshold for the maximum number of parameters is 7, *alternate threshold can be set prior to analysis*).
CWE-1084	**Callable and Method Control Element Excessive Number of Control Elements involving Data Element from Data Manager or File Resource**	A function, method, procedure, stored procedure, or sub-routine has too many SQL or file operations (default threshold for the maximum number of SQL or file operations is 7, *alternate threshold can be set prior to analysis*).
CWE-1081	**Public member element**	The software should not declare an uncontrolled data element as public.
CWE-1090	**Method Control Element Usage of Member Element from other Class Element**	A method from a class accesses a field or member from another class.
CWE-1074	**Class Element Excessive inheritance Level**	A class inheritance level is too large (default threshold for maximum Inheritance levels is 7, *alternate threshold can be set prior to analysis*).
CWE-1086	**Class Element Excessive Number of Children**	The number of children of a class is too large (default threshold for the maximum number of children of a class is 10, *alternate threshold can be set prior to analysis*).
CWE-1041	**Named Callable and Method Control Element Excessive Similarity**	The number of logical instructions that have been copied and pasted to other parts of the software exceeds a threshold value. The default threshold for each instance of copy-pasted code sets the maximum number of allowable copy-pasted instructions at 10% of the total instructions in the instance, *alternate thresholds can be set prior to analysis*).
CWE-561	**Dead code**	The software contains dead code that can never be executed. Thresholds are set at 5% logically dead code or code that is 0% structurally dead. Code that exists in the source but not in the object does not count.
CWE-1061	**Unreachable Named Callable or Method Control Element**	The software contains a function or method that is unreferenced and unused by any other software element in the application. The measure is the number of unreferenced or unused software elements. The defined application boundary determines the scope of the search for software elements that could call a function or method element; exceptions are getters

		and setters, as well as libraries outside the scope of the application.
CWE-570	**Expression is Always False**	The software contains an expression that will always evaluate to false.
CWE-571	**Expression is Always True**	The software contains an expression that will always evaluate to true.
CWE-1062	**Parent Class Element with References to Child Class Element**	A parent class references one of its child classes, directly or indirectly via its methods and fields.
CWE-1087	**Class Element with Virtual Method Element without Virtual Destructor**	A class contains a virtual method, yet the class does not declare any virtual destructor.
CWE-1079	**Parent Class Element without Virtual Destructor Method Element**	For languages in which custom destructors can be written, the parent has no virtual destructor.
CWE-1045	**Child Class Element without Virtual Destructor unlike its Parent Class Element**	For languages in which custom destructors can be written, the child class does not have its own virtual destructor, while its parent class has a virtual destructor.
CWE-1051	**Storable and Member Data Element Initialization with Hard-Coded Network Resource Configuration Data**	A variable, field, member, etc. is initialized with a hard-coded network resource identification information
CWE-484	**Omitted Break Statement in Switch**	The program omits a break statement within a switch or similar construct, causing code associated with multiple conditions to execute when only associated with one condition was intended to execute code.
CWE-480	**Use of Incorrect Operator**	The programmer accidentally uses the wrong operator, which changes the application logic in security-relevant ways.
CWE-478	**Missing Default Case in Switch Statemen**	The code does not have a default case in a switch statement, which can lead to complex logical errors.

CWE-783	**Operator Precedence Logic Error**	While often just a bug, operator precedence logic errors can have serious consequences if they are used in security-critical code, such as making an authentication decision.
CWE-407	**Algorithmic Complexity**	Remove instances where a module has references that cycle back to itself, e.g., the existence of cycles between packages in JAVA.

6.4 Automated Source Code Performance Efficiency Measure Element Descriptions

The quality measure elements (weaknesses violating software quality rules) that compose the CISQ Automated Source Code Performance Efficiency Measure are presented in Table 2. This measure contains 16 parent weaknesses and 3 contributing weaknesses (children in the CWE) that represent variants of these weaknesses. The CWE numbers for contributing weaknesses is presented in light blue cells immediately below the parent weakness whose CWE number is in a dark blue cell.

Table 2. Quality Measure Elements for Automated Source Code Performance Efficiency Measure

CWE #	Descriptor	Weakness Description
CWE-1046	**Immutable Storable and Member Data Element Creation**	A software operation inside a loop creates immutable text data via a string concatenation (which could be avoided by using text buffer instead).
CWE-1042	**Static Member Data Element outside of a Singleton Class Element**	The software declares static field as static, but its parent class is not a singleton class; it does not account for final static fields or members.
CWE-1049	**Data Resource Read and Write Access Excessive Complexity**	A SQL statement with too many joins (default threshold is 5 joins, *alternate threshold can be set prior to analysis*) and too many sub-queries (default threshold is 3 sub-queries, *alternate threshold can be set prior to analysis*) accesses a very large table exceeding a threshold number of rows (default threshold is 1,000,000 rows, *alternate threshold can be set prior analysis*).

CWE-1067	**Data Resource Read Access Unsupported by Index Element**	The syntax of a SQL SELECT statement and the index configuration of a SQL table or SQL view causes the DBMS to run sequential searches on a very large table exceeding a threshold number of rows (default threshold is 1,000,000 rows, *alternate threshold can be set prior to analysis*).
CWE-1089	**Large Data Resource ColumnSet Excessive Number of Index Elements**	A very large table exceeding a threshold number of rows (default is 1,000,000 rows, *alternate threshold can be set prior to analysis*) has too many indices (default threshold for the maximum number of indices is 3, *alternate threshold can be set prior to analysis*).
CWE-1094	**Large Data Resource ColumnSet with Index Element of Excessive Size**	The software writes to a very large table exceeding a threshold number of rows (default threshold is 1,000,000 rows, *alternate threshold can be set prior to analysis*) and has an index whose size is too large (default threshold for the index range is 10, *alternate threshold can be set prior to analysis).*
CWE-1050	**Control Elements Requiring Significant Resource Element within Control Flow Loop Block**	A software operation that is directly or indirectly called within a loop body or loop condition consumes platform resources (messaging, lock, file, stream, directory, etc.) beyond an acceptable threshold (default threshold is **XX** platform resources, *alternate threshold can be set prior to analysis*).
CWE-1060	**Non-stored SQL Callable Control Element with Excessive Number of Data Resource Access**	A server-side non-stored procedure contains too many data queries (default threshold for maximum number of data queries is **5**, *alternate threshold can be set prior to analysis*).
CWE-1073	**Excessive data queries in client-side code**	A client-side software operation contains too many data queries (default threshold for the maximum number of data queries is 2, *alternate threshold can be set prior to analysis*).

CWE-1057	**Data Access Control Element from Outside Designated Data Manager Component**	The software executes a data access outside of a dedicated data access component, thus circumventing the intended design to deny direct data access, thus allowing access only through dedicated data access components. Notes: · The dedicated data access component can be either client-side or server-side, which means that data access components can be developed using non-SQL language. · If there is no dedicated data access component, every data access is a violation. · For some embedded software that requires access to data from anywhere, the whole software is defined as a data access component. This condition must be identified as input to the analysis.
CWE-1043	**Storable and Member Data Element Excessive Number of Aggregated Storable and Member Data Elements**	The software contains a data element aggregated from too many non-primitive data types (default threshold for the maximum number of aggregated non-primitive data types is 5, *alternate threshold can be set prior to analysis*).
CWE-1072	**Data access not using connection pool**	The software executes a data resource management action without using a connection pool (the use of a connection pool is technology dependent; for example, connection pooling is disabled with the addition of 'Pooling=false' to the connection string with ADO.NET or the value of a 'com.sun.jndi.ldap.connect.pool' environment parameter in Java).
CWE-404	**Improper Resource Shutdown or Release**	The program does not release or incorrectly releases a resource before it is made available for re-use.
CWE-401	**Improper Release of Memory Before Removing Last Reference ('Memory Leak')**	The software does not sufficiently track and release allocated memory after it has been used, which slowly consumes remaining memory.
CWE-772	**Missing Release of Resource after Effective Lifetime**	The software does not release a resource after its effective lifetime has ended, i.e., after the resource is no longer needed.
CWE-775	**Missing Release of File Descriptor or Handle after Effective Lifetime**	The software does not release a file descriptor or handle after its effective lifetime has ended, i.e., after the file descriptor/handle is no longer needed.

CWE-1071	**Storable and Member Data Element Memory Allocation Missing De-allocation Control Element**	A method locks and unlocks an object without ever de-referencing it.
CWE-1091	**Storable and Member Data Element Reference Missing De-referencing Control Element**	The software is missing a dereferencing element that operates on a pointer variable and returns an l-value equivalent to the value at the pointer address.
CWE-424	**Improper Protection of Alternate Path**	The product does not sufficiently protect all possible paths that a user can take to access restricted functionality or resources.

6.5 Automated Source Code Reliability Measure Element Descriptions

The quality measure elements (weaknesses violating software quality rules) that compose the CISQ Automated Source Code Reliability Measure are presented in Table 3. This measure contains 36 parent weaknesses and 38 contributing weaknesses (children in the CWE) that represent variants of these weaknesses. The CWE numbers for contributing weaknesses is presented in light blue cells immediately below the parent weakness whose CWE number is in a dark blue cell.

Table 3. Quality Measure Elements for Automated Source Code Reliability Measure

CWE #	Descriptor	Weakness description
CWE-119	**Improper reading or writing to a memory buffer**	The software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer.
CWE-120	**Classic buffer overflow**	The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.
CWE-123	**Write-what-where condition**	The program allows an arbitrary value to be written to an arbitrary location, often as the result of a buffer overflow.
CWE-125	**Out-of-bounds read**	The software reads data past the end, or before the beginning, of the intended buffer.
CWE-130	**Improper Handling of Length Parameter Inconsistency**	The software parses a formatted message or structure, but it does not handle or incorrectly handles a length field that is inconsistent with the actual length of the associated data.

CWE-786	**Access of Memory Location Before Start of Buffer**	The software reads or writes to a buffer using an index or pointer that references a memory location prior to the beginning of the buffer. This typically occurs when a pointer or its index is decremented to a position before the buffer, when pointer arithmetic results in a position before the beginning of the valid memory location, or when a negative index is used.
CWE-787	**Out-of-bounds Write**	The software writes data past the end, or before the beginning, of the intended buffer. The software may modify an index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer.
CWE-788	**Access of Memory Location After End of Buffer**	The software reads or writes to a buffer using an index or pointer that references a memory location after the end of the buffer. This typically occurs when a pointer or its index is decremented to a position before the buffer; when pointer arithmetic results in a position before the buffer; or when a negative index is used, which generates a position before the buffer.
CWE-805	**Buffer Access with Incorrect Length Value**	The software uses a sequential operation to read or write a buffer, but it uses an incorrect length value that causes it to access memory that is outside of the bounds of the buffer.
CWE-822	**Untrusted Pointer Dereference**	The program obtains a value from an untrusted source, converts this value to a pointer, and dereferences the resulting pointer. There are several variants of this weakness, including but not necessarily limited to: • The untrusted value is directly invoked as a function call. • In OS kernels or drivers where there is a boundary between "userland" and privileged memory spaces, an untrusted pointer might enter through an API or system call (see CWE-781 for one such example). • Inadvertently accepting the value from an untrusted control sphere when it did not have to be accepted as input at all. This might occur when the code was originally developed to be run by a single user in a non-networked environment, and the code is then ported to or otherwise exposed to a networked environment.

CWE-823	**Use of Out-of-range Pointer Offset**	The program performs pointer arithmetic on a valid pointer, but it uses an offset that can point outside of the intended range of valid memory locations for the resulting pointer. • While a pointer can contain a reference to any arbitrary memory location, a program typically only intends to use the pointer to access limited portions of memory, such as contiguous memory used to access an individual array. • Programs may use offsets to access fields or sub-elements stored within structured data. The offset might be out-of-range if it comes from an untrusted source, is the result of an incorrect calculation, or occurs because of another error.
CWE-824	**Access of Uninitialized Pointer**	The program accesses or uses a pointer that has not been initialized. If the pointer contains an uninitialized value, then the value might not point to a valid memory location.
CWE-825	**Expired Pointer Dereference**	The program dereferences a pointer that contains a location for memory that was previously valid, but is no longer valid.
CWE-703	**Improper Check or Handling of Exceptional Condition**	Address instances where the software does not properly anticipate or handle exceptional conditions that rarely occur during normal operation of the software.
CWE-248	**Uncaught Exception**	An exception is thrown from a function, but it is not caught.
CWE-391	**Unchecked Error Condition**	Ignoring exceptions and other error conditions may allow an attacker to induce unexpected behavior unnoticed.
CWE-392	**Missing Report of Error Condition**	The software encounters an error but does not provide a status code or return value to indicate that an error has occurred.
CWE-252	**Unchecked Return Value**	The software does not check the return value from a method or function, which can prevent it from detecting unexpected states and conditions.
CWE-908	**Use of Uninitialized Resource**	Address instances where the software uses a resource that has not been properly initialized.
CWE-835	**Loop with Unreachable Exit Condition ('Infinite Loop')**	The program contains an iteration or loop with an exit condition that cannot be reached, i.e., an infinite loop.
CWE-704	**Incorrect Type Conversion or Cast**	The software does not correctly convert an object, resource, or structure from one type to a different type.
CWE-404	**Improper Resource Shutdown or Release**	The program does not release or incorrectly releases a resource before it is made available for re-use.

CWE-772	**Missing Release of Resource after Effective Lifetime**	The software does not release a resource after its effective lifetime has ended, i.e., after the resource is no longer needed.
CWE-401	**Improper Release of Memory Before Removing Last Reference ('Memory Leak')**	The software does not sufficiently track and release allocated memory after it has been used, which slowly consumes remaining memory.
CWE-775	**Missing Release of File Descriptor or Handle after Effective Lifetime**	The software does not release a file descriptor or handle after its effective lifetime has ended, i.e., after the file descriptor/handle is no longer needed. When a file descriptor or handle is not released after use (typically by explicitly closing it), attackers can cause a denial of service by consuming all available file descriptors/handles, or otherwise preventing other system processes from obtaining their own file descriptors/handles.
CWE-390	**Detection of Error Condition Without Action**	The software detects a specific error, but takes no actions to handle the error, for instance, where an exception handling block (such as Catch and Finally blocks) do not contain any instruction, making it impossible to accurately identify and adequately respond to unusual and unexpected conditions.
CWE-662	**Improper Synchronization**	The software attempts to use a shared resource in an exclusive manner, but does not prevent or incorrectly prevents use of the resource by another thread or process.
CWE-366	**Race Condition within a Thread**	If two threads of execution use a resource simultaneously, there exists the possibility that resources may be used while invalid, in turn making the state of execution undefined.
CWE-543	**Use of Singleton Pattern Without Synchronization in a Multithreaded Context**	The software uses the singleton pattern when creating a resource within a multithreaded environment.
CWE-567	**Unsynchronized Access to Shared Data in a Multithreaded Context**	The product does not properly synchronize shared data, such as static variables across threads, which can lead to undefined behavior and unpredictable data changes.
CWE-667	**Improper Locking**	The software does not properly acquire a lock on a resource, or it does not properly release a lock on a resource, leading to unexpected resource state changes and behaviors.
CWE-764	**Multiple Locks of a Critical Resource**	The software locks a critical resource more times than intended, leading to an unexpected state in the system.

CWE-820	**Missing Synchronization**	The software utilizes a shared resource in a concurrent manner but does not attempt to synchronize access to the resource.
CWE-821	**Incorrect Synchronization**	The software utilizes a shared resource in a concurrent manner but it does not correctly synchronize access to the resource.
CWE-1058	**Named Callable and Method Control Element in Multi-Thread Context with non-Final Static Storable or Member Element**	A control element owns an unsafe non-final static data element while it operates in a multi-threaded environment.
CWE-1096	**Singleton Class Instance Creation without Proper Lock Element Management**	The software instantiates a singleton class without activating any prior locking mechanism.
CWE-595	**Comparison of Object References Instead of Object Contents**	The program compares object references instead of the contents of the objects themselves, preventing it from detecting equivalent objects.
CWE-597	**Use of Wrong Operator in String Comparison**	The software uses the wrong operator when comparing a string, such as using "==" when the equals() method should be used instead. In Java, using == or != to compare two strings for equality actually compares two objects for equality, not their values.
CWE-1097	**Persistent Storable Data Element without Proper Comparison Control Element**	Remove instances where the persistent data has missing or improper dedicated comparison operations. Note: * In case of technologies with classes, this means situations where a persistent field is from a class that is made persistent while it does not implement methods from the list of required comparison operations (a JAVA example is the list composed of {'hashCode()','equals()'} methods)
CWE-1098	**Storable or Member Data Element containing Pointer Item Element without Proper Copy Control Element**	The software contains a pointer but no dedicated copy operation or copy constructor.
CWE-1082	**Class Instance Self Destruction Control Element**	Address instances where a class can self-destruct (an example of a self-destruction in C++ is 'delete this')

CWE-1077	**Float Type Storable and Member Data Element Comparison with Equality Operator**	Address instances where the float values of a variable, field, member, etc. are compared for equality using regular comparison operators (an example in JAVA, is the use of '= =' or '!=') instead of being checked for precision.
CWE-1083	**Data Access Control Element from Outside Designated Data Manager Component**	The software executes a data access outside of a dedicated data access component, thus circumventing the intended design to deny direct data access, thus allowing access only through dedicated data access components. Notes: • The dedicated data access component can be either client-side or server-side, which means that data access components can be developed using non-SQL language. • If there is no dedicated data access component, every data access is a violation. • For some embedded software that requires access to data from anywhere, the whole software is defined as a data access component. This condition must be identified as input to the analysis.
CWE-1088	**Synchronous Call Timeout Absence**	Software allows synchronous remote resource access without handling time-out capabilities.
CWE-682	**Incorrect Calculation**	The software performs a calculation that generates incorrect or unintended results that are later used in security-critical decisions or resource management.
CWE-131	**Incorrect Calculation of Buffer Size**	The software does not correctly calculate the size to be used when allocating a buffer, which could lead to a buffer overflow.
CWE-369	**Divide By Zero**	The product divides a value by zero.
CWE-394	**Unexpected Status Code or Return Value**	The software does not properly check when a function or operation returns a value that is legitimate for the function, but is not expected by the software.
CWE-170	**Improper Null Termination**	The software does not terminate or incorrectly terminates a string or array with a null character or equivalent terminator.
CWE-672	**Operation on a Resource after Expiration or Release**	The software uses, accesses, or otherwise operates on a resource after that resource has been expired, released, or revoked.
CWE-415	**Double Free**	The product calls free() twice on the same memory address, potentially leading to modification of unexpected memory locations.

CWE-416	**Use After Free**	Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code.
CWE-459	**Incomplete Cleanup**	The software does not properly "clean up" and remove temporary or supporting resources after they have been used.
CWE-562	**Return of Stack Variable Address**	Because local variables are allocated on the stack, when a program returns a pointer to a local variable, it is returning a stack address. A subsequent function call is likely to re-use this same stack address, thereby overwriting the value of the pointer, which no longer corresponds to the same variable since a function's stack frame is invalidated when it returns. At best this will cause the value of the pointer to change unexpectedly. In many cases it causes the program to crash the next time the pointer is dereferenced
CWE-758	**Reliance on Undefined, Unspecified, or Implementation-Defined Behavior**	The software uses an API function, data structure, or other entity in a way that relies on properties that are not always guaranteed to hold for that entity.
CWE-476	**NULL Pointer Dereference**	A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit.
CWE-681	**Incorrect Conversion between Numeric Types**	The software declares a variable, field, member, etc. with a numeric type, and then updates it with a value from a second numeric type that is incompatible with the first numeric type.
CWE-194	**Unexpected Sign Extension**	The software performs an operation on a number that causes it to be sign-extended when it is transformed into a larger data type. When the original number is negative, this can produce unexpected values that lead to resultant weaknesses.
CWE-195	**Signed to Unsigned Conversion Error**	The software uses a signed primitive and performs a cast to an unsigned primitive, which can produce an unexpected value if the value of the signed primitive cannot be represented using an unsigned primitive.
CWE-196	**Unsigned to Signed Conversion Error**	The software uses an unsigned primitive and performs a cast to a signed primitive, which can produce an unexpected value if the value of the unsigned primitive cannot be represented using a signed primitive.

CWE-197	**Numeric Truncation Error**	When a primitive is cast to a smaller primitive, the high order bits of the large value are lost in the conversion, potentially resulting in an unexpected value that is not equal to the original value. This value may be required as an index into a buffer, a loop iterator, or simply necessary state data. In any case, the value cannot be trusted and the system will be in an undefined state. While this method may be employed viably to isolate the low bits of a value, this usage is rare, and truncation usually implies that an implementation error has occurred.
CWE-484	**Omitted Break Statement in Switch**	The program omits a break statement within a switch or similar construct, causing code associated with multiple conditions to execute when only code associated with one condition was intended to execute.
CWE-665	**Improper Initialization**	The software does not initialize or incorrectly initializes a resource, which might leave the resource in an unexpected state when it is accessed or used.
CWE-456	**Missing Initialization of a Variable**	The software does not initialize critical variables, which causes the execution environment to use unexpected values.
CWE-457	**Use of uninitialized variable**	The software uses a variable that has not been initialized.
CWE-480	**Use of Incorrect Operator**	The programmer accidentally uses the wrong operator, which changes the application logic in security-relevant ways.
CWE-424	**Improper Protection of Alternate Path**	The product does not sufficiently protect all possible paths that a user can take to access restricted functionality or resources.
CWE-833	**Deadlock**	The software contains multiple threads or executable segments that are waiting for each other to release a necessary lock, resulting in deadlock.
CWE-1087	**Class Element with Virtual Method Element without Virtual Destructor**	The software fails to include a virtual destructor in a class that includes a virtual method(s).
CWE-1079	**Parent Class Element without Virtual Destructor Method Element**	The software fails to include a virtual destructor in a parent class.
CWE-1045	**Child Class Element without Virtual Destructor unlike its Parent Class Element**	The software fails to include a virtual destructor in a child class despite the existence of a virtual destructor in the parent class.

CWE-1051	**Storable and Member Data Element Initialization with Hard-Coded Network Resource Configuration Data**	The software contains hard-coded values corresponding to network resource identifications.
CWE-1066	**Serializable Storable Data Element without Serialization Control Element**	The software fails to fully implement serialization capabilities.
CWE-1070	**Serializable Storable Data Element with non-Serializable Item Elements**	The software contains an incomplete implementation of serialization capabilities.

6.6 Automated Source Code Security Measure Element Descriptions

The quality measure elements (weaknesses violating software quality rules) that compose the CISQ Automated Source Code Security Measure are presented in Table 4. This measure contains 37 parent weaknesses and 36 contributing weaknesses (children in the CWE) that represent variants of these weaknesses. The CWE numbers for contributing weaknesses is presented in light blue cells immediately below the parent weakness whose CWE number is in a dark blue cell.

