January 2024

Automated Source Code Resource Sustainability Measure (ASCRSM),

Version 1.0

OMG Document Number: formal/24-01-09

Document URL: https://www.omg.org/spec/ASCRSM/

USE OF SPECIFICATION – TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group 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 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

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

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 https://www.omg.org, under Specifications, Report a Bug/Issue.

Table of Contents

Preface

About the Object Management Group

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 *https://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: *https://www.omg.org/spec*

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
9C Medway Road, PMB 274
Milford, MA 01757
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: https://www.omg.org/report_issue.htm

1 Scope

1.1 Purpose

This specification is derived from the Automated Source Code Performance Efficiency Measure and Automated Source Code Reliability Measure both included in the Automated Source Code Quality Measures (ASCQM) specification (https://www.omg.org/spec/ASCQM/1.0/ and ISO/IEC 5055:2021) to cover common weaknesses (CWEs) that affect the use of energy and other resources. Specifying this measure is important as a source of evidence for complying with emerging regulations and corporate policies regarding reductions in resource usage. This measure is calculated from detecting and counting 40 violations of good architectural and coding practices (weaknesses) in the source code that could result in excessive or unnecessary processing or failures that cause hardware reboots.

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. They also provide evidence of compliance with regulations governing various aspects of software system performance.

Currently there are no standards for most of the software structural quality measures. 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. This specification addresses one aspect of this problem by providing a specification for measuring attributes of the software that affect the efficient use of resources, often referred to as 'Green IT'.

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 Resource Sustainability Measure is a correlated measure rather than an absolute measure of excessive resource usage. That is, since it does not measure all possible resource usage weaknesses, it does not provide an absolute measure of resource inefficiency. However, since it includes counts of what industry experts have determined to be the most severe weaknesses, it provides a strong indicator of the resource inefficiency of a software system. In most instances it will be highly correlated with the probability of inefficient resource usage.

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 presentation of automated techniques for detecting each weakness. 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 detectable by one or more Detection Patterns. Many quality measure elements (weaknesses) will share one or 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 triggered for each weakness, then summing these numbers for all weaknesses included in the Automated Source Code Resource Sustainability Measure.

Each instantiation of a weakness triggers only one of a weakness's detection patterns if multiple detection patterns are relevant to a weakness. Clauses 9 and 10 will present several methods for normalizing the results of evaluating this measure.

2 Conformance

Implementations of this specification shall 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 shall 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 shall not require further human intervention. The analysis and calculation shall be able to repeatedly produce the same results and outputs on the same body of software.
- ***Transparent*** — Implementations that conform to this specification shall 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*** — Implementations of this specification shall state the assumptions and heuristics it uses with sufficient detail so that the calculations may be independently verified by third parties. In addition, all inputs used shall be clearly described and itemized so that they can be audited by a third party.

3 References

The following normative documents contain provisions, which, through reference in this text, constitute provisions of this specification. 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/
- Knowledge Discovery Metamodel, version 1.4 (KDM), https://www.omg.org/spec/KDM/1.4
- Structured Metrics Metamodel, version 1.2 (SMM), formal/2012-01-05
- MOF/XMI Mapping, version 2.5.1 (XMI), https://www.omg.org/spec/XMI/2.5.1/
- 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
- ISO/IEC 5055:2021 OMG Automated Source Code Quality Measures
- ITU-T X.1524 – Series X: Data Networks, Open System Communications and Security – Cybersecurity information exchange – Vulnerability/state exchange – Common weakness enumeration

4 Terms and definitions

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

Common Weakness Enumeration — 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 — 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*).

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

Parent Weakness — 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 — 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*).

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

Software Product Quality Model — model that categorizes software product quality properties into eight characteristics (functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability, and portability) (*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 — 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 — set of software quality attributes that affect a specific category of software quality outcomes (*derived from ISO/IEC 25010*).