Table 4. Quality Measure Elements for Automated Source Code Security Measure

CWE #	Descriptor	Weakness description
CWE-22	**Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')**	The software uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory, but the software does not properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory.
CWE-23	Relative Path Traversal	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize sequences such as ".." that can resolve to a location that is outside of that directory.
CWE-36	**Absolute Path Traversal**	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize absolute path sequences such as

		"/abs/path" that can resolve to a location that is outside of that directory.
CWE-77	**Improper Neutralization of Special Elements used in a Command ('Command Injection')**	The software constructs all or part of a command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended command when it is sent to a downstream component.
CWE-78	**Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')**	The software constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.
CWE-88	**Argument Injection or Modification**	The software does not sufficiently delimit the arguments being passed to a component in another control sphere, allowing alternate arguments to be provided, leading to potentially security-relevant changes.
CWE-79	**Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')**	The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.
CWE-89	**Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')**	The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.
CWE-564	**SQL Injection: Hibernate**	Using Hibernate to execute a dynamic SQL statement built with user-controlled input can allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands.
CWE-99	**Resource injection**	The software receives input from an upstream component, but it does not restrict or incorrectly restricts the input before it is used as an identifier for a resource that may be outside the intended sphere of control.
CWE-119	**Improper Restriction of Operations within the Bounds of a Memory Buffer**	The software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer.

CWE-120	**Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')**	The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.
CWE-123	**Write-what-where condition**	The program allows an arbitrary value to be written to an arbitrary location, often as the result of a buffer overflow.
CWE-125	**Out-of-bounds Read**	The software reads data past the end, or before the beginning, of the intended buffer.
CWE-130	**Improper Handling of Length Parameter Inconsistency**	The software parses a formatted message or structure, but it does not handle or incorrectly handles a length field that is inconsistent with the actual length of the associated data.
CWE-786	**Access of Memory Location Before Start of Buffer**	The software reads or writes to a buffer using an index or pointer that references a memory location prior to the beginning of the buffer. This typically occurs when a pointer or its index is decremented to a position before the buffer, when pointer arithmetic results in a position before the beginning of the valid memory location, or when a negative index is used.
CWE-787	**Out-of-bounds Write**	The software writes data past the end, or before the beginning, of the intended buffer. The software may modify an index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer.
CWE-788	**Access of Memory Location After End of Buffer**	The software reads or writes to a buffer using an index or pointer that references a memory location after the end of the buffer. This typically occurs when a pointer or its index is decremented to a position before the buffer; when pointer arithmetic results in a position before the buffer; or when a negative index is used, which generates a position before the buffer.
CWE-805	**Buffer Access with Incorrect Length Value**	The software uses a sequential operation to read or write a buffer, but it uses an incorrect length value that causes it to access memory that is outside of the bounds of the buffer.

CWE-822	**Untrusted Pointer Dereference**	The program obtains a value from an untrusted source, converts this value to a pointer, and dereferences the resulting pointer. There are several variants of this weakness, including but not necessarily limited to: • The untrusted value is directly invoked as a function call. • In OS kernels or drivers where there is a boundary between "userland" and privileged memory spaces, an untrusted pointer might enter through an API or system call (see CWE-781 for one such example). • Inadvertently accepting the value from an untrusted control sphere when it did not have to be accepted as input at all. This might occur when the code was originally developed to be run by a single user in a non-networked environment, and the code is then ported to or otherwise exposed to a networked environment.
CWE-823	**Use of Out-of-range Pointer Offset**	The program performs pointer arithmetic on a valid pointer, but it uses an offset that can point outside of the intended range of valid memory locations for the resulting pointer. • While a pointer can contain a reference to any arbitrary memory location, a program typically only intends to use the pointer to access limited portions of memory, such as contiguous memory used to access an individual array. • Programs may use offsets to access fields or sub-elements stored within structured data. The offset might be out-of-range if it comes from an untrusted source, is the result of an incorrect calculation, or occurs because of another error.
CWE-824	**Access of Uninitialized Pointer**	The program accesses or uses a pointer that has not been initialized. If the pointer contains an uninitialized value, then the value might not point to a valid memory location.
CWE-825	**Expired Pointer Dereference**	The program dereferences a pointer that contains a location for memory that was previously valid, but is no longer valid.
CWE-129	**Improper Validation of Array Index**	The product uses untrusted input when calculating or using an array index, but the product does not validate or incorrectly validates the index to ensure the index references a valid position within the array.
CWE-134	**Use of Externally-Controlled Format String**	The software uses a function that accepts a format string originating from an external source as an argument, but the format string is not sanitized prior to use based on a list of vetted sanitization functions.

CWE-252	**Unchecked Return Value**	The software does not check the return value from a method or function, which can prevent it from detecting unexpected states and conditions.
CWE-434	**Unrestricted Upload of File with Dangerous Type**	The software allows the upload or transfer files of dangerous types that can be automatically processed within the product's environment.
CWE-665	**Improper Initialization**	The software does not initialize or incorrectly initializes a resource, which might leave the resource in an unexpected state when it is accessed or used.
CWE-456	**Missing Initialization of a Variable**	The software does not initialize critical variables, which causes the execution environment to use unexpected values.
CWE-457	**Use of uninitialized variable**	The software uses a variable that has not been initialized.
CWE-606	**Unchecked input for loop condition**	The software accepts a user input without any range check prior to being used in a loop condition statement.
CWE-662	**Improper Synchronization**	The software attempts to use a shared resource in an exclusive manner, but does not prevent or incorrectly prevents use of the resource by another thread or process.
CWE-366	**Race Condition within a Thread**	If two threads of execution use a resource simultaneously, there exists the possibility that resources may be used while invalid, in turn making the state of execution undefined.
CWE-543	**Use of Singleton Pattern Without Synchronization in a Multithreaded Context**	The software uses the singleton pattern when creating a resource within a multithreaded environment.
CWE-567	**Unsynchronized Access to Shared Data in a Multithreaded Context**	The product does not properly synchronize shared data, such as static variables across threads, which can lead to undefined behavior and unpredictable data changes.
CWE-667	**Improper Locking**	The software does not properly acquire a lock on a resource, or it does not properly release a lock on a resource, leading to unexpected resource state changes and behaviors.
CWE-820	**Missing Synchronization**	The software utilizes a shared resource in a concurrent manner but does not attempt to synchronize access to the resource.
CWE-821	**Incorrect Synchronization**	The software utilizes a shared resource in a concurrent manner but it does not correctly synchronize access to the resource.

CWE-672	**Operation on a Resource after Expiration or Release**	The software uses, accesses, or otherwise operates on a resource after that resource has been expired, released, or revoked.
CWE-415	**Double Free**	The product calls free() twice on the same memory address, potentially leading to modification of unexpected memory locations.
CWE-416	**Use After Free**	Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code.
CWE-681	**Incorrect Conversion between Numeric Types**	The software declares a variable, field, member, etc. with a numeric type, and then updates it with a value from a second numeric type that is incompatible with the first numeric type.
CWE-194	**Unexpected Sign Extension**	The software performs an operation on a number that causes it to be sign-extended when it is transformed into a larger data type. When the original number is negative, this can produce unexpected values that lead to resultant weaknesses.
CWE-195	**Signed to Unsigned Conversion Error**	The software uses a signed primitive and performs a cast to an unsigned primitive, which can produce an unexpected value if the value of the signed primitive cannot be represented using an unsigned primitive.
CWE-196	**Unsigned to Signed Conversion Error**	The software uses an unsigned primitive and performs a cast to a signed primitive, which can produce an unexpected value if the value of the unsigned primitive cannot be represented using a signed primitive.
CWE-197	**Numeric Truncation Error**	When a primitive is cast to a smaller primitive, the high order bits of the large value are lost in the conversion, potentially resulting in an unexpected value that is not equal to the original value. This value may be required as an index into a buffer, a loop iterator, or simply necessary state data. In any case, the value cannot be trusted and the system will be in an undefined state. While this method may be employed viably to isolate the low bits of a value, this usage is rare, and truncation usually implies that an implementation error has occurred.
CWE-404	**Improper Resource Shutdown or Release**	The program does not release or incorrectly releases a resource before it is made available for re-use.
CWE-401	**Improper Release of Memory Before Removing Last Reference ('Memory Leak')**	The software does not sufficiently track and release allocated memory after it has been used, which slowly consumes remaining memory.

CWE-772	**Missing Release of Resource after Effective Lifetime**	The software does not release a resource after its effective lifetime has ended, i.e., after the resource is no longer needed.
CWE-798	**Use of Hard-coded Credentials**	The software contains hard-coded credentials, such as a password or cryptographic key, which it uses for its own inbound authentication, outbound communication to external components, or encryption of internal data.
CWE-259	**Use of Hard-coded Password**	The software contains a hard-coded password, which it uses for its own inbound authentication or for outbound communication to external components.
CWE-321	**Use of Hard-coded Cryptographic Key**	The software uses a hard-coded cryptographic key.
CWE-835	**Loop with Unreachable Exit Condition ('Infinite Loop')**	The program contains an iteration or loop with an exit condition that cannot be reached, i.e., an infinite loop.
CWE 778	**Insufficient logging of security events**	When a security-critical event occurs, it is either not recorded or important details about the event are omitted when logging it
CWE-789	**Uncontrolled Memory Allocation**	The product allocates memory based on an untrusted size value, but it does not validate or incorrectly validates the size, allowing arbitrary amounts of memory to be allocated.
CWE-682	**Incorrect Calculation**	The software performs a calculation that generates incorrect or unintended results that are later used in security-critical decisions or resource management.
CWE-131	**Incorrect Calculation of Buffer Size**	The software does not correctly calculate the size to be used when allocating a buffer, which could lead to a buffer overflow.
CWE-369	**Divide By Zero**	The product divides a value by zero.
CWE-611	**Improper Restriction of XML External Entity Reference ('XXE')**	The software processes an XML document that can contain XML entities with URIs that resolve to documents outside of the intended sphere of control, causing the product to embed incorrect documents into its output.
CWE-502	**Deserialization of Untrusted Data**	The application deserializes untrusted data without sufficiently verifying that the resulting data will be valid.
CWE-775	**Missing Release of File Descriptor or Handle after Effective Lifetime**	The software does not release a file descriptor or handle after its effective lifetime has ended, i.e., after the file descriptor/handle is no longer needed. When a file descriptor or handle is not released after use (typically by explicitly closing it), attackers can cause a denial of service

		by consuming all available file descriptors/handles, or otherwise preventing other system processes from obtaining their own file descriptors/handles.
CWE-783	**Operator Precedence Logic Error**	The program uses an expression in which operator precedence causes incorrect logic to be used. While often just a bug, operator precedence logic errors can have serious consequences if they are used in security-critical code, such as making an authentication decision.
CWE-424	**Improper Protection of Alternate Path**	The product does not sufficiently protect all possible paths that a user can take to access restricted functionality or resources.
CWE-1057	**Circumventing data access routines**	The software executes a data access outside of a dedicated data access component, thus circumventing the intended design to deny direct data access, thus allowing access only through dedicated data access components. Notes: • The dedicated data access component can be either client-side or server-side, which means that data access components can be developed using non-SQL language. • If there is no dedicated data access component, every data access is a violation. • For some embedded software that requires access to data from anywhere, the whole software is defined as a data access component. This condition must be identified as input to the analysis.
CWE-480	**Use of Incorrect Operator**	The programmer accidentally uses the wrong operator, which changes the application logic in security-relevant ways.
CWE-570	**Expression is Always False**	The software contains an expression that will always evaluate to false.
CWE-571	**Expression Is Always True**	The software contains an expression that will always evaluate to true.
CWE-477	**Use of Obsolete Function**	The code uses deprecated or obsolete functions, which suggests that the code has not been actively reviewed or maintained.
CWE-643	**Improper Neutralization of Data within XPath Expressions ('XPath Injection')**	The software uses external input to dynamically construct an XPath expression used to retrieve data from an XML database, but it does not neutralize or incorrectly neutralizes that input. This allows an attacker to control the structure of the query.

CWE-652	**CWE-652 Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')**	The software uses external input to dynamically construct an XQuery expression used to retrieve data from an XML database, but it does not neutralize or incorrectly neutralizes that input. This allows an attacker to control the structure of the query.
CWE-732	**Incorrect Permission Assignment for Critical Resource**	The software specifies permissions for a security-critical resource in a way that allows that resource to be read or modified by unintended actors.
CWE-90	**Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')**	The software constructs all or part of an LDAP query using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended LDAP query when it is sent to a downstream component.
CWE-917	**XML Injection (aka Blind XPath Injection)**	The software does not properly neutralize special elements that are used in XML, allowing attackers to modify the syntax, content, or commands of the XML before it is processed by an end system.

6.4 Introduction to the Specification of Quality Measure Elements

Clauses 7, 8, and 9 display in human readable format the content of the machine readable XMI format file attached to this specification. The content of the machine readable XMI format file represents the Quality Measure Elements with the following conventions

- structural elements included in a weakness pattern are represented in the Knowledge Discovery Metamodel (KDM)
- relations among the structural elements constituting a weakness pattern are represented in the Software Patterns Metamodel Standard (SPMS) to compute measures at the weakness level.
- Calculation of the 4 measures are represented in the Structured Metrics Metamodel (SMM).

6.5 Knowledge Discovery Metamodel (KDM)

This specification uses the Knowledge Discovery Metamodel (KDM) to represent the parsed entities whose relationships create a weakness pattern. The machine readable XMI format file attached to the current specification uses KDM entities in the 'KDM outline' section of the pattern definitions to represent the code elements whose presence or absence indicates an occurrence of the weakness. Descriptions try to remain as generic, yet as accurate as possible, so that the pattern can be applied to as many situations as possible: different technologies, different programming languages, etc. This means:

1. The descriptions include information such as (MethodUnit), (Reads), (ManagesResource), … to identify the KDM entities included in the pattern definition.

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

Additional semantic constraints are required to coordinate producers and consumers of KDM models to use the KDM Program Element layer for control- and data-flow analysis applications, as well as for providing more precision for the Resource Layer and the Abstraction Layer. Micro-KDM achieves this by constraining the granularity of the leaf action elements and their meaning by providing the set of micro-actions with predefined semantics. Micro-KDM treats the original macro-action as a container that owns certain micro-actions with predefined semantics. Thus precise semantics of the macro-action is defined. Thus, micro-KDM constrains the patterns of how to map the statements of the existing system as determined by the programming language into KDM.

KDM is helpful for reading this chapter. However, for readers not familiar with KDM, Table 5 presents a primer which translates standard source code element terms into the KDM outline in this specification.

Table 5. Software elements translated into KDM wording

Software element	KDM outline
function, method, procedure, stored procedure, sub-routine etc.	`CallableUnit|MethodUnit id="ce1" ...`
variable, field, member, etc.	`StorableUnit|MemberUnit id="de1" ...`

class, interface definition and use as a type, use as base class	`ClassUnit|InterfaceUnit id="cu1" ...`
	`StorableUnit id="su1" type="cu1" ...`
	`ClassUnit id="cu2" ...`
	`  Extends "cu1" ...`

method	`ClassUnit id="cu2" ...` `  MethodUnit "mu1" ...`
field, member	`ClassUnit id="cu2" ...` `  MemberUnit "mu1" ...`

SQL stored procedures	DataModel RelationalSchema ... CallableUnit id="cu1" kind="stored" ...
return code value definition and use	CallableUnit|MethodUnit id="ce1" type="ce1_signature" ... Signature "ce1_signature" ParameterUnit id="pu1" kind="return" ... Value|StorableUnit|MemberUnit id="de1" ... ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall" ... Calls "ce1" Reads "de1"
exception	CallableUnit|MethodUnit id="ce1" type="ce1_signature" ... Signature "ce1_signature" ParameterUnit id="pu1" kind="exception" ...
user input data flow	UIModel UIField id="uf1" UIAction id="ua1" implementation="ae1" kind="input" ReadsUI "uf1" ... CodeModel ... StorableUnit id="su1" StorableUnit id="su2" ActionElement id="ae1" kind="UI" Writes "su1" Flow "ae2" ActionElement id="ae2" Flow "ae3" Reads "su1" Writes "su2" ActionElement id="ae3" Flow "ae4" ...

execution path	ActionElement id="ae1" kind="UI" Flow|Calls "ae2" ActionElement id="ae2" Flow|Calls "ae3" ActionElement id="ae3" Flow|Calls "ae4"
RDBMS	DataModel RelationalSchema ...
for loop	ActionElement id="ae5" kind="Compound" StorableUnit id="su3" ActionElement id="ae6" kind="Assign" Reads ... Writes "su3" Flows "ae7" ActionElement id="ae7" kind="LessThan|LessThanOrEqual|GreaterThan|GreaterThanOrEqual" Reads "su3" Reads "su2" TrueFlow "ae8" FalseFlow "ff1" ActionElement id="ae8" kind=... ... ActionElement id="ae9" kind="Incr|Decr" Addresses "loopVariable" Flows "ae6" ActionElement id="ff1" kind="Nop"
while loop	ActionElement id="ae5" kind="Compound" BooleanType id="booleanType" DataElement id="de1" type="booleanType" EntryFlow "tf1" ActionElement id="tf1" ... ... ActionElement id ="ae6" kind="GreaterThan|GreaterThanOrEqual|LessThan|LessThanOrEqual"

	Reads "su2" ... Writes "de1" ActionElement id="ae7" kind="Condition" Reads "de1" TrueFlow "tf1" FalseFlow "ff1" ActionElement id="ff1"
checked	Value|StorableUnit|MemberUnit id="de1" ... ActionElement id="ae1" kind="Equals|NotEqualTo|GreaterThan|GreaterThanOrEqual|LessThan|LessThanOrEqual" ... Reads "de1"

6.6 Software Patterns Metamodel Standard (SPMS)

This specification uses the Software Patterns Metamodel Standard (SPMS) to represent weaknesses as software patterns involving code elements and their relationships in source code. In the machine readable XMI format file attached to the current specification each weakness pattern is represented in SPMS Definitions Classes as follows:

- PatternDefinition (SPMS:PatternDefinition): the pattern specification describing a specific weakness and a specific detection pattern. In the context of this document, each Quality Measure Element is the count of occurrences of the SPMS detection patterns detected in the source code for a specific weakness related to the Quality Characteristic being measured.
- 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 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. Roles for weaknesses are abstractions, while the roles for detection patterns can be linked back to the code elements.
- 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 7 different PatternSections in use:
  - "Descriptor" ("descriptor" in the XMI document) to provide pattern signature, a visible interface of the pattern,
  - "Description" ("description" in XMI document) to provide a human readable explanation of the measure,

- "KDM Outline" ("kdm outline" in XMI document) to provide an illustration of the essential elements related to KDM, in a human readable outline,
- "What to report" ("reporting" in XMI document) to provide the list of elements to report to claim the finding of an occurrence of a detection pattern
- "Reference" ("reference" in XMI document) to provide pointers to the weakness description in the CWE repository
- "Usage name" ("usage_name" in XMI document) to provide a more user-friendly name to the weakness, generally the case when the weakness original name was too strongly KDM-flavored for the general audience

SPMS Relationships Classes:

- MemberOf (SPMS:MemberOf): "An InterpatternRelationship specialized to indicate inclusion in a Category"
- RelatedPattern (SPMS:RelatedPattern) with 4 different Natures (SPMS:Nature) ("DetectedBy", "Detecting"," AggregatedBy", and "Aggregating"): InterpatternRelationships used to model the relations between weaknesses and detection patterns, and between parent and child weaknesses
- 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 Quality Characteristics:
  - "Reliability",
  - "Security",
  - "Performance Efficiency",
  - And "Maintainability".

6.7 Reading guide

For each numbered sub-clause in clause 7:

- Sub-clause 7.x represents the Software Quality characteristic addressed by the associated weakness patterns .
- Sub-clause 7.x.y represents the SPMS and SMM modeling associated with a weakness pattern for a specific weakness associated with the Software Quality characteristic.
- The last sub-clause 7.x.y represents the SMM modeling associated with the quality characteristic computation.

Weakness pattern sub-clauses are summarizing the various aspects related to a weakness:

- (SPMS) usage name pattern section, if any
- (SPMS) reference pattern section

- (SPMS) roles
- (SPMS) contributing weaknesses and parent weakness, if any,
    - useful for reporting of weakness pattern-level information, aggregated or detailed
- (SPMS and SMM) detection patterns,
    - useful for reporting of detection pattern-level findings at the weakness level
    - useful for counting the violations to the weakness, by summing the count of violations to its detection patterns

Last sub-clauses are summarizing the computation of the quality measure scores:

- (SMM) detection patterns,
    - useful for reporting of detection pattern-level findings at the quality characteristic level
    - useful for computing the score of the quality measure, by summing the count of violations to its detection patterns
-

For each numbered sub-clause in clause 8:

- Sub-clause 8.x represents the SPMS modeling associated with a detection pattern

Detection pattern sub-clause are summarizing the various aspects related to a detection pattern:

- (SPMS) descriptor, description, KDM outline, reporting pattern sections,
    - In description and reporting pattern sections, data between angle brackets (e.g.: <ControlElement>) identify SPMS roles

7 List of ASCQM Weaknesses (Normative)

7.1 Weakness Category Maintainability

7.1.1 CWE-407 Algorithmic Complexity

Reference
https://cwe.mitre.org/data/definitions/407 Algorithmic Complexity

Roles
- the <ControlFlow>

Contributing weaknesses
MNT-11 Callable and Method Control Element Excessive Cyclomatic Complexity Value

Detection Patterns
ASCQM Ban Switch in Switch Statement
ASCQM Limit Algorithmic Complexity via Cyclomatic Complexity Value
ASCQM Limit Algorithmic Complexity via Essential Complexity Value
ASCQM Limit Algorithmic Complexity via Module Design Complexity Value

7.1.2 CWE-478 Missing Default Case in Switch Statement

Reference
https://cwe.mitre.org/data/definitions/478 Missing Default Case in Switch Statement

Roles
- the <SwitchStatement>

Detection Patterns
ASCQM Use Default Case in Switch Statement

7.1.3 Weakness CWE-480 Use of Incorrect Operator

Reference
https://cwe.mitre.org/data/definitions/480 Use of Incorrect Operator

Roles
- the <Operator>

Detection Patterns

ASCQM Ban Assignment Operation Inside Logic Blocks
ASCQM Ban Comparison Expression Outside Logic Blocks
ASCQM Ban Incorrect Object Comparison
ASCQM Ban Incorrect String Comparison
ASCQM Ban Logical Operation with a Constant Operand

7.1.4 CWE-484 Omitted Break Statement in Switch

Reference
https://cwe.mitre.org/data/definitions/484 Omitted Break Statement in Switch

Roles
- the <SwitchStatement>

Detection Patterns
ASCQM Use Break in Switch Statement

7.1.5 CWE-561 Dead Code

Reference
https://cwe.mitre.org/data/definitions/561 Dead Code

Roles
- the <DeadCode>

Detection Patterns
ASCQM Ban Exception Definition without Ever Throwing It
ASCQM Ban Logical Dead Code
ASCQM Ban Unreferenced Dead Code

7.1.6 CWE-570 Expression is Always False

Reference
https://cwe.mitre.org/data/definitions/570 Expression is Always False

Roles
- the <BooleanExpression>

Detection Patterns
ASCQM Check Boolean Variables are Updated in Different Conditional Branches before Use

7.1.7 CWE-571 Expression is Always True

Reference

https://cwe.mitre.org/data/definitions/571 Expression is Always True

Roles
- the <BooleanExpression>

Detection Patterns
ASCQM Check Boolean Variables are Updated in Different Conditional Branches before Use

7.1.8 CWE-783 Operator Precedence Logic Error

Reference
https://cwe.mitre.org/data/definitions/783 Operator Precedence Logic Error

Roles
- the <Formula>

Detection Patterns
ASCQM Ban Incorrect Joint Comparison
ASCQM Ban Not Operator On Non-Boolean Operand Of Comparison Operation
ASCQM Ban Not Operator On Operand Of Bitwise Operation

7.1.9 CWE-1075 Control Flow Transfer Control Element outside Switch Block

Usage name
Control transferred outside switch statement

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1075 Control Flow Transfer Control Element outside Switch Block

Roles
- the <SwitchBlock>
- the <ControlFlowTransfer>

Detection Patterns
ASCQM Limit Volume of Similar Code

7.1.10 CWE-1093 Callable and Method Control Element Excessive Cyclomatic Complexity Value

Usage name
Excessive Cyclomatic Complexity

Reference

https://www.omg.org/spec/ASCMM ASCMM-CWE-1093 Callable and Method Control Element Excessive Cyclomatic Complexity Value

Roles
- the <Operation>
- the <ControlFlow>

Parent weaknesses
MNT-11 Callable and Method Control Element Excessive Cyclomatic Complexity Value

Detection Patterns
ASCQM Limit Algorithmic Complexity via Cyclomatic Complexity Value

7.1.11 CWE-1054 Named Callable and Method Control Element with Layer-skipping Call

Usage name
Layer-skipping calls

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1054 Named Callable and Method Control Element with Layer-skipping Call

Roles
- the <Layer1>
- the <Layer2>
- the <Call>

Detection Patterns
ASCQM Ban Unintended Paths

7.1.12 CWE-1064 Callable and Method Control Element Excessive Number of Parameters

Usage name
Excessive parameterization

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1064 Callable and Method Control Element Excessive Number of Parameters

Roles
- the <OperationSignature>

Detection Patterns
ASCQM Limit Number of Parameters

7.1.13 CWE-1084 Callable and Method Control Element Excessive Number of Control Elements involving Data Element from Data Manager or File Resource

Usage name
Control element with excessive data operations

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1084 Callable and Method Control Element Excessive Number of Control Elements involving Data Element from Data Manager or File Resource

Roles
- the <Operation>
- the <DataAccesses>

Detection Patterns
ASCQM Limit Number of Data Access

7.1.14 CWE-1081 Public Member Element

Usage name
Public data element

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1081 Public Member Element

Roles
- the <PublicDataDeclaration>

Detection Patterns
ASCQM Ban Public Data Elements

7.1.15 CWE-1090 Method Control Element Usage of Member Element from other Class Element

Usage name
Cross element data access

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1090 Method Control Element Usage of Member Element from other Class Element

Roles
- the <Class1>
- the <Class2>
- the <Reference>

Detection Patterns
ASCQM Ban Usage of Data Elements from Other Classes

7.1.16 CWE-1074 Class Element Excessive Inheritance Level

Usage name
Excessive inheritance levels

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1074 Class Element Excessive Inheritance Level

Roles
Roles:
- the <ClassInheritanceTree>

Detection Patterns
ASCQM Ban Excessive Number of Inheritance Levels

7.1.17 CWE-1086 Class Element Excessive Number of Children

Usage name
Excessive child classes

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1086 Class Element Excessive Number of Children

Roles
- the <Class>
- the <Children>

Detection Patterns
ASCQM Ban Excessive Number of Children

7.1.18 CWE-1041 Named Callable and Method Control Element Excessive Similarity

Usage name
Element redundancy

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1041 Named Callable and Method Control Element Excessive Similarity

Roles
- the <Operation1>
- the <Operation2>
- the <SimilarCodeElements>

Detection Patterns
ASCQM Limit Volume of Similar Code

7.1.19 CWE-1055 Class Element Excessive Inheritance of Class Elements with Concrete Implementation

Usage name
Excessive inheritance from concrete classes

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1055 Class Element Excessive Inheritance of Class Elements with Concrete Implementation

Roles
- the <ClassInheritanceDeclaration>
- the <ConcreteClasses>

Detection Patterns
ASCQM Ban Excessive Number of Concrete Implementations to Inherit From

7.1.20 CWE-1061 Unreachable Named Callable or Method Control Element

Usage name
Unused code

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1061 Unreachable Named Callable or Method Control Element

Roles
- the <Operation>

Detection Patterns
ASCQM Ban Unreferenced Dead Code

7.1.21 CWE-1052 Storable and Member Data Element Initialization with Hard-Coded Literals

Usage name
Hard-coded literals

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1052 Storable and Member Data Element Initialization with Hard-Coded Literals

Roles
- the <Initialization>
- the <HardCodedValue>

Detection Patterns
ASCQM Ban Hard-Coded Literals used to Initialize Variables

7.1.22 CWE-1048 Callable and Method Control Element Number of Outward Calls

Usage name
Excessive references

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1048 Callable and Method Control Element Number of Outward Calls

Roles
- the <Operation>
- the <OutwardCalls>

Detection Patterns
ASCQM Limit Number of Outward Calls

7.1.23 CWE-1095 Loop Value Update within the Loop

Usage name
Condition value update within loop

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1095 Loop Value Update within the Loop

Roles
- the <LoopCondition>
- the <LoopValueUpdate>

Detection Patterns
ASCQM Ban Loop Value Update within Incremental and Decremental Loop

7.1.24 CWE-1085 Commented-out Code Element Excessive Volume

Usage name
Excessive commented-out code

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1085 Commented-out Code Element Excessive Volume

Roles
- the <CommentedOutCode>

Detection Patterns
ASCQM Limit Volume of Commented-Out Code

7.1.25 CWE-1047 Inter-Module Dependency Cycles

Usage name
Circular dependencies

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1047 Inter-Module Dependency Cycles

Roles
- the <ModuleDependencyCycles>

Detection Patterns
ASCQM Ban Circular Dependencies between Modules

7.1.26 CWE-1080 Source Element Excessive Size

Usage name
Excessively large file

Reference
https://www.omg.org/spec/ASCMM ASCMM-CWE-1080 Source Element Excessive Size

Roles
- the <Operation>
- the <SourceCode>

Detection Patterns
ASCQM Limit Size of Operations Code

7.1.27 CWE-1062 Parent Class Element with References to Child Class Element

Usage name
Parent class referencing child class

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1062 Parent Class Element with References to Child Class Element

Roles
- the <ParentClass>
- the <ChildClass>
- the <Reference>

Detection Patterns
ASCQM Ban Conversion References to Child Class

7.1.28 CWE-1087 Class Element with Virtual Method Element without Virtual Destructor

Usage name
Class with virtual method missing destructor

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1087 Class Element with Virtual Method Element without Virtual Destructor

Roles
- the <Class>
- the <VirtualMethod>

Detection Patterns
ASCQM Implement Virtual Destructor for Classes with Virtual Methods

7.1.29 CWE-1079 Parent Class Element without Virtual Destructor Method Element

Usage name
Parent class missing virtual destructor

Reference

https://www.omg.org/spec/ASCRM ASCRM-CWE-1079 Parent Class Element without Virtual Destructor Method Element

Roles
- the <ParentClass>

Detection Patterns
ASCQM Implement Virtual Destructor for Parent Classes

7.1.30 CWE-1045 Child Class Element without Virtual Destructor unlike its Parent Class Element

Usage name
Child class missing virtual destructor

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1045 Child Class Element without Virtual Destructor unlike its Parent Class Element

Roles
- the <ParentClass>
- the <ParentClassVirtualDestructor>
- the <ChildClass>

Detection Patterns
ASCQM Implement Virtual Destructor for Classes Derived from Class with Virtual Destructor

7.1.31 CWE-1051 Storable and Member Data Element Initialization with Hard-Coded Network Resource Configuration Data

Usage name
Hard-coded network resource information

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1051 Storable and Member Data Element Initialization with Hard-Coded Network Resource Configuration Data

Roles
- the <NetworkResourceAccess>
- the <HardCodedValue>

Detection Patterns
ASCQM Ban Hard-Coded Literals used to Connect to Resource

7.1.32 Maintainability detection patterns

Detection Patterns

ASCQM Ban Assignment Operation Inside Logic Blocks
ASCQM Ban Circular Dependencies between Modules
ASCQM Ban Comparison Expression Outside Logic Blocks
ASCQM Ban Control Flow Transfer
ASCQM Ban Conversion References to Child Class
ASCQM Ban Exception Definition without Ever Throwing It
ASCQM Ban Excessive Number of Children
ASCQM Ban Excessive Number of Concrete Implementations to Inherit From
ASCQM Ban Excessive Number of Inheritance Levels
ASCQM Ban Hard-Coded Literals used to Connect to Resource
ASCQM Ban Hard-Coded Literals used to Initialize Variables
ASCQM Ban Incorrect Joint Comparison
ASCQM Ban Incorrect Object Comparison
ASCQM Ban Incorrect String Comparison
ASCQM Ban Logical Dead Code
ASCQM Ban Logical Operation with a Constant Operand
ASCQM Ban Loop Value Update within Incremental and Decremental Loop
ASCQM Ban Not Operator On Non-Boolean Operand Of Comparison Operation
ASCQM Ban Not Operator On Operand Of Bitwise Operation
ASCQM Ban Public Data Elements
ASCQM Ban Switch in Switch Statement
ASCQM Ban Unintended Paths
ASCQM Ban Unreferenced Dead Code
ASCQM Ban Usage of Data Elements from Other Classes
ASCQM Check Boolean Variables are Updated in Different Conditional Branches before Use
ASCQM Implement Virtual Destructor for Classes Derived from Class with Virtual Destructor
ASCQM Implement Virtual Destructor for Classes with Virtual Methods
ASCQM Implement Virtual Destructor for Parent Classes
ASCQM Limit Algorithmic Complexity via Cyclomatic Complexity Value
ASCQM Limit Algorithmic Complexity via Essential Complexity Value
ASCQM Limit Algorithmic Complexity via Module Design Complexity Value
ASCQM Limit Number of Data Access
ASCQM Limit Number of Outward Calls
ASCQM Limit Number of Parameters
ASCQM Limit Size of Operations Code
ASCQM Limit Volume of Commented-Out Code
ASCQM Limit Volume of Similar Code
ASCQM Use Break in Switch Statement
ASCQM Use Default Case in Switch Statement

7.2 Weakness Category Performance Efficiency

7.2.1 CWE-401 Improper Release of Memory Before Removing Last Reference ('Memory Leak')

Reference
https://cwe.mitre.org/data/definitions/401 Improper Release of Memory Before Removing Last Reference ('Memory Leak')

Roles
- the <MemoryAllocation>

Parent weaknesses
CWE-404 Improper Resource Shutdown or Release

Detection Patterns
ASCQM Ban Comma Operator from Delete Statement
ASCQM Implement Required Operations for Manual Resource Management
ASCQM Release Memory After Use
ASCQM Release Memory after Use with Correct Operation
ASCQM Release Platform Resource after Use
ASCQM Release in Destructor Memory Allocated in Constructor

7.2.2 Weakness CWE-404 Improper Resource Shutdown or Release

Reference
https://cwe.mitre.org/data/definitions/404 Improper Resource Shutdown or Release

Roles
- the <ResourceAllocation>

Contributing weaknesses
CWE-401 Improper Release of Memory Before Removing Last Reference ('Memory Leak')
CWE-772 Missing Release of Resource after Effective Lifetime
CWE-775 Missing Release of File Descriptor or Handle after Effective Lifetime

Detection Patterns
ASCQM Ban Comma Operator from Delete Statement
ASCQM Implement Virtual Destructor for Classes Derived from Class with Virtual Destructor
ASCQM Implement Virtual Destructor for Classes with Virtual Methods
ASCQM Implement Virtual Destructor for Parent Classes
ASCQM Release File Resource after Use in Operation
ASCQM Release Platform Resource after Use
ASCQM Release in Destructor Memory Allocated in Constructor

7.2.3 CWE-424 Improper Protection of Alternate Path

Reference
https://cwe.mitre.org/data/definitions/424 Improper Protection of Alternate Path

Roles
- the <AlternatePath>

Detection Patterns
ASCQM Ban Unintended Paths

7.2.4 CWE-772 Missing Release of Resource after Effective Lifetime

Reference
https://cwe.mitre.org/data/definitions/772 Missing Release of Resource after Effective Lifetime

Roles
- the <ResourceAllocation>

Parent weaknesses
CWE-404 Improper Resource Shutdown or Release

Detection Patterns
ASCQM Release File Resource after Use in Operation
ASCQM Release Platform Resource after Use
ASCQM Release in Destructor Memory Allocated in Constructor

7.2.5 CWE-775 Missing Release of File Descriptor or Handle after Effective Lifetime

Reference
https://cwe.mitre.org/data/definitions/775 Missing Release of File Descriptor or Handle after Effective Lifetime

Roles
- the <FileDescriptorOrHandleAllocation>

Parent weaknesses
Weakness CWE-775 Missing Release of File Descriptor or Handle after Effective Lifetime

Detection Patterns
ASCQM Release File Resource after Use in Class
ASCQM Release File Resource after Use in Operation

7.2.6 CWE-1073 Non-SQL Named Callable and Method Control Element with Excessive Number of Data Resource Access

Usage name
Excessive data queries in client-side code

Reference
https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1073 Non-SQL Named Callable and Method Control Element with Excessive Number of Data Resource Access

Roles
- the <NonSQLOperation>
- the <DataAccesses>

Detection Patterns
ASCQM Ban Excessive Number of Data Resource Access from non-SQL Code

7.2.7 CWE-1057 Data Access Control Element from Outside Designated Data Manager Component

Usage name
Circumventing data access routines

Reference
https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1057 Data Access Control Element from Outside Designated Data Manager Component

Roles
- the <DataManager>
- the <DataAccess>

Detection Patterns
ASCQM Ban Unintended Paths

7.2.8 CWE-1043 Storable and Member Data Element Excessive Number of Aggregated Storable and Member Data Elements

Usage name
Excessively large data element

Reference
https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1043 Storable and Member Data Element Excessive Number of Aggregated Storable and Member Data Elements

Roles
- the <AggregationData>
- the <AggregatedData>

Detection Patterns
ASCQM Limit Number of Aggregated Non-Primitive Data Types

7.2.9 CWE-1072 Data Resource Access not using Connection Pooling Capability

Usage name
Data access not using connection pool

Reference
https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1072 Data Resource Access not using Connection Pooling Capability

Roles
- the <Connection>

Detection Patterns
ASCQM Ban Use of Prohibited Low-Level Resource Management Functionality

7.2.10 CWE-1071 Storable and Member Data Element Memory Allocation Missing De-allocation Control Element

Usage name
Unreleased data

Reference
https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1071 Storable and Member Data Element Memory Allocation Missing De-allocation Control Element

Roles
- the <MemoryAllocation>

Detection Patterns
ASCQM Release Memory after Use with Correct Operation

7.2.11 CWE-1091 Storable and Member Data Element Reference Missing De-referencing Control Element

Reference

https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1091 Storable and Member Data Element Reference Missing De-referencing Control Element

Roles
- the <Object>

Detection Patterns
ASCQM Release Memory after Use with Correct Operation

7.2.12 CWE-1046 Immutable Storable and Member Data Element Creation

Usage name
Immutable text data

Reference
https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1046 Immutable Storable and Member Data Element Creation

Roles
- the <ImmutableDataCreation>

Detection Patterns
ASCQM Ban Incrementral Creation of Immutable Data

7.2.13 CWE-1042 Static Member Data Element outside of a Singleton Class Element

Usage name
Static data outside of singleton class

Reference
https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1042 Static Member Data Element outside of a Singleton Class Element

Roles
- the <StaticDataDeclaration>

Detection Patterns
ASCQM Ban Static Non-Final Data Element Outside Singleton

7.2.14 CWE-1049 Data Resource Read and Write Access Excessive Complexity

Usage name
Complex read/write access

Reference
https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1049 Data Resource Read and Write Access Excessive Complexity

Roles
- the <DataQuery>

Detection Patterns
ASCQM Ban Excessive Complexity of Data Resource Access

7.2.15 CWE-1067 Data Resource Read Access Unsupported by Index Element

Usage name Incorrect indices

Reference
https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1067 Data Resource Read Access Unsupported by Index Element

Roles
- the <DataQuery>
- the <TableOrView>

Detection Patterns
ASCQM Implement Index Required by Query on Large Tables

7.2.16 CWE-1089 Large Data Resource ColumnSet Excessive Number of Index Elements

Usage name
Excessive number of indices on large tables

Reference
https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1089 Large Data Resource ColumnSet Excessive Number of Index Elements

Roles
- the <Table>
- the <Indexes>

Detection Patterns
ASCQM Ban Excessive Number of Index on Columns of Large Tables

7.2.17 CWE-1094 Large Data Resource ColumnSet with Index Element of Excessive Size

Usage name

Excessively large indices on large tables

Reference
https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1094 Large Data Resource ColumnSet with Index Element of Excessive Size

Roles
- the <Table>
- the <Indexes>

Detection Patterns
ASCQM Ban Excessive Size of Index on Columns of Large Tables

7.2.18 CWE-1050 Control Elements Requiring Significant Resource Element within Control Flow Loop Block

Usage name
Resource consuming operation in loop

Reference
https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1050 Control Elements Requiring Significant Resource Element within Control Flow Loop Block

Roles
- the <Loop>
- the <ExpensiveOperation>

Detection Patterns
ASCQM Ban Expensive Operations in Loops

7.2.19 CWE-1060 Non-stored SQL Callable Control Element with Excessive Number of Data Resource Access

Usage name
Excessive data queries in non-stored procedure

Reference
https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1060 Non-stored SQL Callable Control Element with Excessive Number of Data Resource Access

Roles
- the <NonStoredSQLOperation>
- the <DataAccesses>

Detection Patterns

ASCQM Ban Excessive Number of Data Resource Access from non-stored SQL Procedure

7.2.20 Performance Efficiency detection patterns

Detection Patterns

ASCQM Ban Comma Operator from Delete Statement
ASCQM Ban Excessive Complexity of Data Resource Access
ASCQM Ban Excessive Number of Data Resource Access from non-SQL Code
ASCQM Ban Excessive Number of Data Resource Access from non-stored SQL Procedure
ASCQM Ban Excessive Number of Index on Columns of Large Tables
ASCQM Ban Excessive Size of Index on Columns of Large Tables
ASCQM Ban Expensive Operations in Loops
ASCQM Ban Incrementral Creation of Immutable Data
ASCQM Ban Static Non-Final Data Element Outside Singleton
ASCQM Ban Unintended Paths
ASCQM Ban Use of Prohibited Low-Level Resource Management Functionality
ASCQM Implement Index Required by Query on Large Tables
ASCQM Implement Required Operations for Manual Resource Management
ASCQM Implement Virtual Destructor for Classes Derived from Class with Virtual Destructor
ASCQM Implement Virtual Destructor for Classes with Virtual Methods
ASCQM Implement Virtual Destructor for Parent Classes
ASCQM Limit Number of Aggregated Non-Primitive Data Types
ASCQM Release File Resource after Use in Class
ASCQM Release File Resource after Use in Operation
ASCQM Release Memory After Use
ASCQM Release Memory after Use with Correct Operation
ASCQM Release Platform Resource after Use
ASCQM Release in Destructor Memory Allocated in Constructor

7.3 Weakness Category Reliability

7.3.1 CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Reference
https://cwe.mitre.org/data/definitions/119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Roles
- the <BufferOperation>

Contributing weaknesses
CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
CWE-123 Write-what-where Condition
CWE-125 Out-of-bounds Read
CWE-130 Improper Handling of Length Parameter Inconsistency
CWE-786 Access of Memory Location Before Start of Buffer
CWE-787 Out-of-bounds Write
CWE-788 Access of Memory Location After End of Buffer
CWE-805 Buffer Access with Incorrect Length Value
CWE-822 Untrusted Pointer Dereference
CWE-823 Use of Out-of-range Pointer Offset
CWE-824 Access of Uninitialized Pointer
CWE-825 Expired Pointer Dereference

Detection Patterns
ASCQM Ban Input Acquisition Primitives without Boundary Checking Capabilities
ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities
ASCQM Ban Use of Expired Pointer
ASCQM Check Index of Array Access
ASCQM Check Input of Memory Manipulation Primitives
ASCQM Check Input of String Manipulation Primitives with Boundary Checking Capabilities
ASCQM Check Offset used in Pointer Arithmetic
ASCQM Initialize Pointers before Use
ASCQM Sanitize User Input used as Pointer

7.3.2 CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Reference
https://cwe.mitre.org/data/definitions/120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Roles
- the <BufferCopy>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Ban Input Acquisition Primitives without Boundary Checking Capabilities
ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities

7.3.3 CWE-123 Write-what-where Condition

Reference
https://cwe.mitre.org/data/definitions/123 Write-what-where Condition

Roles
- the <BufferWrite>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities

7.3.4 CWE-125 Out-of-bounds Read

Reference
https://cwe.mitre.org/data/definitions/125 Out-of-bounds Read

Roles
- the <BufferRead>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Check Index of Array Access

7.3.5 CWE-130 Improper Handling of Length Parameter Inconsistency

Reference
https://cwe.mitre.org/data/definitions/130 Improper Handling of Length Parameter Inconsistency

Roles
- the <DataHandling>
- the <LengthParameter>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Check Index of Array Access

7.3.6 CWE-131 Incorrect Calculation of Buffer Size

Reference
https://cwe.mitre.org/data/definitions/131 Incorrect Calculation of Buffer Size

Roles
- the <BufferSizeCalculation>

Parent weaknesses
CWE-682 Incorrect Calculation

Detection Patterns
ASCQM Ban Buffer Size Computation Based on Array Element Pointer Size
ASCQM Ban Buffer Size Computation Based on Bitwise Logical Operation
ASCQM Ban Buffer Size Computation Based on Incorrect String Length Value