Software Quality Characteristic Measure — software quality measure derived from measuring the attributes related to a specific software quality characteristic (*ISO/IEC 25020*).

Software Quality Measure — 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 — 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 — set of operations having the object of determining a value of a software quality measure (*ISO/IEC 25010*).

Software Quality Model — 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 — architectural or coding practice or convention that represents good software engineering practice and avoids problems in software development, maintenance, or operations.

Software Quality Sub-characteristic — 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 — component of software code that can be uniquely identified and counted such as a token, decision, or variable.

Structural Quality — 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 (*derived from ISO/IEC 25010*).

Weakness — pattern or structure in the code (Detection Pattern in ASCRSM) that is inconsistent with good architectural or coding practice, violates a software quality rule, and can lead to operational or cost problems (*derived from cwe.mitre.com*).

5 Symbols

ASCPEM — Automated Source Code Performance Efficiency Measure

ASCQM — Automated Source Code Quality Measure

ASCRM — Automated Source Code Reliability Measure

ASCRSM — Automated Source Code Resource Sustainability Measure

CWE — Common Weakness Enumeration

CISQ — Consortium for Information and 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.

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

6.2 Automated Source Code Quality Measure Elements

The weaknesses violating software quality rules that compose the CISQ Automated Source Code Resource Sustainability Measure are grouped by measure in the clauses 6 and 7. The Common Weakness Enumeration repository (CWE, Appendix B) has recently been expanded to include weaknesses from quality characteristics beyond security. All weaknesses included in this measure are identified by their CWE number from the repository. In most cases the description of CWEs is taken from information in the online repository (cwe.mitre.org). Most of the weaknesses included in this measure have been drawn from the four measures in OMG's Automated Source Code Quality Measures (ASCQM). The mapping of the weaknesses from the ASCQM to this measure 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 Resource Sustainability Measure Element Descriptions

The quality measure elements (weaknesses violating software quality rules) that compose the CISQ Automated Source Code Resource Sustainability Measure are presented in Table 1 with their CWE identifier from the Common Weakness Enumeration Repository, their title, and a description of the weakness.

This measure contains 33 weaknesses. The final column lists measures from the Automated Source Code Quality Measures standard (also ISO 5055:2021) that included the weakness in its calculation. Normative descriptions of the weaknesses in Clause 7 will include partial information on the status of some weaknesses as being 'Parents' of other weaknesses (high level descriptions of a tightly related class of weakness), or of being 'Contributing' weaknesses which represent different instantiations of their parent weakness.

Table 1: Quality Measure Elements for Automated Source Code Resource Sustainability Measure

CWE ID	Weakness title	Weakness description	ASCQM measure
248	**Uncaught Exception**	An exception is thrown from a function, but it is not caught.	Reliability
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.	Reliability Security
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.	Reliability
391	**Unchecked Error Condition**	Ignoring exceptions and other error conditions may allow an attacker to induce unexpected behavior unnoticed.	Reliability
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.	Reliability
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.	Reliability
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.	Reliability Security Performance
404	**Improper Resource Shutdown or Release**	The program does not release or incorrectly releases a resource before it is made available for re-use.	Reliability Security Performance
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. When data storage relies on a DBMS, special care shall be given to secure all data accesses and ensure data integrity.	Reliability Security Performance
459	**Incomplete Cleanup**	The software does not properly "clean up" and remove temporary or supporting resources after they have been used.	Reliability
703	**Improper Check or Handling of Exceptional Conditions**	The software does not properly anticipate or handle exceptional conditions that rarely occur during normal operation of the software.	Reliability