7.3.7 CWE-170 Improper Null Termination

Reference
https://cwe.mitre.org/data/definitions/170 Improper Null Termination

Roles
- the <BufferWithoutNULLTermination>

Detection Patterns
ASCQM NULL Terminate Output Of String Manipulation Primitives

7.3.8 CWE-194 Unexpected Sign Extension

Reference
https://cwe.mitre.org/data/definitions/194 Unexpected Sign Extension

Roles
- the <NumberSignExtension>

Parent weaknesses
CWE-681 Incorrect Conversion between Numeric Types

Detection Patterns
ASCQM Ban Incorrect Numeric Implicit Conversion

7.3.9 CWE-195 Signed to Unsigned Conversion Error

Reference
https://cwe.mitre.org/data/definitions/195 Signed to Unsigned Conversion Error

Roles
- the <NumberConversionToUnsigned>

Parent weaknesses
CWE-681 Incorrect Conversion between Numeric Types

Detection Patterns
ASCQM Ban Incorrect Numeric Implicit Conversion

7.3.10 CWE-196 Unsigned to Signed Conversion Error

Reference
https://cwe.mitre.org/data/definitions/196 Unsigned to Signed Conversion Error

Roles
- the <NumberConversionToSigned>

Parent weaknesses
CWE-681 Incorrect Conversion between Numeric Types

Detection Patterns
ASCQM Ban Incorrect Numeric Implicit Conversion

7.3.11 CWE-197 Numeric Truncation Error

Reference
https://cwe.mitre.org/data/definitions/197 Numeric Truncation Error

Roles
- the <NumberTruncation>

Parent weaknesses
CWE-681 Incorrect Conversion between Numeric Types

Detection Patterns

ASCQM Ban Incorrect Numeric Implicit Conversion

7.3.12 CWE-248 Uncaught Exception

Reference
https://cwe.mitre.org/data/definitions/248 Uncaught Exception

Roles
- the <ExceptionThrowDeclaration>
- the <ExceptionCatchSequence>

Parent weaknesses
CWE-703 Improper Check or Handling of Exceptional Conditions

Detection Patterns
ASCQM Catch Exceptions

7.3.13 CWE-252 Unchecked Return Value

Reference
https://cwe.mitre.org/data/definitions/252 Unchecked Return Value

Roles
- the <OperationCall>

Detection Patterns
ASCQM Check Return Value of Resource Operations Immediately
ASCQM Handle Return Value of Must Check Operations

7.3.14 CWE-366 Race Condition within a Thread

Reference
https://cwe.mitre.org/data/definitions/366 Race Condition within a Thread

Roles
- the <Thread1>
- the <Thread2>
- the <ConflictingResource>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Creation of Lock On Private Non-Static Object to Access Private Static Data

ASCQM Data Read and Write without Proper Locking in Multi-Threaded Context

7.3.15 CWE-369 Divide By Zero

Reference
https://cwe.mitre.org/data/definitions/369 Divide By Zero

Roles
- the <Division>

Parent weaknesses
CWE-682 Incorrect Calculation

Detection Patterns
ASCQM Check and Handle ZERO Value before Use as Divisor

7.3.16 CWE-390 Detection of Error Condition Without Action

Reference
https://cwe.mitre.org/data/definitions/390 Detection of Error Condition Without Action

Roles
- the <ErrorCondition>

Detection Patterns
ASCQM Ban Empty Exception Block
ASCQM Handle Return Value of Resource Operations

7.3.17 CWE-391 Unchecked Error Condition

Reference
https://cwe.mitre.org/data/definitions/391 Unchecked Error Condition

Roles
- the <ErrorConditionProcessing>

Parent weaknesses
CWE-703 Improper Check or Handling of Exceptional Conditions

Detection Patterns
ASCQM Ban Empty Exception Block
ASCQM Ban Useless Handling of Exceptions

7.3.18 CWE-392 Missing Report of Error Condition

Reference
https://cwe.mitre.org/data/definitions/392 Missing Report of Error Condition

Roles
- the <ErrorConditionProcessing>

Parent weaknesses
CWE-703 Improper Check or Handling of Exceptional Conditions

Detection Patterns
ASCQM Ban Useless Handling of Exceptions

7.3.19 CWE-394 Unexpected Status Code or Return Value

Reference
https://cwe.mitre.org/data/definitions/394 Unexpected Status Code or Return Value

Roles
- the <ReturnValue>

Detection Patterns
ASCQM Ban Incorrect Numeric Conversion of Return Value
ASCQM Handle Return Value of Must Check Operations
ASCQM Handle Return Value of Resource Operations

7.3.20 CWE-401 Improper Release of Memory Before Removing Last Reference ('Memory Leak')

Reference
https://cwe.mitre.org/data/definitions/401 Improper Release of Memory Before Removing Last Reference ('Memory Leak')

Roles
- the <MemoryAllocation>

Parent weaknesses
CWE-404 Improper Resource Shutdown or Release

Detection Patterns
ASCQM Ban Comma Operator from Delete Statement
ASCQM Implement Required Operations for Manual Resource Management
ASCQM Release Memory After Use
ASCQM Release Memory after Use with Correct Operation

ASCQM Release Platform Resource after Use
ASCQM Release in Destructor Memory Allocated in Constructor

7.3.21 CWE-404 Improper Resource Shutdown or Release

Reference
https://cwe.mitre.org/data/definitions/404 Improper Resource Shutdown or Release

Roles
- the <ResourceAllocation>

Contributing weaknesses
CWE-401 Improper Release of Memory Before Removing Last Reference ('Memory Leak')
CWE-772 Missing Release of Resource after Effective Lifetime
CWE-775 Missing Release of File Descriptor or Handle after Effective Lifetime

Detection Patterns
ASCQM Ban Comma Operator from Delete Statement
ASCQM Implement Virtual Destructor for Classes Derived from Class with Virtual Destructor
ASCQM Implement Virtual Destructor for Classes with Virtual Methods
ASCQM Implement Virtual Destructor for Parent Classes
ASCQM Release File Resource after Use in Operation
ASCQM Release Platform Resource after Use
ASCQM Release in Destructor Memory Allocated in Constructor

7.3.22 CWE-415 Double Free

Reference
https://cwe.mitre.org/data/definitions/415 Double Free

Roles
- the <ResourceRelease >
- the <ResourceAccess>
- the <ResourceUse>

Parent weaknesses
CWE-672 Operation on a Resource after Expiration or Release

Detection Patterns
ASCQM Ban Double Free On Pointers

7.3.23 CWE-416 Use After Free

Reference

https://cwe.mitre.org/data/definitions/416 Use After Free

Roles
- the <ResourceRelease>
- the <ResourceUse>

Parent weaknesses
CWE-672 Operation on a Resource after Expiration or Release

Detection Patterns
ASCQM Ban Free Operation on Pointer Received as Parameter
ASCQM Ban Use of Expired Pointer
ASCQM Implement Copy Constructor for Class With Pointer Resource

7.3.24 CWE-424 Improper Protection of Alternate Path

Reference
https://cwe.mitre.org/data/definitions/424 Improper Protection of Alternate Path

Roles
- the <AlternatePath>

Detection Patterns
ASCQM Ban Unintended Paths

7.3.25 CWE-456 Missing Initialization of a Variable

Reference
https://cwe.mitre.org/data/definitions/456 Missing Initialization of a Variable

Roles
- the <VariableDeclaration>

Parent weaknesses
CWE-665 Improper Initialization

Detection Patterns
ASCQM Ban Allocation of Memory with Null Size
ASCQM Initialize Variables

7.3.26 CWE-459 Incomplete Cleanup

Reference
https://cwe.mitre.org/data/definitions/459 Incomplete Cleanup

Roles
- the <ResourceAllocation>
- the <ResourceRelease>

Detection Patterns
ASCQM Release Memory after Use with Correct Operation

7.3.27 CWE-476 NULL Pointer Dereference

Reference
https://cwe.mitre.org/data/definitions/476 NULL Pointer Dereference

Roles
- the <PointerDereferencing>

Detection Patterns
ASCQM Check NULL Pointer Value before Use

7.3.28 CWE-480 Use of Incorrect Operator

Reference
https://cwe.mitre.org/data/definitions/480 Use of Incorrect Operator

Roles
- the <Operator>

Detection Patterns
ASCQM Ban Assignment Operation Inside Logic Blocks
ASCQM Ban Comparison Expression Outside Logic Blocks
ASCQM Ban Incorrect Object Comparison
ASCQM Ban Incorrect String Comparison
ASCQM Ban Logical Operation with a Constant Operand

7.3.29 CWE-484 Omitted Break Statement in Switch

Reference
https://cwe.mitre.org/data/definitions/484 Omitted Break Statement in Switch

Roles
- the <SwitchStatement>

Detection Patterns
ASCQM Use Break in Switch Statement

7.3.30 CWE-543 Use of Singleton Pattern Without Synchronization in a Multithreaded Context

Reference
https://cwe.mitre.org/data/definitions/543 Use of Singleton Pattern Without Synchronization in a Multithreaded Context

Roles
- the <SingletonUse>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Non-Final Static Data in Multi-Threaded Context
ASCQM Singleton Creation without Proper Locking in Multi-Threaded Context

7.3.31 CWE-562 Return of Stack Variable Address

Reference
https://cwe.mitre.org/data/definitions/562 Return of Stack Variable Address

Roles
- the <ReturnStatement>

Detection Patterns
ASCQM Ban Return of Local Variable Address
ASCQM Ban Storage of Local Variable Address in Global Variable

7.3.32 CWE-567 Unsynchronized Access to Shared Data in a Multithreaded Context

Reference
https://cwe.mitre.org/data/definitions/567 Unsynchronized Access to Shared Data in a Multithreaded Context

Roles
- the <SharedDataAccess>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Non-Final Static Data in Multi-Threaded Context

ASCQM Data Read and Write without Proper Locking in Multi-Threaded Context

7.3.33 CWE-595 Comparison of Object References Instead of Object Contents

Reference
https://cwe.mitre.org/data/definitions/595 Comparison of Object References Instead of Object Contents

Roles
- the <ObjectReferencesComparison>

Contributing weaknesses
CWE-597 Use of Wrong Operator in String Comparison
RLB-4 Persistent Storable Data Element without Proper Comparison Control Element

Detection Patterns
ASCQM Ban Incorrect Object Comparison
ASCQM Ban Incorrect String Comparison
ASCQM Implement Correct Object Comparison Operations

7.3.34 CWE-597 Use of Wrong Operator in String Comparison

Reference
https://cwe.mitre.org/data/definitions/597 Use of Wrong Operator in String Comparison

Roles
- the <StringComparison>

Parent weaknesses
CWE-595 Comparison of Object References Instead of Object Contents

Detection Patterns
ASCQM Ban Incorrect String Comparison

7.3.35 CWE-662 Improper Synchronization

Reference
https://cwe.mitre.org/data/definitions/662 Improper Synchronization

Roles
- the <Thread1>
- the <Thread2>
- the <SharedResourceAccess>

Contributing weaknesses
CWE-366 Race Condition within a Thread
CWE-543 Use of Singleton Pattern Without Synchronization in a Multithreaded Context
CWE-567 Unsynchronized Access to Shared Data in a Multithreaded Context
CWE-667 Improper Locking
CWE-764 Multiple Locks of a Critical Resource
CWE-820 Missing Synchronization
CWE-821 Incorrect Synchronization
CWE-833 Deadlock
RLB-11 Named Callable and Method Control Element in Multi-Thread Context with non-Final Static Storable or Member Element
RLB-12 Singleton Class Instance Creation without Proper Lock Element Management

Detection Patterns

7.3.36 CWE-667 Improper Locking

Reference
Reference https://cwe.mitre.org/data/definitions/667 Improper Locking
Roles
Roles:
- the <Thread1>
- the <Thread2>
- the <SharedResourceAccess>
- the <Lock>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Incorrect Synchronization Mechanisms
ASCQM Ban Non-Final Static Data in Multi-Threaded Context
ASCQM Ban Resource Access without Proper Locking in Multi-Threaded Context
ASCQM Data Read and Write without Proper Locking in Multi-Threaded Context
ASCQM Singleton Creation without Proper Locking in Multi-Threaded Context

7.3.37 CWE-672 Operation on a Resource after Expiration or Release

Reference
https://cwe.mitre.org/data/definitions/672 Operation on a Resource after Expiration or Release

Roles
- the <ResourceRelease>
- the <ResourceAccess>

Contributing weaknesses
CWE-415 Double Free
CWE-416 Use After Free

Detection Patterns
ASCQM Ban Double Release of Resource
ASCQM Ban Use of Expired Resource

7.3.38 CWE-681 Incorrect Conversion between Numeric Types

Reference
https://cwe.mitre.org/data/definitions/681 Incorrect Conversion between Numeric Types

Roles
- the <NumericConversion>

Contributing weaknesses
CWE-194 Unexpected Sign Extension
CWE-195 Signed to Unsigned Conversion Error
CWE-196 Unsigned to Signed Conversion Error
CWE-197 Numeric Truncation Error

Detection Patterns
ASCQM Ban Incorrect Numeric Implicit Conversion

7.3.39 CWE-682 Incorrect Calculation

Reference
https://cwe.mitre.org/data/definitions/682 Incorrect Calculation

Roles
- the <Calculation>

Contributing weaknesses
CWE-131 Incorrect Calculation of Buffer Size
CWE-369 Divide By Zero

Detection Patterns

7.3.40 CWE-703 Improper Check or Handling of Exceptional Conditions

Reference
https://cwe.mitre.org/data/definitions/703 Improper Check or Handling of Exceptional Conditions

Roles
- the <ErrorHandling>

Contributing weaknesses
CWE-166 Improper Handling of Missing Special Element
CWE-167 Improper Handling of Additional Special Element
CWE-168 Improper Handling of Inconsistent Special Elements
CWE-228 Improper Handling of Syntactically Invalid Structure
CWE-248 Uncaught Exception
CWE-280 Improper Handling of Insufficient Permissions or Privileges
CWE-391 Unchecked Error Condition
CWE-392 Missing Report of Error Condition
CWE-393 Return of Wrong Status Code
CWE-754 Improper Check for Unusual or Exceptional Conditions
CWE-755 Improper Handling of Exceptional Conditions

Detection Patterns
ASCQM Ban Useless Handling of Exceptions

7.3.41 CWE-704 Incorrect Type Conversion or Cast

Reference
https://cwe.mitre.org/data/definitions/704 Incorrect Type Conversion or Cast

Roles
- the <TypeConversion>

Contributing weaknesses
CWE-843 Access of Resource Using Incompatible Type ('Type Confusion')

Detection Patterns
ASCQM Ban Incorrect Type Conversion

7.3.42 CWE-758 Reliance on Undefined, Unspecified, or Implementation-Defined Behavior

Reference
https://cwe.mitre.org/data/definitions/758 Reliance on Undefined, Unspecified, or Implementation-Defined Behavior

Roles
- the <Statement>

Detection Patterns
ASCQM Ban Delete of VOID Pointer

ASCQM Ban Reading and Writing the Same Variable Used as Assignment Value
ASCQM Ban Variable Increment or Decrement Operation in Operations using the Same Variable

7.3.43 CWE-764 Multiple Locks of a Critical Resource

Reference
https://cwe.mitre.org/data/definitions/764 Multiple Locks of a Critical Resource

Roles
- the <Lock1>
- the <Lock2>
- the <Resource>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Sequential Acquisitions of Single Non-Reentrant Lock

7.3.44 CWE-772 Missing Release of Resource after Effective Lifetime

Reference
https://cwe.mitre.org/data/definitions/772 Missing Release of Resource after Effective Lifetime

Roles
- the <ResourceAllocation>

Parent weaknesses
CWE-404 Improper Resource Shutdown or Release

Detection Patterns
ASCQM Release File Resource after Use in Operation
ASCQM Release Platform Resource after Use
ASCQM Release in Destructor Memory Allocated in Constructor

7.3.45 CWE-775 Missing Release of File Descriptor or Handle after Effective Lifetime

Reference
https://cwe.mitre.org/data/definitions/775 Missing Release of File Descriptor or Handle after Effective Lifetime

Roles
- the <FileDescriptorOrHandleAllocation>

Parent weaknesses
CWE-404 Improper Resource Shutdown or Release

Detection Patterns
ASCQM Release File Resource after Use in Class
ASCQM Release File Resource after Use in Operation

7.3.46 CWE-786 Access of Memory Location Before Start of Buffer

Reference
https://cwe.mitre.org/data/definitions/786 Access of Memory Location Before Start of Buffer

Roles
- the <MemoryAccess>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Check Index of Array Access
ASCQM Check Input of String Manipulation Primitives with Boundary Checking Capabilities

7.3.47 CWE-787 Out-of-bounds Write

Reference
https://cwe.mitre.org/data/definitions/787 Out-of-bounds Write

Roles
- the <BufferWrite>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Check Index of Array Access
ASCQM Check Input of Memory Manipulation Primitives

7.3.48 CWE-788 Access of Memory Location After End of Buffer

Reference
https://cwe.mitre.org/data/definitions/788 Access of Memory Location After End of Buffer

Roles
- the <MemoryAccess>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities
ASCQM Check Index of Array Access
ASCQM Check Input of Memory Manipulation Primitives

7.3.49 CWE-805 Buffer Access with Incorrect Length Value

Reference
https://cwe.mitre.org/data/definitions/805 Buffer Access with Incorrect Length Value

Roles
- the <BufferAccess>
- the <LengthParameter>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities
ASCQM Check Input of Memory Manipulation Primitives
ASCQM Check Input of String Manipulation Primitives with Boundary Checking Capabilities

7.3.50 CWE-820 Missing Synchronization

Reference
https://cwe.mitre.org/data/definitions/820 Missing Synchronization

Roles
- the <SharedResourceUse>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Resource Access without Proper Locking in Multi-Threaded Context

7.3.51 CWE-821 Incorrect Synchronization

Reference
https://cwe.mitre.org/data/definitions/821 Incorrect Synchronization

Roles
- the <SharedResourceUse>
- the <IncorrectSynchronization>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Incorrect Synchronization Mechanisms

7.3.52 CWE-822 Untrusted Pointer Dereference

Reference
https://cwe.mitre.org/data/definitions/822 Untrusted Pointer Dereference

Roles
- the <PointerDereferencing>
- the <TaintedInput>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Sanitize User Input used as Pointer

7.3.53 CWE-823 Use of Out-of-range Pointer Offset

Reference
https://cwe.mitre.org/data/definitions/823 Use of Out-of-range Pointer Offset

Roles
- the <PointerOffset>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Check Offset used in Pointer Arithmetic

7.3.54 CWE-824 Access of Uninitialized Pointer

Reference
Reference https://cwe.mitre.org/data/definitions/824 Access of Uninitialized Pointer

Roles
Roles:
- the <PointerAccess>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Initialize Pointers before Use

7.3.55 CWE-825 Expired Pointer Dereference

Reference
https://cwe.mitre.org/data/definitions/825 Expired Pointer Dereference

Roles
- the <PointerAccess>
- the <PointerRelease>

Parent weaknesses
CWE-672 Operation on a Resource after Expiration or Release

Detection Patterns
ASCQM Ban Use of Expired Pointer

7.3.56 CWE-833 Deadlock

Reference
https://cwe.mitre.org/data/definitions/833 Deadlock

Roles
- the <Thread1>
- the <Thread2>
- the <ConflictingLock>

Parent weaknesses
Weakness CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Incompatible Lock Acquisition Sequences
ASCQM Ban Use of Thread Control Primitives with Known Deadlock Issues

7.3.57 CWE-835 Loop with Unreachable Exit Condition ('Infinite Loop')

Reference
https://cwe.mitre.org/data/definitions/835 Loop with Unreachable Exit Condition ('Infinite Loop')

Roles
- the <InfiniteLoop>

Detection Patterns
ASCQM Ban Unmodified Loop Variable Within Loop
ASCQM Ban While TRUE Loop Without Path To Break

7.3.58 CWE-908 Use of Uninitialized Resource

Reference
https://cwe.mitre.org/data/definitions/908 Use of Uninitialized Resource

Roles
- the <ResourceUse>

Detection Patterns
ASCQM Initialize Resource before Use

7.3.59 CWE-1083 Data Access Control Element from Outside Designated Data Manager Component

Usage name
Circumventing data access routines

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1083 Data Access Control Element from Outside Designated Data Manager Component

Roles
- the <DataManager>
- the <DataAccess>

Detection Patterns
ASCQM Ban Unintended Paths

7.3.60 CWE-1058 Named Callable and Method Control Element in Multi-Thread Context with non-Final Static Storable or Member Element

Usage name

Non-final static data in a multi-threaded environment

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1058 Named Callable and Method Control Element in Multi-Thread Context with non-Final Static Storable or Member Element

Roles
- the <Operation>
- the <NonFinalStaticData>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Non-Final Static Data in Multi-Threaded Context

7.3.61 CWE-1096 Singleton Class Instance Creation without Proper Lock Element Management

Usage name
Improper locking of singleton classes

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1096 Singleton Class Instance Creation without Proper Lock Element Management

Roles
- the <SingletonUse>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Singleton Creation without Proper Locking in Multi-Threaded Context

7.3.62 CWE-1087 Class Element with Virtual Method Element without Virtual Destructor

Usage name
Class with virtual method missing destructor

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1087 Class Element with Virtual Method Element without Virtual Destructor

Roles
- the <Class>
- the <VirtualMethod>

Detection Patterns
ASCQM Implement Virtual Destructor for Classes with Virtual Methods

7.3.63 CWE-1079 Parent Class Element without Virtual Destructor Method Element

Usage name
Parent class missing virtual destructor

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1079 Parent Class Element without Virtual Destructor Method Element

Roles
- the <ParentClass>

Detection Patterns
ASCQM Implement Virtual Destructor for Parent Classes

7.3.64 CWE-1045 Child Class Element without Virtual Destructor unlike its Parent Class Element

Usage name
Child class missing virtual destructor

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1045 Child Class Element without Virtual Destructor unlike its Parent Class Element

Roles
- the <ParentClass>
- the <ParentClassVirtualDestructor>
- the <ChildClass>

Detection Patterns
ASCQM Implement Virtual Destructor for Classes Derived from Class with Virtual Destructor

7.3.65 CWE-1051 Storable and Member Data Element Initialization with Hard-Coded Network Resource Configuration Data

Usage name
Hard-coded network resource information

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1051 Storable and Member Data Element Initialization with Hard-Coded Network Resource Configuration Data

Roles
- the <NetworkResourceAccess>
- the <HardCodedValue>

Detection Patterns
ASCQM Ban Hard-Coded Literals used to Connect to Resource

7.3.66 CWE-1088 Synchronous Call Time-Out Absence

Usage name
Synchronous call with missing timeout

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1088 Synchronous Call Time-Out Absence

Roles
- the <SynchronousCall>
- the <TimeOutOption>

Detection Patterns
ASCQM Manage Time-Out Mechanisms in Blocking Synchronous Calls