762	**Mismatched Memory Management Routines**	The application attempts to return a memory resource to the system, but it calls a release function that is not compatible with the function that was originally used to allocate that resource.	not in ASCQM
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.	Reliability Security Performance
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.	Reliability Security Performance
833	**Deadlock**	The software contains multiple threads or executable segments that are waiting for each other to release a necessary lock, resulting in deadlock.	Reliability
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.	Reliability Security
1043	**Data Element Aggregating an Excessively Large Number of Non-Primitive Elements**	The software uses a data element that has an excessively large number of sub-elements with non-primitive data types such as structures or aggregated objects. (default threshold for the maximum number of aggregated non-primitive data types is 5, *alternate threshold can be set prior to analysis*).	Performance
1046	**Creation of Immutable Text Using String Concatenation**	This programming pattern can be inefficient in comparison with use of text buffer data elements. This issue can make the software perform more slowly. If the relevant code is reachable by an attacker, then this performance problem might introduce a vulnerability.	Performance
1049	**Excessive Data Query Operations in a Large Data Table**	The software performs a data query with a large number of joins and sub-queries on a large data table. (default thresholds are 5 joins, 3 sub-queries, and 1,000,000 rows for a large table, *alternate thresholds for all three parameters can be set prior to analysis*).	Performance
1050	**Excessive Platform Resource Consumption within a Loop**	The software has a loop body or loop condition that contains a control element that directly or indirectly consumes platform resources, e.g., messaging, sessions, locks, or file descriptors. (default threshold for resource consumption should be set based on the system architecture prior to analysis).	Performance

1051	**Initialization with Hard-Coded Network Resource Configuration Data**	The software initializes data using hard-coded values that act as network resource identifiers.	Reliability Maintenance
1057	**Data Access Operations Outside of Expected Data Manager Component**	The software uses a dedicated, central data manager component as required by design, but it contains code that performs data-access operations that do not use this data manager. 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 weakness. · 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.	Security Performance
1060	**Excessive Number of Inefficient Server-Side Data Accesses**	The software performs too many data queries without using efficient data processing functionality such as stored procedures. (default threshold for maximum number of data queries is 5, alternate threshold can be set prior to analysis).	Performance
1067	**Excessive Execution of Sequential Searches of Data Resource**	The software contains a data query against a SQL table or view that is configured in a way that does not utilize an index and may cause sequential searches to be performed. (default threshold for a weakness to be counted is a query on a table of at least 500 rows, or an alternate threshold recommended by the database vendor. No weakness should be counted under conditions where the vendor recommends an index should not be used. An alternate threshold can be set prior to analysis).	Performance
1069	**Empty Exception Block**	An invokable code block contains an exception handling block that does not contain any code, i.e., is empty.	not in ASCQM
1072	**Non-SQL Invokable Control Element with Excessive Number of Data Resource Accesses**	The software contains a client with a function or method that contains a large number of data accesses/queries that are sent through a data manager, i.e., does not use efficient database capabilities. (default threshold for the maximum number of data queries is 2, alternate threshold can be set prior to analysis).	Performance

1073	**Non-SQL Invokable Control Element with Excessive Number of Data Resource Accesses**	The software contains a client with a function or method that contains a large number of data accesses/queries that are sent through a data manager, i.e., does not use efficient database capabilities.	Performance
1083	**Data Access from Outside Designated Data Manager Component**	The software is intended to manage data access through a particular data manager component such as a relational or non-SQL database, but it contains code that performs data access operations without using that component. 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.	Reliability
1088	**Synchronous Access of Remote Resource without Timeout**	The code has a synchronous call to a remote resource, but there is no timeout for the call, or the timeout is set to infinite.	Reliability
1089	**Large Data Table with Excessive Number of Indices**	The software uses a large data table (default is 1,000,000 rows; alternate threshold can be set prior to analysis) that contains an excessively large number of indices. (default threshold for the maximum number of indices is 3, alternate threshold can be set prior to analysis).	Performance
1091	**Use of Object without Invoking Destructor Method**	The software contains a method that accesses an object but does not later invoke the element's associated finalize/destructor method.	Performance
1094	**Excessive Index Range Scan for a Data Resource**	The software contains an index range scan for a large data table, (default threshold is 1,000,000 rows, alternate threshold can be set prior to analysis) but the scan can cover a large number of rows. (default threshold for the index range is 10, alternate threshold can be set prior to analysis).	Performance
1235	**Incorrect Use of Autoboxing and Unboxing for Performance Critical Operations**	The code uses boxed primitives, which may introduce inefficiencies into performance-critical operations.	not in ASCQM

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 measure is 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 of detection patterns try to remain as generic, yet as accurate as possible, so that the detection pattern can be applied to as many situations as possible such as different technologies and different programming languages. 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.

Detection Patterns presented in Clause 8 use micro-KDM to provide greater granularity to their specification of weakness patterns. 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 2: 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”
    - “Maintainability”

6.7 Specification of Detection Patterns

Detection patterns provide guidance for automated detection of the weaknesses enumerated in Clause 7. Each weakness may have several different instantiations in the source code. Thus, a weakness may be associated with several different detection patterns. Each detection pattern may be associated with weaknesses in several different quality measures. There are 78 detection patterns associated with the weaknesses in Automated Source Code Resource Sustainability Measures. This number will grow as more detection patterns are discovered and specified.

Detection Patterns use micro-KDM to provide greater granularity to their specification of weakness patterns. 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. Micro-KDM constrains the patterns of how to map the statements of the existing system as determined by the programming language into KDM.

6.8 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-clauses 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

This page intentionally left blank.

7 ASCRSM Weakness Specifications (Normative)

7.1 CWE-248 Uncaught Exception

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

Roles
- the <ExceptionThrowDeclaration>
- the <ExceptionCatchSequence>

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

Detection Patterns
ASCQM Catch Exceptions

7.2 CWE-252 Unchecked Return Value

Reference
https://cwe.mitre.org/data/definitions/252

Roles
- the <OperationCall>

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

7.3 CWE-390 Detection of Error Condition Without Action

Reference
https://cwe.mitre.org/data/definitions/390

Roles
- the <ErrorCondition>

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

7.4 CWE-391 Unchecked Error Condition

Reference
https://cwe.mitre.org/data/definitions/391

Roles
- the <ErrorConditionProcessing>

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

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

7.5 CWE-392 Missing Report of Error Condition

Reference
https://cwe.mitre.org/data/definitions/392

Roles
- the <ErrorConditionProcessing>

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

Detection Patterns
ASCQM Ban Useless Handling of Exceptions

7.6 CWE-394 Unexpected Status Code or Return Value

Reference
https://cwe.mitre.org/data/definitions/394

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

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

Reference
https://cwe.mitre.org/data/definitions/401

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 Memory after Use with Correct Reference
ASCQM Release Platform Resource after Use
ASCQM Release in Destructor Memory Allocated in Constructor

7.8 CWE-404 Improper Resource Shutdown or Release

Reference
https://cwe.mitre.org/data/definitions/404

Roles
- the <ResourceAllocation>

Contributing weaknesses
CWE-401 Improper Release of Memory Before Removing Last Reference ('Memory Leak')
CWE-762 Mismatched Memory Management Routines
CWE-772 Missing Release of Resource after Effective Lifetime
CWE-775 Missing Release of File Descriptor or Handle after Effective Lifetime

Detection Patterns
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 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 Memory after Use with Correct Reference
ASCQM Release Platform Resource after Use
ASCQM Release in Destructor Memory Allocated in Constructor

7.9 CWE-424 Improper Protection of Alternate Path

Reference
https://cwe.mitre.org/data/definitions/424

Roles
- the <AlternatePath>

Detection Patterns
ASCQM Ban Unintended Paths

7.10 CWE-459 Incomplete Cleanup

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

Roles
- the <ResourceAllocation>
- the <ResourceRelease>

Detection Patterns
ASCQM Release Memory after Use with Correct Operation
ASCQM Release Memory after Use with Correct Reference