7.3.67 CWE-1066 Serializable Storable Data Element without Serialization Control Element

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1066 Serializable Storable Data Element without Serialization Control Element

Roles
- the <SerializableData>

Detection Patterns
ASCQM Ban Non-Serializable Elements in Serializable Objects

7.3.68 CWE-1070 Serializable Storable Data Element with non-Serializable Item Elements

Reference

https://www.omg.org/spec/ASCRM ASCRM-CWE-1070 Serializable Storable Data Element with non-Serializable Item Elements

Roles
- the <SerializableData>
- the <NonSerialibleChildData>

Detection Patterns
ASCQM Ban Non-Serializable Elements in Serializable Objects

7.3.69 CWE-1097 Persistent Storable Data Element without Proper Comparison Control Element

Usage name
Persistent data without proper comparison controls

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1097 Persistent Storable Data Element without Proper Comparison Control Element

Roles
- the <PersistentData>

Parent weaknesses
CWE-595 Comparison of Object References Instead of Object Contents

Detection Patterns
ASCQM Implement Correct Object Comparison Operations

7.3.70 CWE-1098 Storable or Member Data Element containing Pointer Item Element without Proper Copy Control Element

Usage name
Improper copy capabilities for data pointers

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1098 Storable or Member Data Element containing Pointer Item Element without Proper Copy Control Element

Roles
- the <ParentData>
- the <PointerChildData>

Detection Patterns
ASCQM Implement Copy Constructor for Class With Pointer Resource

7.3.71 CWE-1082 Class Instance Self Destruction Control Element

Usage name
Self-destruction

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1082 Class Instance Self Destruction Control Element

Roles
- the <SelfDestruction>

Detection Patterns
ASCQM Ban Self Destruction

7.3.72 CWE-1077 Float Type Storable and Member Data Element Comparison with Equality Operator

Usage name
Improper equality comparisons of float-type numerical data

Reference
https://www.omg.org/spec/ASCRM ASCRM-CWE-1077 Float Type Storable and Member Data Element Comparison with Equality Operator

Roles
- the <FloatNumberEqualityComparison>

Detection Patterns
ASCQM Ban Incorrect Float Number Comparison

7.3.73 CWE-665 Improper Initialization

Reference
https://cwe.mitre.org/data/definitions/665 Improper Initialization

Roles
- the <Initialization>

Contributing weaknesses
CWE-456 Missing Initialization of a Variable

CWE-457 Use of Uninitialized Variable

Detection Patterns
ASCQM Ban Self Assignment
ASCQM Initialize Pointers before Use
ASCQM Initialize Variables before Use

7.3.74 CWE-457 Use of Uninitialized Variable

Reference
https://cwe.mitre.org/data/definitions/457 Use of Uninitialized Variable

Roles
- the <VariableDeclaration>
- the <VariableUse>

Parent weaknesses
CWE-665 Improper Initialization

Detection Patterns
ASCQM Ban Allocation of Memory with Null Size
ASCQM Initialize Variables

7.3.75 Reliability detection patterns

Detection Patterns
ASCQM Ban Allocation of Memory with Null Size
ASCQM Ban Assignment Operation Inside Logic Blocks
ASCQM Ban Buffer Size Computation Based on Array Element Pointer Size
ASCQM Ban Buffer Size Computation Based on Bitwise Logical Operation
ASCQM Ban Buffer Size Computation Based on Incorrect String Length Value
ASCQM Ban Comma Operator from Delete Statement
ASCQM Ban Comparison Expression Outside Logic Blocks
ASCQM Ban Creation of Lock On Inappropriate Object Type
ASCQM Ban Creation of Lock On Non-Final Object
ASCQM Ban Creation of Lock On Private Non-Static Object to Access Private Static Data
ASCQM Ban Delete of VOID Pointer
ASCQM Ban Double Free On Pointers
ASCQM Ban Double Release of Resource
ASCQM Ban Empty Exception Block
ASCQM Ban Free Operation on Pointer Received as Parameter
ASCQM Ban Hard-Coded Literals used to Connect to Resource
ASCQM Ban Incompatible Lock Acquisition Sequences
ASCQM Ban Incorrect Float Number Comparison

ASCQM Ban Incorrect Numeric Conversion of Return Value
ASCQM Ban Incorrect Numeric Implicit Conversion
ASCQM Ban Incorrect Object Comparison
ASCQM Ban Incorrect String Comparison
ASCQM Ban Incorrect Synchronization Mechanisms
ASCQM Ban Incorrect Type Conversion
ASCQM Ban Input Acquisition Primitives without Boundary Checking Capabilities
ASCQM Ban Logical Operation with a Constant Operand
ASCQM Ban Non-Final Static Data in Multi-Threaded Context
ASCQM Ban Non-Serializable Elements in Serializable Objects
ASCQM Ban Reading and Writing the Same Variable Used as Assignment Value
ASCQM Ban Resource Access without Proper Locking in Multi-Threaded Context
ASCQM Ban Return of Local Variable Address
ASCQM Ban Self Destruction
ASCQM Ban Sequential Acquisitions of Single Non-Reentrant Lock
ASCQM Ban Sleep Between Lock Acquisition and Release
ASCQM Ban Storage of Local Variable Address in Global Variable
ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities
ASCQM Ban Unintended Paths
ASCQM Ban Unmodified Loop Variable Within Loop
ASCQM Ban Use of Expired Pointer
ASCQM Ban Use of Expired Resource
ASCQM Ban Use of Thread Control Primitives with Known Deadlock Issues
ASCQM Ban Useless Handling of Exceptions
ASCQM Ban Variable Increment or Decrement Operation in Operations using the Same Variable
ASCQM Ban While TRUE Loop Without Path To Break
ASCQM Catch Exceptions
ASCQM Check Index of Array Access
ASCQM Check Input of Memory Manipulation Primitives
ASCQM Check Input of String Manipulation Primitives with Boundary Checking Capabilities
ASCQM Check NULL Pointer Value before Use
ASCQM Check Offset used in Pointer Arithmetic
ASCQM Check Return Value of Resource Operations Immediately
ASCQM Check and Handle ZERO Value before Use as Divisor
ASCQM Data Read and Write without Proper Locking in Multi-Threaded Context
ASCQM Handle Return Value of Must Check Operations
ASCQM Handle Return Value of Resource Operations
ASCQM Implement Copy Constructor for Class With Pointer Resource
ASCQM Implement Correct Object Comparison Operations

7.4 Weakness Category Security

7.4.1 CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Reference
https://cwe.mitre.org/data/definitions/119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Roles
- the <BufferOperation>

Contributing weaknesses
CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
CWE-123 Write-what-where Condition
CWE-125 Out-of-bounds Read
CWE-130 Improper Handling of Length Parameter Inconsistency
CWE-786 Access of Memory Location Before Start of Buffer
CWE-787 Out-of-bounds Write
CWE-788 Access of Memory Location After End of Buffer
CWE-805 Buffer Access with Incorrect Length Value
CWE-822 Untrusted Pointer Dereference
CWE-823 Use of Out-of-range Pointer Offset
CWE-824 Access of Uninitialized Pointer
CWE-825 Expired Pointer Dereference

Detection Patterns
ASCQM Ban Input Acquisition Primitives without Boundary Checking Capabilities
ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities
ASCQM Ban Use of Expired Pointer
ASCQM Check Index of Array Access
ASCQM Check Input of Memory Manipulation Primitives
ASCQM Check Input of String Manipulation Primitives with Boundary Checking Capabilities
ASCQM Check Offset used in Pointer Arithmetic
ASCQM Initialize Pointers before Use
ASCQM Sanitize User Input used as Pointer

7.4.2 CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Reference
https://cwe.mitre.org/data/definitions/120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Roles
- the <BufferCopy>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Ban Input Acquisition Primitives without Boundary Checking Capabilities
ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities

7.4.3 CWE-123 Write-what-where Condition

Reference
https://cwe.mitre.org/data/definitions/123 Write-what-where Condition

Roles
- the <BufferWrite>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities

7.4.4 CWE-125 Out-of-bounds Read

Reference
https://cwe.mitre.org/data/definitions/125 Out-of-bounds Read

Roles
- the <BufferRead>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Check Index of Array Access

7.4.5 CWE-129 Improper Validation of Array Index

Reference
https://cwe.mitre.org/data/definitions/129 Improper Validation of Array Index

Roles
- the <ArrayAccess>
- the <TaintedIndex>

Detection Patterns
ASCQM Sanitize User Input used as Array Index

7.4.6 CWE-130 Improper Handling of Length Parameter Inconsistency

Reference
https://cwe.mitre.org/data/definitions/130 Improper Handling of Length Parameter Inconsistency

Roles
- the <DataHandling>
- the <LengthParameter>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Check Index of Array Access

7.4.7 CWE-131 Incorrect Calculation of Buffer Size

Reference
Reference https://cwe.mitre.org/data/definitions/131 Incorrect Calculation of Buffer Size

Roles
- the <BufferSizeCalculation>

Parent weaknesses
CWE-682 Incorrect Calculation

Detection Patterns
ASCQM Ban Buffer Size Computation Based on Array Element Pointer Size
ASCQM Ban Buffer Size Computation Based on Bitwise Logical Operation
ASCQM Ban Buffer Size Computation Based on Incorrect String Length Value

7.4.8 CWE-134 Use of Externally-Controlled Format String

Reference
https://cwe.mitre.org/data/definitions/134 Use of Externally-Controlled Format String

Roles
- the <Formatting>
- the <TaintedFormatString>

Detection Patterns
ASCQM Sanitize User Input used as String Format

7.4.9 CWE-194 Unexpected Sign Extension

Reference
https://cwe.mitre.org/data/definitions/194 Unexpected Sign Extension

Roles
- the <NumberSignExtension>

Parent weaknesses
CWE-681 Incorrect Conversion between Numeric Types

Detection Patterns
ASCQM Ban Incorrect Numeric Implicit Conversion

7.4.10 CWE-195 Signed to Unsigned Conversion Error

Reference
https://cwe.mitre.org/data/definitions/195 Signed to Unsigned Conversion Error

Roles
- the <NumberConversionToUnsigned>

Parent weaknesses
CWE-681 Incorrect Conversion between Numeric Types

Detection Patterns

7.4.11 CWE-196 Unsigned to Signed Conversion Error

Reference
https://cwe.mitre.org/data/definitions/196 Unsigned to Signed Conversion Error

Roles
- the <NumberConversionToSigned>

Parent weaknesses
CWE-681 Incorrect Conversion between Numeric Types

Detection Patterns
ASCQM Ban Incorrect Numeric Implicit Conversion

7.4.12 CWE-197 Numeric Truncation Error

Reference
https://cwe.mitre.org/data/definitions/197 Numeric Truncation Error

Roles
- the <NumberTruncation>

Parent weaknesses
CWE-681 Incorrect Conversion between Numeric Types

Detection Patterns
ASCQM Ban Incorrect Numeric Implicit Conversion

7.4.13 CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

Reference
https://cwe.mitre.org/data/definitions/22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

Roles
- the <PathManipulationStatement>
- the <TaintedInput>

Contributing weaknesses
CWE-23 Relative Path Traversal
CWE-36 Absolute Path Traversal

Detection Patterns
ASCQM Sanitize User Input used in Path Manipulation

7.4.14 CWE-23 Relative Path Traversal

Reference
https://cwe.mitre.org/data/definitions/23 Relative Path Traversal

Roles
- the <PathManipulation>
- the <TaintedInput>

Parent weaknesses
CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

Detection Patterns

ASCQM Sanitize User Input used in Path Manipulation

7.4.15 CWE-252 Unchecked Return Value

Reference
https://cwe.mitre.org/data/definitions/252 Unchecked Return Value

Roles
- the <OperationCall>

Detection Patterns
ASCQM Check Return Value of Resource Operations Immediately
ASCQM Handle Return Value of Must Check Operations

7.4.16 CWE-259 Use of Hard-coded Password

Reference
https://cwe.mitre.org/data/definitions/259 Use of Hard-coded Password

Roles
- the <Authentication>
- the <HardCodedValue>

Parent weaknesses
CWE-798 Use of Hard-coded Credentials

Detection Patterns
ASCQM Ban Hard-Coded Literals used to Connect to Resource

7.4.17 CWE-321 Use of Hard-coded Cryptographic Key

Reference
https://cwe.mitre.org/data/definitions/321 Use of Hard-coded Cryptographic Key

Roles
- the <Authentication>
- the <HardCodedCryptographicKey>

Parent weaknesses
CWE-798 Use of Hard-coded Credentials

Detection Patterns
ASCQM Ban Hard-Coded Literals used to Connect to Resource

7.4.18 CWE-36 Absolute Path Traversal

Reference
https://cwe.mitre.org/data/definitions/36 Absolute Path Traversal

Roles
- the <PathManipulation>
- the <TaintedInput>

Parent weaknesses
CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

Detection Patterns
ASCQM Sanitize User Input used in Path Manipulation

7.4.19 CWE-366 Race Condition within a Thread

Reference
https://cwe.mitre.org/data/definitions/366 Race Condition within a Thread

Roles
- the <Thread1>
- the <Thread2>
- the <ConflictingResource>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Creation of Lock On Private Non-Static Object to Access Private Static Data
ASCQM Data Read and Write without Proper Locking in Multi-Threaded Context

7.4.20 CWE-369 Divide by Zero

Reference
https://cwe.mitre.org/data/definitions/369 Divide By Zero

Roles
- the <Division>

Parent weaknesses
CWE-682 Incorrect Calculation

Detection Patterns

ASCQM Check and Handle ZERO Value before Use as Divisor

7.4.21 CWE-401 Improper Release of Memory Before Removing Last Reference ('Memory Leak')

Reference
https://cwe.mitre.org/data/definitions/401 Improper Release of Memory Before Removing Last Reference ('Memory Leak')

Roles
- the <MemoryAllocation>

Parent weaknesses
CWE-404 Improper Resource Shutdown or Release

Detection Patterns
ASCQM Ban Comma Operator from Delete Statement
ASCQM Implement Required Operations for Manual Resource Management
ASCQM Release Memory After Use
ASCQM Release Memory after Use with Correct Operation
ASCQM Release Platform Resource after Use
ASCQM Release in Destructor Memory Allocated in Constructor

7.4.22 CWE-404 Improper Resource Shutdown or Release

Reference
https://cwe.mitre.org/data/definitions/404 Improper Resource Shutdown or Release

Roles
- the <ResourceAllocation>

Contributing weaknesses
Weakness CWE-401 Improper Release of Memory Before Removing Last Reference ('Memory Leak')
Weakness CWE-772 Missing Release of Resource after Effective Lifetime
Weakness CWE-775 Missing Release of File Descriptor or Handle after Effective Lifetime

Detection Patterns
ASCQM Ban Comma Operator from Delete Statement
ASCQM Implement Virtual Destructor for Classes Derived from Class with Virtual Destructor
ASCQM Implement Virtual Destructor for Classes with Virtual Methods
ASCQM Implement Virtual Destructor for Parent Classes
ASCQM Release File Resource after Use in Operation
ASCQM Release Platform Resource after Use
ASCQM Release in Destructor Memory Allocated in Constructor

7.4.23 CWE-424 Improper Protection of Alternate Path

Reference
https://cwe.mitre.org/data/definitions/424 Improper Protection of Alternate Path

Roles
- the <AlternatePath>

Detection Patterns
ASCQM Ban Unintended Paths

7.4.24 CWE-434 Unrestricted Upload of File with Dangerous Type

Reference
https://cwe.mitre.org/data/definitions/434 Unrestricted Upload of File with Dangerous Type

Roles
- the <FileUpload>

Detection Patterns
ASCQM Sanitize User Input used in Path Manipulation

7.4.25 CWE-456 Missing Initialization of a Variable

Reference
https://cwe.mitre.org/data/definitions/456 Missing Initialization of a Variable

Roles
- the <VariableDeclaration>

Parent weaknesses
CWE-665 Improper Initialization

Detection Patterns
ASCQM Ban Allocation of Memory with Null Size
ASCQM Initialize Variables

7.4.26 CWE-457 Use of Uninitialized Variable

Reference
https://cwe.mitre.org/data/definitions/457 Use of Uninitialized Variable

Roles

- the <VariableDeclaration>
- the <VariableUse>

Parent weaknesses
CWE-665 Improper Initialization

Detection Patterns
ASCQM Ban Allocation of Memory with Null Size
ASCQM Initialize Variables

7.4.27 CWE-477 Use of Obsolete Function

Reference
https://cwe.mitre.org/data/definitions/477 Use of Obsolete Function

Roles
- the <ObsoleteFunctionCall>

Detection Patterns
ASCQM Ban Use of Deprecated Libraries

7.4.28 CWE-480 Use of Incorrect Operator

Reference
https://cwe.mitre.org/data/definitions/480 Use of Incorrect Operator

Roles
- the <Operator>

Detection Patterns
ASCQM Ban Assignment Operation Inside Logic Blocks
ASCQM Ban Comparison Expression Outside Logic Blocks
ASCQM Ban Incorrect Object Comparison
ASCQM Ban Incorrect String Comparison
ASCQM Ban Logical Operation with a Constant Operand

7.4.29 CWE-502 Deserialization of Untrusted Data

Reference
https://cwe.mitre.org/data/definitions/502 Deserialization of Untrusted Data

Roles
- the <Deserialization>
- the <TaintedData>

Detection Patterns
ASCQM Sanitize User Input used as Serialized Object

7.4.30 CWE-543 Use of Singleton Pattern Without Synchronization in a Multithreaded Context

Reference
https://cwe.mitre.org/data/definitions/543 Use of Singleton Pattern Without Synchronization in a Multithreaded Context

Roles
- the <SingletonUse>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Non-Final Static Data in Multi-Threaded Context
ASCQM Singleton Creation without Proper Locking in Multi-Threaded Context

7.4.31 CWE-564 SQL Injection: Hibernate

Reference
https://cwe.mitre.org/data/definitions/564 SQL Injection: Hibernate

Roles
- the <HibernateSQLStatement>
- the <TaintedInput>

Parent weaknesses
CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Detection Patterns
ASCQM Sanitize User Input used in SQL Access

7.4.32 CWE-567 Unsynchronized Access to Shared Data in a Multithreaded Context

Reference
https://cwe.mitre.org/data/definitions/567 Unsynchronized Access to Shared Data in a Multithreaded Context

Roles
- the <SharedDataAccess>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Non-Final Static Data in Multi-Threaded Context
ASCQM Data Read and Write without Proper Locking in Multi-Threaded Context

7.4.33 CWE-570 Expression is Always False

Reference
https://cwe.mitre.org/data/definitions/570 Expression is Always False

Roles
- the <BooleanExpression>

Detection Patterns
ASCQM Check Boolean Variables are Updated in Different Conditional Branches before Use

7.4.34 CWE-571 Expression is Always True

Reference
https://cwe.mitre.org/data/definitions/571 Expression is Always True

Roles
- the <BooleanExpression>

Detection Patterns
ASCQM Check Boolean Variables are Updated in Different Conditional Branches before Use

7.4.35 CWE-606 Unchecked Input for Loop Condition

Reference
https://cwe.mitre.org/data/definitions/606 Unchecked Input for Loop Condition

Roles
- the <LoopCondition>
- the <TaintedValue>

Detection Patterns
ASCQM Sanitize User Input used in Loop Condition

7.4.36 CWE-643 Improper Neutralization of Data within XPath Expressions ('XPath Injection')

Reference
https://cwe.mitre.org/data/definitions/643 Improper Neutralization of Data within XPath Expressions ('XPath Injection')

Roles
- the <XPathExpression>
- the <TaintedValue>

Detection Patterns
ASCQM Sanitize User Input used in Document Navigation Expression

7.4.37 CWE-652 Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')

Reference
https://cwe.mitre.org/data/definitions/652 Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')

Roles
- the <XQueryExpression>
- the <TaintedValue>

Detection Patterns
ASCQM Sanitize User Input used in Document Manipulation Expression

7.4.38 CWE-662 Improper Synchronization

Reference
https://cwe.mitre.org/data/definitions/662 Improper Synchronization

Roles
- the <Thread1>
- the <Thread2>
- the <SharedResourceAccess>

Contributing weaknesses
CWE-366 Race Condition within a Thread
CWE-543 Use of Singleton Pattern Without Synchronization in a Multithreaded Context
CWE-567 Unsynchronized Access to Shared Data in a Multithreaded Context
CWE-667 Improper Locking
CWE-764 Multiple Locks of a Critical Resource
CWE-820 Missing Synchronization
CWE-821 Incorrect Synchronization
CWE-833 Deadlock

RLB-11 Named Callable and Method Control Element in Multi-Thread Context with non-Final Static Storable or Member Element
RLB-12 Singleton Class Instance Creation without Proper Lock Element Management

Detection Patterns
ASCQM Ban Incorrect Synchronization Mechanisms
ASCQM Ban Non-Final Static Data in Multi-Threaded Context
ASCQM Ban Resource Access without Proper Locking in Multi-Threaded Context
ASCQM Data Read and Write without Proper Locking in Multi-Threaded Context
ASCQM Singleton Creation without Proper Locking in Multi-Threaded Context

7.4.39 CWE-665 Improper Initialization

Reference
https://cwe.mitre.org/data/definitions/665 Improper Initialization

Roles
- the <Initialization>

Contributing weaknesses
CWE-456 Missing Initialization of a Variable
CWE-457 Use of Uninitialized Variable

Detection Patterns
ASCQM Ban Self Assignment
ASCQM Initialize Pointers before Use
ASCQM Initialize Variables before Use

7.4.40 CWE-667 Improper Locking

Reference
https://cwe.mitre.org/data/definitions/667 Improper Locking

Roles
- the <Thread1>
- the <Thread2>
- the <SharedResourceAccess>
- the <Lock>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Creation of Lock On Inappropriate Object Type

ASCQM Ban Creation of Lock On Non-Final Object
ASCQM Ban Creation of Lock On Private Non-Static Object to Access Private Static Data
ASCQM Ban Resource Access without Proper Locking in Multi-Threaded Context
ASCQM Ban Sleep Between Lock Acquisition and Release
ASCQM Data Read and Write without Proper Locking in Multi-Threaded Context
ASCQM Release Lock After Use

7.4.41 CWE-672 Operation on a Resource after Expiration or Release

Reference
https://cwe.mitre.org/data/definitions/672 Operation on a Resource after Expiration or Release

Roles
- the <ResourceRelease>
- the <ResourceAccess>

Contributing weaknesses
CWE-415 Double Free
CWE-416 Use After Free

Detection Patterns
ASCQM Ban Double Release of Resource
ASCQM Ban Use of Expired Resource

7.4.42 CWE-681 Incorrect Conversion between Numeric Types

Reference
https://cwe.mitre.org/data/definitions/681 Incorrect Conversion between Numeric Types