7.11 CWE-703 Improper Check or Handling of Exceptional Conditions

Reference
https://cwe.mitre.org/data/definitions/703

Roles
- the <ErrorHandling>

Contributing weaknesses
CWE-248 Uncaught Exception
CWE-391 Unchecked Error Condition
CWE-392 Missing Report of Error Condition

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

7.12 CWE-762 Mismatched Memory Management Routines

Reference
https://cwe.mitre.org/data/definitions/762

Roles
- the <MemoryAllocation>
- the <MemoryRelease>

Parent weaknesses
CWE-404 Improper Resource Shutdown or Release

Detection Patterns
ASCQM Release Memory after Use with Correct Operation

7.13 CWE-772 Missing Release of Resource after Effective Lifetime

Reference
https://cwe.mitre.org/data/definitions/772

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

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

Reference
https://cwe.mitre.org/data/definitions/775

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

7.15 CWE-833 Deadlock

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

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

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

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

Reference
https://cwe.mitre.org/data/definitions/835

Roles
- the <InfiniteLoop>

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

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

Usage name
Excessively large data element

Reference
https://cwe.mitre.org/data/definitions/1043

Roles
- the <AggregationData>
- the <AggregatedData>

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

7.18 CWE-1046 Creation of Immutable Text Using String Concatenation

Usage name
Immutable text data

Reference
https://cwe.mitre.org/data/definitions/1046

Roles
- the <ImmutableDataCreation>

Detection Patterns
ASCQM Ban Incrementral Creation of Immutable Data

7.19 CWE-1049 Excessive Data Query Operations in a Large Data Table

Usage name
Complex read/write access

Reference
https://cwe.mitre.org/data/definitions/1049

Roles
- the <DataQuery>

Detection Patterns
ASCQM Ban Excessive Complexity of Data Resource Access

7.20 CWE-1050 Excessive Platform Resource Consumption within a Loop

Usage name
Resource consuming operation in loop

Reference
https://cwe.mitre.org/data/definitions/1050

Roles
- the <Loop>
- the <ExpensiveOperation>

Detection Patterns
ASCQM Ban Expensive Operations in Loops

7.21 CWE-1051 Initialization with Hard-Coded Network Resource Configuration Data

Usage name
Hard-coded network resource information

Reference
https://cwe.mitre.org/data/definitions/1057

Roles
- the <DataManager>
- the <DataAccess>

Detection Patterns
ASCQM Ban Unintended Path

7.22 CWE-1057 Data Access Operations Outside of Designated Data Manager Component

Usage name
Circumventing data access routines

Reference
https://cwe.mitre.org/data/definitions/1057

Roles
- the <DataManager>
- the <DataAccess>

Detection Patterns
ASCQM Ban Unintended Path

7.23 CWE-1060 Excessive Number of Inefficient Server-Side Data Accesses

Usage name
Excessive data queries in non-stored procedure

Reference
https://cwe.mitre.org/data/definitions/1060

Roles
- the <NonStoredSQLOperation>
- the <DataAccesses>

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

7.24 CWE-1067 Excessive Execution of Sequential Searches of Data Resource

Usage name
Incorrect indicies

Reference
https://cwe.mitre.org/data/definitions/1067

Roles
- the <DataQuery>
- the <TableOrView>

Detection Patterns
ASCQM Implement Index Required by Query on Large Tables

7.25 CWE-1069 Empty Exception Block

Reference
https://cwe.mitre.org/data/definitions/1069

Roles
- the <ErrorConditionProcessing>

Detection Patterns
ASCQM Ban Empty Exception Block

7.26 CWE-1072 Data Resource Access without use of Connection Pooling

Usage name
Data access not using connection pool

Reference
https://cwe.mitre.org/data/definitions/1072

Roles
- the <Connection>

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

7.27 CWE-1073 Non-SQL Invokable Control Element with Excessive Number of Data Resource Access

Usage name
Excessive data queries in client-side code