Roles
- the <NumericConversion>

Contributing weaknesses
CWE-194 Unexpected Sign Extension
CWE-195 Signed to Unsigned Conversion Error
CWE-196 Unsigned to Signed Conversion Error
CWE-197 Numeric Truncation Error

Detection Patterns
ASCQM Ban Incorrect Numeric Implicit Conversion

7.4.43 CWE-682 Incorrect Calculation

Reference

https://cwe.mitre.org/data/definitions/682 Incorrect Calculation

Roles
- the <Calculation>

Contributing weaknesses
CWE-131 Incorrect Calculation of Buffer Size
CWE-369 Divide By Zero

Detection Patterns

7.4.44 CWE-732 Incorrect Permission Assignment for Critical Resource

Reference
https://cwe.mitre.org/data/definitions/732 Incorrect Permission Assignment for Critical Resource

Roles
- the <PermissionAssignment>

Detection Patterns
ASCQM Ban File Creation with Default Permissions

7.4.45 CWE-77 Improper Neutralization of Special Elements used in a Command ('Command Injection')

Reference
https://cwe.mitre.org/data/definitions/77 Improper Neutralization of Special Elements used in a Command ('Command Injection')

Roles
- the <Command>
- the <TaintedValue>

Contributing weaknesses
CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
CWE-88 Argument Injection or Modification

Detection Patterns

7.4.46 CWE-772 Missing Release of Resource after Effective Lifetime

Reference
https://cwe.mitre.org/data/definitions/772 Missing Release of Resource after Effective Lifetime

Roles
- the <ResourceAllocation>

Parent weaknesses
CWE-404 Improper Resource Shutdown or Release

Detection Patterns
ASCQM Release File Resource after Use in Operation
ASCQM Release Platform Resource after Use
ASCQM Release in Destructor Memory Allocated in Constructor

7.4.47 CWE-775 Missing Release of File Descriptor or Handle after Effective Lifetime

Reference
https://cwe.mitre.org/data/definitions/775 Missing Release of File Descriptor or Handle after Effective Lifetime

Roles
- the <FileDescriptorOrHandleAllocation>

Parent weaknesses
CWE-775 Missing Release of File Descriptor or Handle after Effective Lifetime

Detection Patterns
ASCQM Release File Resource after Use in Class
ASCQM Release File Resource after Use in Operation

7.4.48 CWE-778 Insufficient Logging

Reference
https://cwe.mitre.org/data/definitions/778 Insufficient Logging

Roles
- the <SecurityExceptionOrError>

Detection Patterns
ASCQM Log Caught Security Exceptions

7.4.49 CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

Reference
https://cwe.mitre.org/data/definitions/78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

Roles
- the <OSCommand>
- the <TaintedValue>

Parent weaknesses
CWE-77 Improper Neutralization of Special Elements used in a Command ('Command Injection')

Detection Patterns
ASCQM Sanitize User Input used in System Command

7.4.50 CWE-783 Operator Precedence Logic Error

Reference
https://cwe.mitre.org/data/definitions/783 Operator Precedence Logic Error

Roles
- the <Formula>

Detection Patterns
ASCQM Ban Incorrect Joint Comparison
ASCQM Ban Not Operator On Non-Boolean Operand Of Comparison Operation
ASCQM Ban Not Operator On Operand Of Bitwise Operation

7.4.51 CWE-786 Access of Memory Location Before Start of Buffer

Reference
https://cwe.mitre.org/data/definitions/786 Access of Memory Location Before Start of Buffer

Roles
- the <MemoryAccess>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Check Index of Array Access
ASCQM Check Input of String Manipulation Primitives with Boundary Checking Capabilities

7.4.52 CWE-787 Out-of-bounds Write

Reference
https://cwe.mitre.org/data/definitions/787 Out-of-bounds Write

Roles
- the <BufferWrite>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Check Index of Array Access
ASCQM Check Input of Memory Manipulation Primitives

7.4.53 CWE-788 Access of Memory Location After End of Buffer

Reference
https://cwe.mitre.org/data/definitions/788 Access of Memory Location After End of Buffer

Roles
- the <MemoryAccess>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities
ASCQM Check Index of Array Access
ASCQM Check Input of Memory Manipulation Primitives

7.4.54 CWE-789 Uncontrolled Memory Allocation

Reference
https://cwe.mitre.org/data/definitions/789 Uncontrolled Memory Allocation

Roles
- the <MemoryAllocation>

Detection Patterns
ASCQM Check Input of Memory Allocation Primitives
ASCQM Sanitize User Input used as Array Index

7.4.55 CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Reference
https://cwe.mitre.org/data/definitions/79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Roles
- the <WebPageGenerationStatement>
- the <TaintedInput>

Detection Patterns
ASCQM Sanitize Stored Input used in User Output
ASCQM Sanitize User Input used in User Output

7.4.56 CWE-798 Use of Hard-coded Credentials

Reference
https://cwe.mitre.org/data/definitions/798 Use of Hard-coded Credentials

Roles
- the <HardCodedValue>
- the <Authentication>

Contributing weaknesses
CWE-259 Use of Hard-coded Password
CWE-321 Use of Hard-coded Cryptographic Key

Detection Patterns
ASCQM Ban Hard-Coded Literals used to Connect to Resource

7.4.57 CWE-805 Buffer Access with Incorrect Length Value

Reference
https://cwe.mitre.org/data/definitions/805 Buffer Access with Incorrect Length Value

Roles
- the <BufferAccess>
- the <LengthParameter>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities
ASCQM Check Input of Memory Manipulation Primitives
ASCQM Check Input of String Manipulation Primitives with Boundary Checking Capabilities

7.4.58 CWE-820 Missing Synchronization

Reference
https://cwe.mitre.org/data/definitions/820 Missing Synchronization

Roles
- the <SharedResourceUse>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Resource Access without Proper Locking in Multi-Threaded Context

7.4.59 CWE-821 Incorrect Synchronization

Reference
https://cwe.mitre.org/data/definitions/821 Incorrect Synchronization

Roles
- the <SharedResourceUse>
- the <IncorrectSynchronization>

Parent weaknesses
CWE-662 Improper Synchronization

Detection Patterns
ASCQM Ban Incorrect Synchronization Mechanisms

7.4.60 CWE-822 Untrusted Pointer Dereference

Reference
https://cwe.mitre.org/data/definitions/822 Untrusted Pointer Dereference

Roles
- the <PointerDereferencing>
- the <TaintedInput>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Sanitize User Input used as Pointer

7.4.61 CWE-823 Use of Out-of-range Pointer Offset

Reference
https://cwe.mitre.org/data/definitions/823 Use of Out-of-range Pointer Offset

Roles
- the <PointerOffset>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Check Offset used in Pointer Arithmetic

7.4.62 CWE-824 Access of Uninitialized Pointer

Reference
https://cwe.mitre.org/data/definitions/824 Access of Uninitialized Pointer

Roles
- the <PointerAccess>

Parent weaknesses
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

Detection Patterns
ASCQM Initialize Pointers before Use

7.4.63 CWE-825 Expired Pointer Dereference

Reference
https://cwe.mitre.org/data/definitions/825 Expired Pointer Dereference

Roles
- the <PointerAccess>
- the <PointerRelease>

Parent weaknesses
CWE-672 Operation on a Resource after Expiration or Release

Detection Patterns
ASCQM Ban Use of Expired Pointer

7.4.64 CWE-835 Loop with Unreachable Exit Condition ('Infinite Loop')

Reference

https://cwe.mitre.org/data/definitions/835 Loop with Unreachable Exit Condition ('Infinite Loop')

Roles
- the <InfiniteLoop>

Detection Patterns
ASCQM Ban Unmodified Loop Variable Within Loop
ASCQM Ban While TRUE Loop Without Path To Break

7.4.65 CWE-88 Argument Injection or Modification

Reference
https://cwe.mitre.org/data/definitions/88 Argument Injection or Modification

Roles
- the <Command>
- the <TaintedInput>

Parent weaknesses
CWE-77 Improper Neutralization of Special Elements used in a Command ('Command Injection')

Detection Patterns
ASCQM Sanitize User Input used in System Command

7.4.66 CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Reference
https://cwe.mitre.org/data/definitions/89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Roles
- the <SQLStatement>
- the <TaintedInput>

Contributing weaknesses
Weakness CWE-564 SQL Injection: Hibernate

Detection Patterns
ASCQM Sanitize User Input used in Document Manipulation Expression
ASCQM Sanitize User Input used in Document Navigation Expression

7.4.67 CWE-90 Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')

Reference
https://cwe.mitre.org/data/definitions/90 Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')

Roles
- the <LDAPQuery>
- the <TaintedInput>

Detection Patterns
ASCQM Sanitize User Input used to access Directory Resources

7.4.68 CWE-91 XML Injection (aka Blind XPath Injection)

Reference
https://cwe.mitre.org/data/definitions/91 XML Injection (aka Blind XPath Injection)

Roles
- the <XMLHandlingExpression>
- the <TaintedValue>

Detection Patterns
ASCQM Sanitize User Input used in Document Manipulation Expression
ASCQM Sanitize User Input used in Document Navigation Expression

7.4.69 CWE-99 Improper Control of Resource Identifiers ('Resource Injection')

Reference
https://cwe.mitre.org/data/definitions/99 Improper Control of Resource Identifiers ('Resource Injection')

Roles
- the <ResourceIdentifier>
- the <TaintedValue>

Detection Patterns
ASCQM Sanitize User Input used in Path Manipulation

7.4.70 CWE-611 Improper Restriction of XML External Entity Reference ('XXE')

Reference

https://cwe.mitre.org/data/definitions/CWE-611 Improper Restriction of XML External Entity Reference ('XXE')

Roles
- the <XMLHandlingOperation>

Detection Patterns
ASCQM Secure Use of Unsafe XML Processing with Secure Parser
ASCQM Secure XML Parsing with Secure Options

7.4.71 CWE-1057 Data Access Control Element from Outside Designated Data Manager Component

Usage name
Circumventing data access routines

Reference
https://www.omg.org/spec/ASCPEM ASCPEM-CWE-1057 Data Access Control Element from Outside Designated Data Manager Component

Roles
- the <DataManager>
- the <DataAccess>

Detection Patterns
ASCQM Ban Unintended Paths

7.4.72 CWE-415 Double Free

Reference
https://cwe.mitre.org/data/definitions/415 Double Free

Roles
- the <ResourceRelease >
- the <ResourceAccess>
- the <ResourceUse>

Parent weaknesses
CWE-672 Operation on a Resource after Expiration or Release

Detection Patterns
ASCQM Ban Double Free On Pointers

7.4.73 CWE-416 Use After Free

Reference
https://cwe.mitre.org/data/definitions/416 Use After Free

Roles
- the <ResourceRelease>
- the <ResourceUse>

Parent weaknesses
CWE-672 Operation on a Resource after Expiration or Release

Detection Patterns
ASCQM Ban Free Operation on Pointer Received as Parameter
ASCQM Ban Use of Expired Pointer
ASCQM Implement Copy Constructor for Class With Pointer Resource

7.4.74 Security detection patterns

Detection Patterns
ASCQM Ban Allocation of Memory with Null Size
ASCQM Ban Assignment Operation Inside Logic Blocks
ASCQM Ban Buffer Size Computation Based on Array Element Pointer Size
ASCQM Ban Buffer Size Computation Based on Bitwise Logical Operation
ASCQM Ban Buffer Size Computation Based on Incorrect String Length Value
ASCQM Ban Comma Operator from Delete Statement
ASCQM Ban Comparison Expression Outside Logic Blocks
ASCQM Ban Creation of Lock On Inappropriate Object Type
ASCQM Ban Creation of Lock On Non-Final Object
ASCQM Ban Creation of Lock On Private Non-Static Object to Access Private Static Data
ASCQM Ban Double Free On Pointers
ASCQM Ban Double Release of Resource
ASCQM Ban File Creation with Default Permissions
ASCQM Ban Free Operation on Pointer Received as Parameter
ASCQM Ban Hard-Coded Literals used to Connect to Resource
ASCQM Ban Incompatible Lock Acquisition Sequences
ASCQM Ban Incorrect Joint Comparison
ASCQM Ban Incorrect Numeric Implicit Conversion
ASCQM Ban Incorrect Object Comparison
ASCQM Ban Incorrect String Comparison
ASCQM Ban Incorrect Synchronization Mechanisms
ASCQM Ban Input Acquisition Primitives without Boundary Checking Capabilities
ASCQM Ban Logical Operation with a Constant Operand
ASCQM Ban Non-Final Static Data in Multi-Threaded Context

ASCQM Ban Not Operator On Non-Boolean Operand Of Comparison Operation
ASCQM Ban Not Operator On Operand Of Bitwise Operation
ASCQM Ban Resource Access without Proper Locking in Multi-Threaded Context
ASCQM Ban Self Assignment
ASCQM Ban Sequential Acquisitions of Single Non-Reentrant Lock
ASCQM Ban Sleep Between Lock Acquisition and Release
ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities
ASCQM Ban Unintended Paths
ASCQM Ban Unmodified Loop Variable Within Loop
ASCQM Ban Use of Deprecated Libraries
ASCQM Ban Use of Expired Pointer
ASCQM Ban Use of Expired Resource
ASCQM Ban Use of Thread Control Primitives with Known Deadlock Issues
ASCQM Ban While TRUE Loop Without Path To Break
ASCQM Check Boolean Variables are Updated in Different Conditional Branches before Use
ASCQM Check Index of Array Access
ASCQM Check Input of Memory Allocation Primitives
ASCQM Check Input of Memory Manipulation Primitives
ASCQM Check Input of String Manipulation Primitives with Boundary Checking Capabilities
ASCQM Check Offset used in Pointer Arithmetic
ASCQM Check Return Value of Resource Operations Immediately
ASCQM Check and Handle ZERO Value before Use as Divisor
ASCQM Data Read and Write without Proper Locking in Multi-Threaded Context
ASCQM Handle Return Value of Must Check Operations
ASCQM Implement Copy Constructor for Class With Pointer Resource
ASCQM Implement Required Operations for Manual Resource Management
ASCQM Implement Virtual Destructor for Classes Derived from Class with Virtual Destructor
ASCQM Implement Virtual Destructor for Classes with Virtual Methods
ASCQM Implement Virtual Destructor for Parent Classes
ASCQM Initialize Pointers before Use
ASCQM Initialize Variables
ASCQM Initialize Variables before Use
ASCQM Log Caught Security Exceptions
ASCQM Release File Resource after Use in Class
ASCQM Release File Resource after Use in Operation
ASCQM Release Lock After Use
ASCQM Release Memory After Use
ASCQM Release Memory after Use with Correct Operation
ASCQM Release Platform Resource after Use
ASCQM Release in Destructor Memory Allocated in Constructor
ASCQM Sanitize Stored Input used in User Output
ASCQM Sanitize User Input used as Array Index
ASCQM Sanitize User Input used as Pointer
ASCQM Sanitize User Input used as Serialized Object

ASCQM Sanitize User Input used as String Format
ASCQM Sanitize User Input used in Document Manipulation Expression
ASCQM Sanitize User Input used in Document Navigation Expression
ASCQM Sanitize User Input used in Expression Language Statement
ASCQM Sanitize User Input used in Loop Condition
ASCQM Sanitize User Input used in Path Manipulation
ASCQM Sanitize User Input used in SQL Access
ASCQM Sanitize User Input used in System Command
ASCQM Sanitize User Input used in User Output
ASCQM Sanitize User Input used to access Directory Resources
ASCQM Secure Use of Unsafe XML Processing with Secure Parser
ASCQM Secure XML Parsing with Secure Options
ASCQM Singleton Creation without Proper Locking in Multi-Threaded Context

8. ASCQM Weakness Detection Patterns

8.1 ASCQM Check Index of Array Access

Descriptor

ASCQM Check Index of Array Access(PathFromDeclarationStatementToUseAsAnIndexStatement, VariableDeclarationStatement, ArrayAccessStatement)

Description

Identify occurrences in application model where
- the <PathFromDeclarationStatementToUseAsAnIndexStatement> path
- from the <VariableDeclarationStatement> variable declaration statement
- to the <ArrayAccessStatement> array access statement using the variable as an index,
- lacks a range check operation.

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
...
StorableUnit id="su1"
StorableUnit id="su2"
ArrayType id="at1"
StorableUnit id="su3" type="at1"
...
ActionElement id="ae2"
    Flow "ae3"
    Reads "su1"
    Writes "su2"
ActionElement id="ae3"
    Flow "ae4"
ActionElement id="ae4"
    Flow "ae5"
ActionElement id="ae5" kind="ArraySelect|ArrayReplace"
    Addresses "su3"
    Reads "su2"
    Reads|Writes ...
 ...
```

KDM elements absent from the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
ActionElement id="ae2" kind="GreaterThan|GreaterThanOrEqual"
    Reads "su2"
    Reads ...
    ...
ActionElement id="ae3" kind="LessThan|LessThanOrEqual"
    Reads "su2"
    Reads ...
    ...
```

What to report

Roles to report are
- the <PathFromDeclarationStatementToUseAsAnIndexStatement> path
- the <VariableDeclarationStatement> variable declaration statement
- the <ArrayAccessStatement> array access statement

8.2 ASCQM Check Input of Memory Manipulation Primitives

Descriptor

ASCQM Check Input of Memory Manipulation Primitives(MemoryManipulationCall)

Description

Identify occurrences in application model where
- the <MemoryManipulationCall> call to a memory manipulation function, procedure, method, ... with boundary checking capabilities
- uses the length parameter without range checking its value

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
PointerType id="pt1"
IntegerType id="it1"
ControlElement id="ce1" name="memcpy|..." type="ce1_signature"
    Signature id="ce1_signature"
        ...
        ParameterUnit id="pu1" type="dt1" kind="byValue"
        ParameterUnit id="pu2" type="pt1" kind="return"
    ...
...
StorableUnit id="su1"  type="it1"
...
ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall"
```

```
    ...
    Reads "su1"
    Calls "ce1"
```

KDM elements absent from the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
ActionElement id="ae2" kind="GreaterThan|GreaterThanOrEqual"
    Reads "su1"
    ...
ActionElement id="ae3" kind="LessThan|LessThanOrEqual"
    Reads "su1"
    ...
```

What to report

Roles to report
- the <MemoryManipulationCall> call to a memory manipulation function, procedure, method, ... with boundary checking capabilities

8.3 ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities

Descriptor

ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities(StringManipulationCall)

Description

Identify occurrences in application model where
- the <StringManipulationCall> call to a string manipulation function, procedure, method, ... without boundary checking capabilities

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
ControlElement id="ce1" name="strcpy|strlen|..."
    ...
...
ActionElement id="ae3" kind="Call|PtrCall|MethodCall|VirtualCall"
    ...
    Calls "ce1"
```

What to report

Roles to report:
- the <StringManipulationCall> call to a string manipulation function, procedure, method, ... without boundary checking capabilities

8.4 ASCQM Check Input of String Manipulation Primitives with Boundary Checking Capabilities

Descriptor

ASCQM Check Input of String Manipulation Primitives with Boundary Checking Capabilities(StringManipulationCall)

Description

Identify occurrences in application model where
- the <StringManipulationCall> call to a string manipulation function, procedure, method, ... with boundary checking capabilities
- uses the length parameter without range checking its value

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
StringType id="st1"
IntegerType id="it1"
ControlElement id="ce1" name="strncpy|strncat|..." type="ce1_signature"
    Signature id="ce1_signature"
        ParameterUnit id="pu1" type="st1"
        ParameterUnit id="pu2" type="it1" kind="byValue"
        ParamteterUnit id="pu3" type="st1" kind="return"
    ...
...
StorableUnit id="su1" type="it1"
...
ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall"
    ...
    Reads "su1"
    Calls "ce1"
```

KDM elements absent from the application model

KDM outline illustrating only the essential elements related to micro KDM:

```
ActionElement id="ae2" kind="GreaterThan|GreaterThanOrEqual"
    Reads "su1"
    ...
ActionElement id="ae3" kind="LessThan|LessThanOrEqual"
    Reads "su1"
    ...
```

What to report

Roles to report
- the <StringManipulationCall> call to a string manipulation function, procedure, method, ... with boundary checking capabilities

8.5 ASCQM Ban Use of Expired Pointer

Descriptor

ASCQM Ban Use of Expired Pointer(PathToPointerAccessFromPointerRelease, PointerReleaseStatement, PointerAccessStatement)

Description

Identify occurrences in application model where
- the <PathToPointerAccessFromPointerRelease> path
- from the <PointerReleaseStatement> resource release statement
- to the <PointerAccessStatement> resource access statement

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
ClassUnit|IntegerType|DecimalType|FloatType|StringType|VoidType|... id="dt1"
PointerType id="pt1"
    ItemUnit id="pi1" type="dt1"
StorableUnit id="su1"  type="pt1"
...
ActionElement id="ae1" name="free|delete|..."
    Addresses "pt1"
    Flows "ae2"
ActionElement id="ae2"
    Flows "ae3"
ActionElement id="ae3"
kind=PtrSelect|PtrReplace|Call|PtrCall|MethodCall|VirtualCall"
    Reads|Addresses "pt1"
...
```

or

```
ClassUnit|IntegerType|DecimalType|FloatType|StringType|VoidType|... id="dt1"
name="dt1"
PointerType id="pt1" name="pt1"
    ItemUnit id="iu1" type="dt1" ext="dt1 & pt1"
StorableUnit id="su1"  type="dt1"
StorableUnit id="su2"  type="pt1"
    HasType "pt1"
    HasValue "su1"
...
ActionElement id="ae1" name="free|delete|...|push_back|..."
    Addresses "su1"
    Flows "ae2"
ActionElement id="ae2"
    Flows "ae3"
ActionElement id="ae3"
kind=PtrSelect|PtrReplace|Call|PtrCall|MethodCall|VirtualCall"
    Reads|Addresses "su2"
```

What to report

Roles to report
- the <PathToPointerAccessFromPointerRelease> path
- the <PointerReleaseStatement> resource release statement
- the <PointerAccessStatement> resource access statement

8.6 ASCQM Ban Input Acquisition Primitives without Boundary Checking Capabilities

Descriptor

ASCQM Ban Input Acquisition Primitives without Boundary Checking Capabilities(InputAcquisitionCall)

Description

Identify occurrences in application model where
- the <InputAcquisitionCall> call to an input acquisition function, procedure, method, ... without boundary checking capabilities

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
ControlElement id="ce1" name="gets|scanf|..."
```

```
    ...
...
ActionElement id="ae3" kind="Call|PtrCall|MethodCall|VirtualCall"
    ...
    Calls "ce1"
```

What to report

Roles to report:
- the <InputAcquisitionCall> call to an input acquisition function, procedure, method, ... without boundary checking capabilities

8.7 ASCQM Check Offset used in Pointer Arithmetic

Descriptor

ASCQM Check Offset used in Pointer Arithmetic(ArithmeticExpression, EvaluationStatement)

Description

Identify occurrences in application model where
- the result of the <ArithmeticExpression> arithmetic expression,
- with an offset value which is not range checked
- is used to dererence the pointer in the <EvaluationStatement> evaluation statement

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
...
PointerType id="pt1"
StorableUnit id="su1" type="pt1"
...
IntegerType id="it1"
StorableUnit id="su2" type="it1"
StorableUnit id="su3" type="it1"
...
ActionElement id="ae1" kind="Add|Substract"
    Reads "su1"
    Reads "su2"
    Writes "su3"
...
ActionElement id="ae2" kind="PtrSelect|PtrReplace"
    Addresses "su3"
    ..
```

KDM elements absent from the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
ActionElement id="ae2" kind="GreaterThan|GreaterThanOrEqual"
    Reads "su2"
    Reads ...
    ...
ActionElement id="ae3" kind="LessThan|LessThanOrEqual"
    Reads "su2"
    Reads ...
    ...
```

What to report

Roles to report are
- the <ArithmeticExpression> arithmetic expression
- the <EvaluationStatement> evaluation statement

8.8 ASCQM Sanitize User Input used as Pointer

Descriptor

ASCQM Sanitize User Input used as Pointer(PathFromUserInputToPointerDereferencing, UserInput, PointerDereferencingStatement, PointerDereferencingSanitizationControlElementList)

Description

Identify occurrences in application model where
- the <PathFromUserInputToPointerDereferencing> path
- from the <UserInput> user interface input
- to the <PointerDereferencingStatement> pointer dereferencing statement,
- lacks a sanitization operation from the <PointerDereferencingSanitizationControlElementList> list of vetted sanitization.