Reference
https://cwe.mitre.org/data/definitions/1073

Roles
- the <NonSQLOperation>
- the <DataAccesses>

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

7.28 CWE-1083 Data Access from Outside Designated Data Manager Component

Usage name
Circumventing data access routines

Reference
https://cwe.mitre.org/data/definitions/1083

Roles
- the <DataManager>
- the <DataAccess>

Detection Patterns
ASCQM Ban Unintended Paths

7.29 CWE-1088 Synchronous Access of Remote Resource without Timeout

Usage name
Synchronous call with missing timeout

Reference
https://cwe.mitre.org/data/definitions/1088

Roles
- the <SynchronousCall>
- the <TimeOutOption>

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

7.30 CWE-1089 Large Data Table with Excessive Number of Indices

Usage name

Excessive number of indices on large tables

Reference
https://cwe.mitre.org/data/definitions/1089

Roles
- the <Table>
- the <Indexes>

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

7.31 CWE-1091 Use of Object without Invoking Destructor Method

Reference
https://cwe.mitre.org/data/definitions/1091

Roles
- the <Object>

Detection Patterns
ASCQM Release Memory after Use with Correct Operation

7.32 CWE-1094 Excessive Index Range for a Data Resource

Usage name
Excessively large indices on large tables

Reference
https://cwe.mitre.org/data/definitions/1094

Roles
- the <Table>
- the <Indexes>

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

7.33 CWE-1235 Incorrect Use of Autoboxing and Unboxing for Performance Critical Operations

Reference
https://cwe.mitre.org/data/definitions/1235

Roles
- the <Autoboxing/Unboxing>

Detection Patterns
ASCQM Ban Autoboxing in Loops
ASCQM Ban Unboxing in Loops

7.34 ASCRSM Detection Patterns

ASCQM Ban Autoboxing in Loops
ASCQM Ban Comma Operator from Delete Statement
ASCQM Ban Empty Exception Block
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 Hard-Coded Literals used to Connect to Resource
ASCQM Ban Incompatible Lock Acquisition Sequences
ASCQM Ban Incorrect Numeric Conversion of Return Value
ASCQM Ban Incremental Creation of Immutable Data
ASCQM Ban Unboxing in Loops
ASCQM Ban Unintended Paths
ASCQM Ban Unmodified Loop Variable Within Loop
ASCQM Ban Use of Prohibited Low-Level Resource Management Functionality
ASCQM Ban Use of Thread Control Primitives with Known Deadlock Issues
ASCQM Ban Useless Handling of Exceptions
ASCQM Ban While TRUE Loop Without Path To Break
ASCQM Catch Exceptions
ASCQM Check Return Value of Resource Operations Immediately
ASCQM Handle Return Value of Must Check Operations
ASCQM Handle Return Value of Resource Operations
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 Manage Time-Out Mechanisms in Blocking Synchronous Calls
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 Memory after Use with Correct Reference
ASCQM Release Platform Resource after Use
ASCQM Release in Destructor Memory Allocated in Constructor

8 ASCRSM Weakness Detection Patterns (Normative)

8.1 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

8.2 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. For example, 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

8.3 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

8.4 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

8.5 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

8.6 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
- compounded 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

8.7 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

8.8 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. For example: 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

8.9 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. For example: 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

8.10 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

8.11 ASCQM Release Memory After Use

Descriptor
ASCQM Release Memory After Use(MemoryAllocationStatement)

Description
Identify occurrences in application model where:
- the <MemoryAllocationStatement> memory allocation statement
- lacking a corresponding memory release statement