The list of vetted sanitization primitives is an input to provide to the measurement process.

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
UIModel
    UIField id="uf1"
```

```
    UIAction id="ua1" implementation="ae1" kind="input"
        ReadsUI "uf1"
...
CodeModel
    ...
    StorableUnit id="su1"
    StorableUnit id="su2"
    ActionElement id="ae1" kind="UI"
        Writes "su1"
        Flow "ae2"
    ActionElement id="ae2"
        Flow "ae3"
        Reads "su1"
        Writes "su2"
    ActionElement id="ae3"
        Flow "ae4"
    ActionElement id="ae4"
        Flow "ae5"
    ActionElement id="ae5" kind="PtrSelect"
        Addresses "su2"
        Reads|Writes ...
 ...
```

KDM elements absent from the application model

KDM outline illustrating only the essential elements related to micro KDM:

```
ControlElement id="ce1" kind="sanitization"
...
ActionElement id="ae3" kind="Call|PtrCall|MethodCall|VirtualCall"
        Flow "ae4"
        Calls "ce1"
        Reads "su2"
        Writes "su2"
...
```

1.1.1.What to report

Roles to report are
- the <PathFromUserInputToPointerDereferencing> path
- the <UserInput> user interface input
- the <PointerDereferencingStatement> pointer dereferencing statement,
- the <PointerDereferencingSanitizationControlElementList> list of vetted sanitization.

8.9 ASCQM Initialize Pointers before Use

Descriptor

ASCQM Initialize Pointers before Use(PathToPointerAccessFromPointerDeclaration, PointerDeclarationStatement, PointerAccessStatement)

Description

Identify occurrences in application model where
- the <PathToPointerAccessFromPointerDeclaration> path
- from the <PointerDeclarationStatement> pointer declaration statement
- to the <PointerAccessStatement> pointer access statement
- lacks a pointer initialization statement

excluding variable and platform resources

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
...
PointerType id="pt1"
StorableUnit id="su1" type="pt1"
...
ActionElement id="ae2" ...
     Flows "ae3"
ActionElement id="ae3" kind="PtrSelect"
     Reads "su1"
     ...
...
```

KDM elements absent from the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
...
ActionElement id="ae1" kind="Assign|Ptr"
    Writes "su1"
    Flows "ae2"
...
```

What to report

Roles to report are
- the <PathToPointerAccessFromPointerDeclaration> path
- the <PointerDeclarationStatement> pointer declaration statement
- the <PointerAccessStatement> pointer access statement

8.10 ASCQM Check NULL Pointer Value before Use

Descriptor

ASCQM Check NULL Pointer Value before Use(EvaluationStatement)

Description

Identify occurrences in application model where
- a pointer is evaluated in the <EvaluationStatement> evaluation statement
- with no NULL comparison operation performed on the pointer immediately before

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
...
PointerType id="pt1"
    ItemUnit id="iu1"
StorableUnit id="su1" type="pt1"
ActionElement id="ae3" kind="PtrSelect|PtrReplace"
    Reads "iu1"
    Addresses "su1"
```

KDM elements absent from the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
...
Value id="v1" name="NULL|nullptr"
StorableUnit id="su2"
ActionElement id="ae1" kind="NotEqual"
    Reads "v1"
    Reads "su1"
    Writes "su2"
    Flows "ae2"
ActionElement id="ae2" kind="Condition"
    Reads "su2"
    TrueFlow "ae3"
    FalseFlow "ff1"
...
```

What to report

Roles to report are
- the <EvaluationStatement> evaluation statement

8.11 ASCQM Ban Use of Expired Resource

Descriptor

ASCQM Ban Use of Expired Resource(PathToResourceAccessFromResourceRelease, ResourceReleaseStatement, ResourceAccessStatement)

Description

Identify occurrences in application model where
- the <PathToResourceAccessFromResourceRelease> path
- from the <ResourceReleaseStatement> resource release statement
- to the <ResourceAccessStatement> resource access statement
excluding pointers

KDM outline illustratio

KDM outline illustrating only the essential elements related to micro KDM:

```
PlatformModel
    ...
    DataManager|FileResource id="pr1"
    ...
    PlatformResource id="pa1" kind="open" implementation="ae4"
        ManagesResource "pr1"
    PlatformResource id="pa2" kind="close" implementation="ae1"
        ManagesResource "pr1"

...
CodeModel
    ...
    ActionElement id="ae1" kind="PlatformAction"
        Flows "ae3"
    ActionElement id="ae3"
         Flows "ae4"
    ActionElement id="ae4" kind="PlatformAction"
     ...
...
```

What to report

Roles to report
- the <PathToResourceAccessFromResourceRelease> path
- the <ResourceReleaseStatement> resource release statement
- the <ResourceAccessStatement> resource access statement

8.12 ASCQM Ban Double Release of Resource

Descriptor

ASCQM Ban Double Release of Resource(PathToResourceReleaseFromResourceRelease, FirstResourceReleaseStatement, SecondResourceReleaseStatement)

Description

Identify occurrences in application model where
- the <PathToResourceReleaseFromResourceRelease> path
- from the <FirstResourceReleaseStatement> resource release statement
- to the <SecondResourceReleaseStatement> resource release statement

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
PlatformModel
    ...
    DataManager|ExecutionResource id="pr1"
    ...
    PlatformAction id="pa2" kind="close" implementation="ae1 ae4"
        ManagesResource "pr1"

...
CodeModel
    ...
    ActionElement id="ae1" kind="PlatformAction"
        Flows "ae3"
    ActionElement id="ae3"
         Flows "ae4"
    ActionElement id="ae4" kind="PlatformAction"
        ...
...
```

What to report

Roles to report
- the <PathToResourceReleaseFromResourceRelease> path
- the <FirstResourceReleaseStatement> resource release statement
- the <SecondResourceReleaseStatement> resource release statement

8.13 ASCQM Implement Copy Constructor for Class With Pointer Resource

Descriptor

ASCQM Implement Copy Constructor for Class With Pointer Resource(Class, Pointer)

Description

Identify occurrences in application model where
- the <Class> Class
- owns the <Pointer> pointer resource
- but lacks a copy constructor

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
PointerType id="pointerType"
    ...
ClassUnit id="cu1"
    MemberUnit id="mu1" type="pointerType"
    ...
```

KDM elements absent from the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
ClassUnit id="cu1"
    ...
    MethodUnit is="m1"
name="class|this|__construct|new|New|__new__|alloc|constructor|initialize|...
" methodKind="constructor" type="m1_signature"
        Signature id = "m1_signature"
            ParameterUnit id="p1" name="p1" type="class" kind="byReference"
            ParameterUnit id="r" name="r" type="class" kind="return"
        ...
```

What to report

Roles to report are
- the <Class> Class
- the <Pointer> pointer resource

8.14 ASCQM Ban Free Operation on Pointer Received as Parameter

Descriptor

ASCQM Ban Free Operation on Pointer Received as Parameter(ReleaseStatement, Signature)

Description

Identify occurrences in application model where
- the pointer is released by the <ReleaseStatement> release statement
- and was received as a parameter in the <Signature> signature

The list of release operations are technology, language dependent. E.g. with C-type languages: free, delete.

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
...
PointerType id="pt1"
...
ControlElement id="ce1" name="free|delete|..."
...
CallableUnit kind="regular|external|stored" | MethodUnit id="ce2"
type="ce2_signature"
    Signature id="ce2_signature"
        ParameterUnit id="pu1" kind="byReference" type="pt1"
    ...
    ActionElement id="ae1" kind="Call|PtrCall[MethodCall|VirtualCall"
        Calls "ce1"
        Reads "pu1"
    ...
```

What to report

Roles to report are
- the <ReleaseStatement> release statement
- the <Signature> signature

8.15 ASCQM Ban Delete of VOID Pointer

Descriptor

ASCQM Ban Delete of VOID Pointer(DeclarationStatement, ReleaseStatement)

Description

Identify occurrences in application model where
- the pointer declared as a VOID pointer in <DeclarationStatement> declaration statement
- is released by the <ReleaseStatement> release statement
- without ever been casted into a non-VOID pointer

The list of release operations are technology, language dependent. E.g. with C-type languages: delete.

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
VoidType id="vt1"
PointerType id="pt1"
    ItemUnit id="iu1" type="vt1"
StorableUnit id="su1" type="pt1"
ControlElement id="ce1" name="delete|..."
    ...
ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall"
    Reads "su1"
    Calls "ce1"
```

KDM elements absent from the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
...
IntegerType|DecimalType|FloatType|StringType|ClassUnit id="dt1"
PointerType id="pt2"
    ItemUnit id="iu2" type="dt1"
ActionElement id="ae2" kind="TypeCast|DynCast"
    Reads "su1"
    UsesType "pt2"
    Writes "su1"
...
```

What to report

Roles to report are
- the <DeclarationStatement> declaration statement
- the <ReleaseStatement> release statement

8.16 ASCQM Ban Variable Increment or Decrement Operation in Operations using the Same Variable

Descriptor

ASCQM Ban Variable Increment or Decrement Operation in Operations using the Same Variable(VariableAssignment)

Description

Identify occurrences in application model where
- the <VariableAssignment> variable assignment
- uses the outcome of increment or decrement operation on a variable
- jointly with the variable itself

e.g.: x + x++;

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
StorableUnit id="su1"
ActionElement id="ae1" kind="Compound"
    ActionElement id="ae2" kind="Incr|Decr"
        Addresses "su1"
    ...
    ActionElement id="ae3"
        ...
        Reads "su1"
...
```

What to report

Roles to report
- the <VariableAssignment> variable assignment

8.17 ASCQM Ban Reading and Writing the Same Variable Used as Assignment Value

Descriptor

ASCQM Ban Reading and Writing the Same Variable Used as Assignment Value(VariableAssignment)

Description

Identify occurrences in application model where
- the <VariableAssignment> variable assignment
- uses the outcome of an operation on a variable
- jointly with the assignment of the variable itself

e.g.: x = a + (a=2);

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
StorableUnit id="su1"
ActionElement id="ae1" kind="Compound"
    StorableUnit id="su2"
    ActionElement id="ae2" kind="Assign"
        ...
        Writes "su1"
    ...
    ActionElement id="ae3"
        ...
        Reads "su1"
        Writes "su2"
    ActionElement id="ae4" kind="Assign"
        Reads "su2"
        Writes ...
```

What to report

Roles to report
- the <VariableAssignment> variable assignment

8.18 ASCQM Handle Return Value of Resource Operations

Descriptor

ASCQM Handle Return Value of Resource Operations(CallToTheOperation)

Description

Identify occurrences in application model where
- the platform resource management function, method, procedure, ... is called in the <CallToTheOperation> call statement
- with no use in a conditional statement of the return value

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
PlatformModel
    ...
    DataManager|ExecutionResource|... id="pr1"
    ...
    PlatformResource id="pa1"  implementation="ae1"
```

```
        ManagesResource|ReadsResource|WritesResource "pr1"
    ...
CodeModel
...
    CallableUnit|MethodUnit id="ce1" type="ce1_signature"
        Signature id="ce1_signature"
            ParameterUnit id="pu1" kind="return"
    ...
    ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall"
...
```

KDM elements absent from the application model

KDM outline illustrating only the essential elements related to micro KDM:

```
StorableUnit id="su1"
...
ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall"
    Writes "su1"
    Flows "ae2"
ActionElement id="ae2" kind="Switch"
    Reads "su1"
    GuardedFlow "gf1"
    GuardedFlow|FalseFlow "gf2"
...
```

or

```
StorableUnit id="su1"
StorableUnit id="su2"
...
ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall"
    Writes "su1"
    Flows "ae2"
ActionElement id="ae2"
kind="Equal|NotEqual|LessThan|LessThanOrEqual|GreaterThan|GreatedThanOrEqual"
    Reads "su1"
    Writes "su2"
    Flows "ae3"
ActionElement id="ae3" kind="Condition"
    TrueFlow "tf1"
    FalseFlow "ff1"
...
```

What to report

Roles to report are
- the <CallToTheOperation> call statement

8.19 ASCQM Ban Incorrect Numeric Conversion of Return Value

Descriptor

ASCQM Ban Incorrect Numeric Conversion of Return Value(FunctionMethodOrProcedure, VariableDataType, CallStatement, TargetDataType)

Description

Identify occurrences in application model where
- the <FunctionMethodOrProcedure> function, method, procedure, ...
- declared to return a value with the <VariableDataType> numerical data type
- is called in the <CallStatement> call statement
- with assignment of its return value to a variable of the <TargetDataType> second numerical data type
- which is incompatible with the first one
- without any explicit casting

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
IntegerType|DecimalType|FloatType id="dt1"
IntegerType|DecimalType|FloatType id="dt2"
StorableUnit|ItemUnit|MemberUnit|Value id="de1" type="dt2"
...
CallableUnit|MethodUnit id="ce1" type="ce1_signature"
attribute="CheckReturnValue|..."
    Signature id="ce1_signature"
        ParameterUnit id="pu1" kind="return" type="dt1"
...
ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall"
    Calls "ce1"
    Writes "de1"
...
```

and the numeric datatypes are not compatible.

What to report

Roles to report are
- the <FunctionMethodOrProcedure> function, method, procedure, ...
- the <VariableDataType> numerical data type
- the <CallStatement> call statement with assignment
- the <TargetDataType> second numerical data type

8.20 ASCQM Handle Return Value of Must Check Operations

Descriptor

ASCQM Handle Return Value of Must Check Operations(CallToTheOperation)

Description

Identify occurrences in application model where
- the must-check function, method, procedure, ... is called in the <CallToTheOperation> call statement
- with no use in a conditional statement of the return value

The must-check nature of a function, method, procedure, ... is technology dependent. E.g. in Java: the @CheckReturnValue annotation

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
...
CallableUnit|MethodUnit id="ce1" type="ce1_signature"
attribute="CheckReturnValue|..."
    Signature id="ce1_signature"
        ParameterUnit id="pu1" kind="return"
...
ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall"
...
```

KDM elements absent from the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
StorableUnit id="su1"
...
ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall"
    Writes "su1"
    Flows "ae2"
ActionElement id="ae2" kind="Switch"
    Reads "su1"
    GuardedFlow "gf1"
    GuardedFlow|FalseFlow "gf2"
...
```

or

```
StorableUnit id="su1"
StorableUnit id="su2"
...
ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall"
    Writes "su1"
    Flows "ae2"
ActionElement id="ae2"
kind="Equal|NotEqual|LessThan|LessThanOrEqual|GreaterThan|GreatedThanOrEqual"
    Reads "su1"
    Writes "su2"
    Flows "ae3"
ActionElement id="ae3" kind="Condition"
    TrueFlow "tf1"
    FalseFlow "ff1"
...
```

What to report

Roles to report are
- the <CallToTheOperation> call statement

8.21 ASCQM Check Return Value of Resource Operations Immediately

Descriptor

ASCQM Check Return Value of Resource Operations Immediately(CallToTheOperation)

Description

Identify occurrences in application model where
- a platform resource management function, procedure, method, ... is called in the <CallToTheOperation> call statement
- with no operation performed immediately after on the return value

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
PlatformModel
    ...
    DataManager|ExecutionResource|... id="pr1"
    ...
```

```
    PlatformResource id="pa1"  implementation="ae1"
        ManagesResource|ReadsResource|WritesResource "pr1"
    ...
CodeModel
    CallableUnit|MethodUnit id="ce1" type="ce1_signature"
        Signature id="ce1_signature"
            ParameterUnit id="pu1" kind="return"
    ...
    ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall"
    ...
```

KDM elements absent from the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
StorableUnit id="su1"
...
ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall"
    Writes "su1"
    Flows "ae2"
ActionElement id="ae2"
    Reads "su1"
```

What to report

Roles to report are
- the <CallToTheOperation> call statement8.22

8.22 ASCQM Ban Useless Handling of Exceptions

Descriptor

ASCQM Ban Useless Handling of Exceptions(CatchBlock)

Description

Identify occurrences in application model where
- the <CatchBlock> catch block
- does not report on the error condition as a new throw or as a return value

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
...
CatchUnit id="cu1"
    ...
```

```
...
```

KDM elements absent from the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
...
CatchUnit id="cu1"
    ...
    ActionElement id="ae1" kind="Throw"
        Throws ...
...
```

or

```
...
CatchUnit id="cu1"
    ...
    ActionElement id="ae1" kind="Return"
        Reads ...
...
```

What to report

Roles to report are
- the <CatchBlock> catch block

8.23 ASCQM Ban Incorrect Object Comparison

Descriptor

ASCQM Ban Incorrect Object Comparison(ObjectEqualityComparisonExpression)

Description

Identify occurrences in application model where
- the <ObjectEqualityComparisonExpression> equality comparison expression between two objects

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
ClassUnit id="cu1"
StorableUnit|ItemUnit|MemberUnit id="de1" type="cu1"
```

```
StorableUnit|ItemUnit|MemberUnit id="de2" type="cu1"
ActionElement id="ae1" kind="Equals|NotEqual" ext="de1 == de2 | de1 != de2"
    Reads "de1"
    Reads "de2"
```

What to report

Roles to report are
- the <ObjectEqualityComparisonExpression> equality comparison expression

8.24 ASCQM Ban Assignment Operation Inside Logic Blocks

Descriptor

ASCQM Ban Assignment Operation Inside Logic Blocks(AssignmentExpression, LogicBlock)

Description

Identify occurrences in application model where
- the <AssignmentExpression> assignment expression
- is used within the <LogicBlock> logic block

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
...
ActionElement id="ae1" kind="Compound"
    StorableUnit|MemberUnit id="de1"
    ...
    ActionElement id="ae2" kind="Condition|Switch"
        Reads "de1"
        ActionElement id="ae3" kind="Assign"
            Writes "de1"
...
```

What to report

Roles to report are
- the <AssignmentExpression> assighment expression
- the <LogicBlock> logic block

8.25 ASCQM Ban Comparison Expression Outside Logic Blocks

Descriptor

ASCQM Ban Comparison Expression Outside Logic Blocks(ComparisonExpression)

Description

Identify occurrences in application model where
- the <ComparisonExpression> comparison expression
- is not used within a logic block

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
...
ActionElement id="ae1" kind="Compound"
    StorableUnit|MemberUnit id="de1"
    ...
    ActionElement id="ae3" kind="Equal"
        Reads "de1"
...
```

KDM elements absent from the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
...
ActionElement id="ae1" kind="Compound"
    StorableUnit|MemberUnit id="de1"
    ...
    ActionElement id="ae2" kind="Condition|Switch"
        Reads "su1"
        StorableUnit id="su1" type="register"
        ActionElement id="ae3" kind="Equal"
            Writes "su1"
            Reads "de1"
...
```

What to report

Roles to report are
- the <ComparisonExpression> comparison expression

8.26 ASCQM Ban Incorrect String Comparison

Descriptor

ASCQM Ban Incorrect String Comparison(StringEqualityComparisonExpression)

Description

Identify occurrences in application model where
- the <StringEqualityComparisonExpression> equality comparison expression between two strings

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
StringType id="st1"
StorableUnit|ItemUnit|MemberUnit id="de1" type="st1"
StorableUnit|ItemUnit|MemberUnit id="de2" type="st1"
ActionElement id="ae1" kind="Equals|NotEqual" ext="de1 == de2 | de1 != de2"
    Reads "de1"
    Reads "de2"
```

What to report

Roles to report are
- the <StringEqualityComparisonExpression> equality comparison expression

8.27 ASCQM Ban Logical Operation with a Constant Operand

Descriptor

ASCQM Ban Logical Operation with a Constant Operand(ComparisonExpression)

Description

Identify occurrences in application model where
- the <ComparisonExpression> comparison expression with a constant operand

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
Value id="v1"
...
ActionElement id="ae1" kind="And|Or|Xor"
    Reads "v1"
    ...
```

What to report

Roles to report are
- the <ComparisonExpression> comparison expression

8.28 ASCQM Implement Correct Object Comparison Operations

Descriptor

ASCQM Implement Correct Object Comparison Operations(Class)

Description

Identify occurrences in application model where
- the <Class> class
- lacking the required comparison operations

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
ClassUnit id="cu1"
```

KDM elements absent from the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
BooleanType id="bt1"
IntegerType id="it1"
...
ClassUnit id="cu1"
    ...
    MethodUnit id="mu1" name="equals|Equals|operator==|..."
type="mu1_signature"
        Signature id="mu1_signature"
            ParameterUnit id="pu1" kind="byReference" type="cu1"
            ParameterUnit id="pu2" kind="Return" type="bt1"
    ...
    MethodUnit id="mu2" name="hashCode|GetHashCode|hash|..."
type="mu2_signature"
        Signature id="mu2_signature"
            ParameterUnit id="pu3" kind="byReference" type="cu1"
            ParameterUnit id="pu4" kind="Return" type="it1"
...
```

What to report

Roles to report
- the <Class> class

8.30 ASCQM Ban Comma Operator from Delete Statement

Descriptor

ASCQM Ban Comma Operator from Delete Statement(DeleteStatement, CommaStatement)

Description

Identify occurrences in application model where
- the <DeleteStatement> delete statement
- coumpounded with the <CommaStatement> comma statement

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
...
CallableUnit id="cu1" name="delete" callableKind="operator"
CallableUnit id="cu2" name="comma" callableKind="operator"
...
ActionElement id="ae1" kind="Compound" ext="delete x, y"
    ActionElement id="ae2" kind="Call"
         Calls "cu1"
    ...
    ActionElement id="ae3" kind="Call"
         Calls "cu2"
    ...
...
```

What to report

Roles to report are
- the <DeleteStatement> delete this statement
- the <CommaStatement> comma statement

8.31 ASCQM Release in Destructor Memory Allocated in Constructor

Descriptor

ASCQM Release in Destructor Memory Allocated in Constructor(MemoryAllocationStatement)

Description

Identify occurrences in application model where
- the <MemoryAllocationStatement> memory allocation statement in the class constructor
- lacking a corresponding memory release statement in the class destructor

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
ClassUnit|IntegerType|DecimalType|FloatType|StringType|VoidType|... id="dt1"
PointerType id="pt1"
    ItemUnit id="iu1" type="dt1"
...
ClassUnit id="cu1"
    ...
    StorableUnit id="su1" type="pt1"
    ...
    MethodUnit id="mu1" MethodKind="constructor"
        ...
        ActionElement id="ae1" kind="New|NewArray"
            Creates "dt1"
            Writes "su1"
...
```

or

```
ControlElement id="ce1" name="malloc|calloc|..."
...
ClassUnit|IntegerType|DecimalType|FloatType|StringType|VoidType|... id="dt1"
PointerType id="pt1"
    ItemUnit id="iu1" type="dt1"
...
ClassUnit id="cu1"
    ...
    StorableUnit id="su1" type="pt1"
    ...
    MethodUnit id="mu1" MethodKind="constructor"
        ...
        ActionElement id="ae1" kind="Call"
            Calls "ce1"
            Writes "su1"
...
```

KDM elements absent from the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
ControlElement id="ce2" name="delete|delete[]|free|..."
```

```
...
ClassUnit id="cu1"
    ...
    MethodUnit id="mu2" MethodKind="destructor"
        ...
        ActionElement id="ae2" kind="Call"
             Addresses "su1"
             Calls "ce2"
```

What to report

Roles to report
- the <MemoryAllocationStatement> memory allocation statement

8.32 ASCQM Release Memory after Use with Correct Operation

Descriptor

ASCQM Release Memory after Use with Correct Operation(MemoryAllocationStatement, MemoryReleaseStatement)

Description

Identify occurrences in the application model where
- the memory is allocated via the <MemoryAllocationStatement> allocation statement
- then released via the mismatched <MemoryReleaseStatement> release statement

The pairs of matching allocation/deallocation primitives and operations are technology, framework, language dependant. E.g.: malloc/free, calloc/free, realloc/free in C/C+, new/delete, new[]/delete[] in C+, new/Release() with COM IUnknown interface.

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
ClassUnit|IntegerType|DecimalType|FloatType|StringType|VoidType|... id="dt1"
PointerType id="pt1"
    ItemUnit id="iu1" type="dt1"
...
StorableUnit id="su1" type="pt1"
...
ActionElement id="ae1" kind="New"
    Creates "dt1"
    Writes "su1"
...
ControlElement id="ce2" name="delete[]|free|..."
...
```

```
ActionElement id="ae2" kind="Call"
    Addresses "su1"
    Calls "ce2"
```

or

```
ClassUnit|IntegerType|DecimalType|FloatType|StringType|VoidType|... id="dt1"
PointerType id="pt1"
    ItemUnit id="iu1" type="dt1"
...
StorableUnit id="su1" type="pt1"
...
ActionElement id="ae1" kind="NewArray"
    Creates "dt1"
    Writes "su1"
...
ControlElement id="ce2" name="delete|free|..."
...
ActionElement id="ae2" kind="Call"
    Addresses "su1"
    Calls "ce2"
```

or

```
ControlElement id="ce1" name="malloc|calloc|..."
...
ClassUnit|IntegerType|DecimalType|FloatType|StringType|VoidType|... id="dt1"
PointerType id="pt1"
    ItemUnit id="iu1" type="dt1"
...
StorableUnit id="su1" type="pt1"
...
ActionElement id="ae1" kind="Call"
    Calls "ce1"
    Writes "su1"
...
ControlElement id="ce2" name="delete|delete[]|..."
...
ActionElement id="ae2" kind="Call"
    Addresses "su1"
    Calls "ce2"
```

What to report

Roles to report are
- the <MemoryAllocationStatement> allocation statement
- the <MemoryReleaseStatement> release statement

8.33 ASCQM Implement Required Operations for Manual Resource Management

Descriptor

ASCQM Implement Required Operations for Manual Resource Management(ObjectDeclaration)

Description

Identify occurrences in application model where
- the <ObjectDeclaration> object declaration
- declares an object with manual resource management capabilities
- which lacks the required operation.

The manual resource management capability is technology, framework, and language dependent. E.g.: class inheritance from IDisposable in C#, and AutoClosable in Java, class with __enter__ in python.

KDM outline illustration

KDM elements present in the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
InterfaceUnit id="iu1" name="IDisposable|AutoClosable|..."
...
ClassUnit id="cu1"
     Extends "iu1"
     ...
```

of

```
...
ClassUnit id="cu1"
     MethodUnit "mu1" name="__enter__"
     ...
```

KDM elements absent from the application model
KDM outline illustrating only the essential elements related to micro KDM:

```
ClassUnit id="cu1"
     ...
     MethodUnit "mu1" name="dispose|close|__exit__|..."
```

What to report

Roles to report
- the <ObjectDeclaration> object declaration

8.34 ASCQM Release Platform Resource after Use

Descriptor

ASCQM Release Platform Resource after Use(FunctionProcedureOrMethod, ResourceAllocationStatement, PathToExitWithoutResourceRelease)

Description

Identify occurrences in application model where
- the <FunctionProcedureOrMethod> function, procedure, method, ...
- uses the <ResourceAllocationStatement> resource allocation statement
- excluding memory and file resources
- while there exist the <PathToExitWithoutResourceRelease> path to exit the <FunctionProcedureOrMethod> function, procedure, method, ... without releasing the resource

KDM outline illustration

KDM outline illustrating only the essential elements related to micro KDM:

```
PlatformModel
    ...
    DataManager|ExecutionResource id="pr1"
    ...
    PlatformAction id="pa1" kind="open" implementation="ae1"
        ManagesResource "pr1"
    PlatformAction id="pa2" kind="close" implementation="ae2"
        ManagesResource "pr1"

...
CodeModel
    ...
    CallableUnit|MethodUnit id="ce1"  name="..."
        ...
        ActionElement id="ae1" kind="PlatformAction"
            Flows "ae3"
        ActionElement id="ae3"
             Flows "ae4"
        ActionElement id="ae4" kind="Return"
        ...
        ActionElement id="ae2" kind="PlatformAction"
        ...
...
```

What to report

Roles to report
- the <FunctionProcedureOrMethod> function, procedure, method, ...
- the <ResourceAllocationStatement> file resource open statement
- the <PathToExitWithoutResourceRelease> path to exit

9. Calculation of Quality and Functional Density Measures

9.1 Calculation of the Base Measures (Normative)

After reviewing several alternatives, a count of total violations of quality rules was selected as the best option for a base measure for each of the four software quality characteristics covered in this specification. Software quality characteristic 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 violations of quality rules cannot be isolated to a single component, but rather involve interactions among several components. Therefore, each Automated Source Code Quality Measure score is computed as the sum of its quality measure elements counted across an entire application.

The calculation of an Automated Source Code Quality Measure score progresses as follows:

- Detection pattern score is the count of occurrences,
- Weakness score is its detection pattern score,
- Quality characteristic score is the sum of its weakness scores.

That is,
Occurrence Count of Weakness x = Σ (Occurrences of ASCQM-y)
Where x = a CWE weakness (CWE-119, CWE-120, etc.)
y = a detection pattern for weakness x

and
Occurrence Count of Weakness Category x = Σ (Occurrence Count of ASCQM-y)
Where x = a software quality characteristic (Reliability, Security, Performance Efficiency, Maintainability)
y = a detection pattern for quality characteristic x

9.2 Functional Density of Weaknesses (Non-normative)

In order to compare quality results among different applications, the Automated Source Code Quality Measures can be normalized by size to create a density measure. There are several size measures with which the density of quality 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 the quality of applications. OMG's Automated Function Points (AFP) measure offers an automatable size measure that, as an OMG Supported Specification, is standardized. AFP was adapted from the International Function Point User Group's (IFPUG) counting guidelines, and is commercially supported. Although other size measures can be used to evaluate the density of security violations, the following density measure for quality violations is derived from OMG supported specifications for

Automated Function Points and the Automated Source Code Security Measure. Thus, the functional density of Security violations is a simple division expressed as follows.

ASCxM-density = ASCxM / AFP

where x = a software quality characteristic (R, S, PE, M)

10. Alternative Weighted Measures and Uses (Informative)

10.1 Additional Derived Measures

There are many additional weighting schemes that can be applied to the Automated Source Code Quality Measures or to the quality measure elements that composing them. Table 6 presents several weighted measure candidates and their potential uses. However, these weighting schemes are not derived from any existing standards and are therefore not normative.

Table 6. Informative Weighting Schemes for Security Measurement

Weighting scheme	Potential uses
Weight each quality measure element by its severity	Measuring risk of quality problems such as data theft, outages, response degradation, etc.
Weight each quality measure element by its effort to fix	Measuring cost of ownership, estimating future corrective maintenance effort and costs
Weight each module or application component by its density of quality weaknesses	Prioritizing modules or application components for corrective maintenance or replacement

11. References (Informative)

Common Weakness Enumeration. http://cwe.mitre.org . Bedford, MA: MITRE Corporation.

Consortium for IT Software Quality (2010). http://www.it-cisq.org . Needham, MA: Object Management Group, Consortium for IT Software Quality (CISQ).

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

International Organization for Standards (2007). ISO/IEC 25020 Systems and software engineering: Systems and software Quality Requirements and Evaluation (SQuaRE) – Measurement of system and software product quality – Measurement reference model and guide. Geneva, Switzerland.

International Organization for Standards (2011). ISO/IEC 25010:2011 Systems and software engineering – System and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models. Geneva, Switzerland.

International Organization for Standards (2011). ISO/IEC 25020:2007 Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Measurement reference model and guide.

International Organization for Standards (2012). ISO/IEC 25023 Systems and software engineering: Systems and software Quality Requirements and Evaluation (SQuaRE) – Measurement of system and software product quality. Geneva, Switzerland.

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

Martin, R.A. & Barnum, S. (2006). *Status update: The Common Weakness Enumeration*. NIST Static Analysis Summit, Gaithersburg, MD Jun 29, 2006.

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

Appendix A: Consortium for IT Software Quality (CISQ)

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

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

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

The Consortium for IT Software Quality (CISQ), a consortium managed by OMG, was formed in 2010 to create international standards for automating measures of size and structural quality characteristics from source code. These measures are intended for use by IT organizations, IT service providers, and software vendors in contracting, developing, testing, accepting, and deploying software systems. Executives from the member companies that joined CISQ prioritized Reliability, Security, Performance Efficiency, and Maintainability as the initial structural quality measures to be specified.

An international team of experts drawn from CISQ's 24 original companies formed into working groups to define CISQ measures. Weaknesses that had a high probability of causing reliability, security, performance efficiency, or maintainability problems were selected for inclusion in the four measures. The original CISQ members 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 weaknesses. This list was pared down to a set of weaknesses they believed had to be remediated to avoid serious operational or cost problems. These 86 weaknesses became the

foundation of the original specifications of the automated source code measures for Reliability, Security, Performance Efficiency, and Maintainability.

Appendix B: Common Weakness Enumeration (CWE)

The Common Weakness Enumeration (CWE) repository (http://cwe.mitre.org/) maintained by MITRE Corporation is a collection of over 800 weaknesses in software architecture and source code that malicious actors have used to gain unauthorized entry into systems or to cause malicious actions. The CWE is a widely used industry source (http://cwe.mitre.org/community/citations.html) that provides a foundation for an ITU and ISO/IEC standard, in addition to 2 ISO/IEC technical reports:

- SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY Cybersecurity information exchange – Vulnerability/state exchange - Common weakness enumeration (CWE)
- ISO/IEC 29147:2014 Information Technology -- Security Techniques -- Vulnerability Disclosure"
- ISO/IEC TR 24772:2013 Information technology -- Programming languages -- Guidance to avoiding vulnerabilities in programming languages through language selection and use
- ISO/IEC Technical Report is ISO/IEC TR 20004:2012 Information Technology -- Security Techniques -- Refining Software Vulnerability Analysis under ISO/IEC 15408 and ISO/IEC 18045

The CWE/SANS Institute Top 25 Most Dangerous Software Errors is a list of the 25 most widespread and frequently exploited security weaknesses in the CWE repository. The previous version of the CISQ Automated Source Code Security Measure (ASCSM) was based on 22 of the CWE/SANS Top 25 that could be detected and counted in source code. In this revision, the number of security weaknesses is being expanded beyond the CWE/SANS Top 25 since there are other weaknesses severe enough to be incorporated in the CISQ measure. In addition, many CWEs also cause reliability problems and are therefore included in the CISQ reliability measure. Wherever a CWE is included in any of the 4 CISQ structural quality measures, its CWE identifier will be noted.

Since the CWE is recognized as the primary industry repository of security weaknesses, it is supported by the majority of vendors providing tools and technology in the software security domain (http://cwe.mitre.org/compatible/compatible.html), such as Coverity, HP Fortify, Klockwork, IBM, CAST, Veracode, and others. These vendors already have capabilities for detecting many of the CWEs. Industry experts who developed the CWE purposely worded the CWEs to be language and application agnostic in order to allow vendors to develop detectors specific to a wide range of languages and application types beyond the scope that could be covered in the CWE. Since some of the CWEs may not be relevant in some languages, the reduced opportunity for anti-patterns in those cases will be reflected in the scores.

Appendix C: Disposition of Weaknesses from the Original CISQ Measures to This Specification

Maintainability Measure

CISQ identifier	Disposition
ASCMM-MNT-1	CWE-1075
ASCMM-MNT-2	CWE-1055
ASCMM-MNT-3	CWE-1052
ASCMM-MNT-4	CWE-1048
ASCMM-MNT-5	CWE-1095
ASCMM-MNT-6	CWE-1085
ASCMM-MNT-7	CWE-1047
ASCMM-MNT-8	CWE-1080
ASCMM-MNT-9:	CWE-424
ASCMM-MNT-10	CWE-424
ASCMM-MNT-11	CWE-1093
ASCMM-MNT-12	CWE-1054
ASCMM-MNT-13	CWE-1064
ASCMM-MNT-14	CWE-1084
ASCMM-MNT-15	CWE-1081
ASCMM-MNT-16	CWE-1090
ASCMM-MNT-17	CWE-1074
ASCMM-MNT-18	CWE-1086
ASCMM-MNT-19	CWE-1041
ASCMM-MNT-20	CWE-1061

Performance Efficiency Measure

CISQ identifier	Disposition
ASCPEM-PRF-1	Dropped
ASCPEM-PRF-2	CWE-1046
ASCPEM-PRF-3	CWE-1042
ASCPEM-PRF-4	CWE-1049
ASCPEM-PRF-5	CWE-1067
ASCPEM-PRF-6	CWE-1089
ASCPEM-PRF-7	CWE-1094
ASCPEM-PRF-8	CWE-1050
ASCPEM-PRF-9	CWE-1060
ASCPEM-PRF-10	CWE-1073
ASCPEM-PRF-11	CWE-1057
ASCPEM-PRF-12	CWE-1043
ASCPEM-PRF-13	CWE-1072
ASCPEM-PRF-14	CWE-1071
ASCPEM-PRF-15	CWE-1091

Reliability Measure

CISQ identifier	Disposition
ASCRM-CWE-120	Retained - child of CWE-119
ASCRM-CWE-252data	Dropped
ASCRM-CWE-252resource	Dropped
ASCRM-CWE-396	Dropped
ASCRM-CWE-397	Dropped
ASCRM-CWE-456	Retained
ASCRM-CWE-674	Dropped
ASCRM-CWE-704	Retained
ASCRM-CWE-772	Retained - child of CWE-404
ASCRM-CWE-788	Retained – child of CWE-119
ASCRM-RLB-1	Dropped
ASCRM-RLB-2	CWE-1066
ASCRM-RLB-3	CWE-1070
ASCRM-RLB-4	CWE-1097
ASCRM-RLB-5	CWE-404
ASCRM-RLB-6	CWE-1098
ASCRM-RLB-7	CWE-1082
ASCRM-RLB-8	Dropped
ASCRM-RLB-9	CWE-1077
ASCRM-RLB-10	CWE-1057
ASCRM-RLB-11	CWE-1058
ASCRM-RLB-12	CWE-1096
ASCRM-RLB-13	Moved to Maintainability
ASCRM-RLB-14	CWE-1062
ASCRM-RLB-15	CWE-1087

ASCRM-RLB-16	CWE-1079
ASCRM-RLB-17	CWE-1045
ASCRM-RLB-18	CWE-1051
ASCRM-RLB-19	CWE-1088

Security

CISQ identifier	Disposition
ASCSM-CWE-22	Retained
ASCSM-CWE-78	Retained
ASCSM-CWE-79	Retained
ASCSM-CWE-89	Retained
ASCSM-CWE-99	Retained
ASCSM-CWE-120	Retained
ASCSM-CWE-129	Retained
ASCSM-CWE-134	Retained
ASCSM-CWE-252resource	Retained as CWE-252
ASCSM-CWE-327	Dropped
ASCSM-CWE-396	Dropped
ASCSM-CWE-397	Dropped
ASCSM-CWE-434	Retained
ASCSM-CWE-456	Retained
ASCSM-CWE-606	Retained
ASCSM-CWE-667	Retained
ASCSM-CWE-672	Retained
ASCSM-CWE-681	Retained
ASCSM-CWE-772	Retained
ASCSM-CWE-789	Retained
ASCSM-CWE-798	Retained
ASCSM-CWE-835	Retained

Appendix D: Relationship of the CISQ Structural Quality Measures to ISO 25000 Series Standards (SQuarE)

ISO/IEC 25010 defines the product quality model for software-intensive systems (Figure 1). This model is composed of 8 quality characteristics, four of which are the subject of CISQ structural quality measures (indicated in blue). Each of ISO/IEC 25010's eight quality characteristics consists of several quality sub-characteristics that define the domain of issues covered by their parent quality characteristic. CISQ structural quality measures conform to the definitions in ISO/IEC 25010. The sub-characteristics of each quality characteristic were used to ensure the CISQ measures covered the domain of issues in each of the four areas. ISO/IEC 25010 is currently undergoing revision with CISQ participation. The CISQ measures will conform with definitions in the revised ISO/IEC 25010-2 when published.

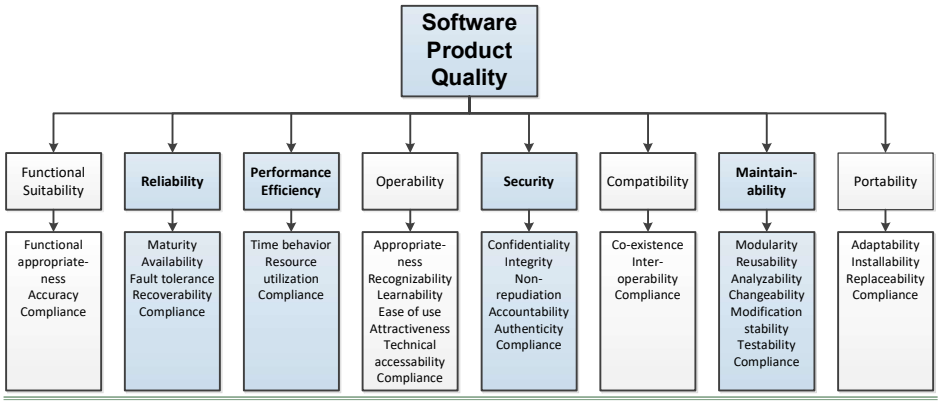


Figure 1. Software Quality Characteristics from ISO/IEC 25010 with CISQ measure areas highlighted.

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 2 presents an example of the ISO/IEC 25023 quality measurement framework using a partial decomposition for the Automated Source Code Security Measure.

Figure 2 displays the hierarchical relationships indicating how CISQ conforms to the reference measurement structure established in ISO/IEC 25020 that governs software quality measures in ISO/IEC 25023. This structure is presented using the CISQ Security measure as an example. The CISQ measures only use ISO's quality subcharacteristics for ensuring that the CISQ weaknesses covered the measurable domain of an ISO quality characteristic as defined in ISO/IEC 25010. CISQ's weaknesses (CWEs) correspond to ISO's quality attributes. CISQ weaknesses are represented as one or more detection

patterns among structural code elements in the software. Variations in how a weakness may be instantiated are represented by its association with several different detection patterns. Each occurrence of a detection pattern represents an occurrence of a weakness in the software. Occurrences of these detection patterns in the software correspond to ISO's quality measure elements and are the elements calculated in the CISQ measures.

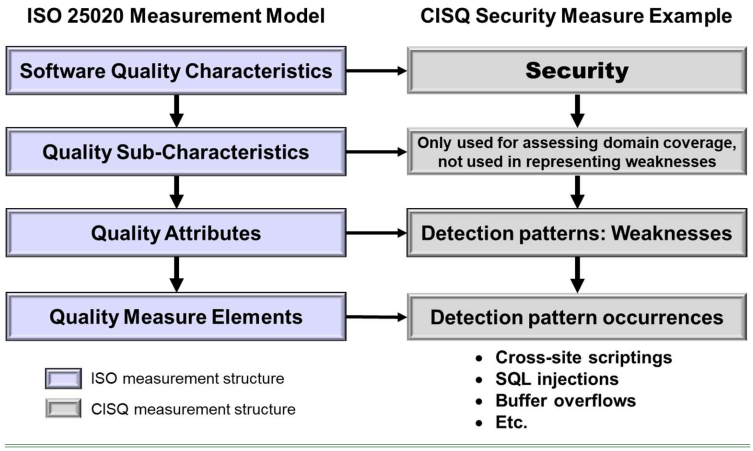


Figure 2. ISO/IEC 250210 Framework for Software Quality Characteristics Measurement

Clause 6 of this specification lists weaknesses grouped by quality characteristic that correspond to ISO/IEC 25020's quality attributes. A weakness is detected by identifying patterns of code elements in the software (called detection patterns) that instantiate the weakness. Each detection pattern equates to a quality measure element used in calculating the CISQ quality measures. In Clause 7, quality attributes (weaknesses) are transformed into the KDM and SPMS-based detection patterns that represent them. The CISQ quality measures are then calculated by detecting and counting occurrences of detection patterns, each of which indicates the existence of a weakness in the software. These calculations are represented in the Structured Metrics Metamodel (SMM).