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"
...
StorableUnit id="su1" type="pt1"
...
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"
...
StorableUnit id="su1" type="pt1"
...
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|..."
  ...
  ActionElement id="ae2" kind="Call"
      Addresses "su1"
     Calls "ce2"
```

What to report

Roles to report :
- the <MemoryAllocationStatement> memory allocation statement

8.12 ASCQM Implement Virtual Destructor for Classes Derived from Class with Virtual Destructor

Descriptor

ASCQM Implement Virtual Destructor for Classes Derived from Class with Virtual Destructor(Class, ParentClass, ParentVirtualDestructor)

Description

Identify occurrences in application model where :
- the <Class> class
- inherits from the <ParentClass> parent class
- with the <ParentVirtualDestructor> virtual destructor
- but lacks a virtual destructor

KDM outline illustration

KDM elements present in the application model

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

```
ClassUnit id="c1"
     ....
    MethodUnit is="m1"  methodKind="method" isVirtual="true"
        ...
ClassUnit id="c2"  InheritsFrom="c1"
     ...
```

KDM elements absent from the application model

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

```
ClassUnit id="c2"
     ....
     MethodUnit  is="m2"  methodKind="destructor"  isVirtual="true"
          ...
```

What to report
Roles to report are :
- the <Class> class
- the <ParentClass> parent class
- the <ParentVirtualDestructor> virtual destructor

8.13 ASCQM Implement Virtual Destructor for Parent Classes

Descriptor
ASCQM Implement Virtual Destructor for Parent Classes(Class, ParentClass)

Description
Identify occurrences in application model where:
- the <Class> class
- inherits from the <ParentClass> parent class
- which lacks a virtual destructor

KDM outline illustration
KDM elements present in the application model

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

```
ClassUnit id="c1"
     ....
ClassUnit id="c2"  InheritsFrom="c1"
     ...
```

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

```
  ClassUnit id="c1"
      ....
      MethodUnit is="m1"  methodKind="method" isVirtual="true"
          ...
```

What to report
Roles to report are:
- the <Class> class
- the <ParentClass> parent class

8.14 ASCQM Release File Resource after Use in Operation

Descriptor
ASCQM Release File Resource after Use in Operation(FunctionProcedureOrMethod, FileResourceOpenStatement, PathToExitWithoutFileResourceClose)

Description
Identify occurrences in application model where:
- the <FunctionProcedureOrMethod> function, procedure, method, ...
- uses the <FileResourceOpenStatement> file resource open statement
- while there exist the <PathToExitWithoutFileResourceClose> path to exit the <FunctionProcedureOrMethod> function, procedure, method, ... without releasing the file resource

The path to exit the function, procedure, method, includes calls to other functions, procedures, methods, ...

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

```
PlatformModel
     ...
    FileResource 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 <FileResourceOpenStatement> file resource open statement
- the <PathToExitWithoutFileResourceClose> path to exit

8.15 ASCQM Implement Virtual Destructor for Classes with Virtual Methods

Descriptor
ASCQM Implement Virtual Destructor for Classes with Virtual Methods(Class, VirtualMethod)

Description
Identify occurrences in application model where:
- the <Class> class
- owns the <VirtualMethod> virtual method
- but lacks a virtual destructor

KDM outline illustration
KDM elements present in the application model

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

```
ClassUnit id="c1"
     ....
      MethodUnit is="m1" methodKind="method" isVirtual="true"

          ...
```

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

```
ClassUnit id="c1"
     ....
    MethodUnit  is="m2"  methodKind="destructor"  isVirtual="true"
        ...
```

What to report
Roles to report are:
- the <Class> class
- the <VirtualMethod> virtual method

8.16 ASCQM Manage Time-Out Mechanisms in Blocking Synchronous Calls

Descriptor
ASCQM Manage Time-Out Mechanisms in Blocking Synchronous Calls(BlockingSynchronousCall, TimeOutOption)

Description
Identify occurrences in application model where:
- the <BlockingSynchronousCall> synchronous call
- doesn't use its <TimeOutOption> time-out option

The list of blocking synchronous primitives is technology, framework, language dependent. For example, in Java: connect(), receive().

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

```
ControlElement id="ce1" name="connect|receive|..." type="ce1_signature"
    Signature id="ce1_signature"
        ...
        ParameterUnit id="pu1" name="timeout|..."
    ...
Value id="v1" attribute="infinite_wait"
...
ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall"
    ...
    Calls "ce1"
    Reads "v1"
```

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

```
...
Value id="v2" attribute="finite_wait"
...
ActionElement id="ae1" kind="Call|PtrCall|MethodCall|VirtualCall"
    ...
    Calls "ce1"
    Reads "v2"
```

What to report
Roles to report:
- the <BlockingSynchronousCall> synchronous call
- the <TimeOutOption> time-out option

8.17 ASCQM Ban Hard-Coded Literals used to Connect to Resource

Descriptor
ASCQM Ban Hard-Coded Literals used to Connect to Resource(InitializationStatement, ResourceAccessStatement)

Description
Identify occurrences in application model where:
- the <InitializationStatement> initialization statement
- initialize a variable used in the <ResourceAccessStatement> resource access statement as parameter to call a resource access primitive

It covers credentials, passwords, encryption keys, tokens, remember-me keys...

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

```
Value id="hcv" name="hcv"
...
StorableUnit|ItemUnit|MemberUnit  id="su1"
...
ActionElement id="ae1" kind="Assign
    Reads "hcv"
    Writes "su1"
...
MarshalledResource|MessagingResource|DataManager|ExecutionResource  id="nwr"
...
ControlElement  id="ce1"
     ...
    ActionELement id="ae2" kind="Platform"
        ManagesResource|ReadsResource|WritesResource "nwr"
...
ActionElement id="ae3" kind="Call|PtrCall|MethodCall|VirtualCall"
    Reads "su1"
     ...
    Calls "ce1"
```

What to report
Roles to report are:
- the <InitializationStatement> initialization statement
- the <ResourceAccessStatement> resource access statement

8.18 ASCQM Ban Unintended Paths

Descriptor
ASCQM Ban Unintended Paths(ArchitectureModel, Relation, Caller, Callee, OriginModule, TargetModule)

Description
Identify occurrences in the application model where:
- the <Relation> call-type, data, use relations
- between the <Caller> caller
- grouped in the <OriginModule> origin layer, component, or subsystem
- and the <Callee> callee
- grouped into the <TargetModule> target layer, component, or subsystem
- as defined in the <ArchitectureModel> architectural blueprint defining layers, components, or subsystems
- where relations from the <OriginModule> layer, component, or subsystem to the <TargetModule> layer, component, or subsystem are not intended

The architectural blueprint defining layers, components, or subsystems is application dependent.

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

```
...
Layer|Component|Subsystem  id="m1"
     ...
    CallableUnit callableKind="regular|external|stored" | MethodUnit id="ce1"
name="..."
        ...
```

```
   ActionElement id="ae1"
      UsesType|Reads|Writes|Creates|Addresses|Calls|Dispatches  "ce2"
...
Layer|Component|Subsystem  id="m2"
     ...
    CallableUnit callableKind="regular|external|stored" | MethodUnit id="ce2"
name="..."
...
```

With "m1" not intended to reference "m2"

What to report

Roles to report are:
- the <ArchitectureModel> architectural blueprint
- the <Relation> relation
- the <Caller> caller
- the <Callee> callee
- the <OriginModule> origin layer, component, or subsystem
- the <TargetModule> target layer, component, or subsystem

8.19 ASCQM Ban While TRUE Loop Without Path To Break

Descriptor

ASCQM Ban While TRUE Loop Without Path To Break(WhileTrueLoop)

Description

Identify occurrences in the application model where:
- the <WhileTrueLoop> "while true" loop
- lacks a control flow to a break statement out of the loop

KDM outline illustration

KDM elements present in the application model

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

```
BooleanType id="booleanType"
Value id="true" name="true" type="booleanType"
ActionElement id="ae1" kind="Compound"
    ActionElement id="ae2" kind="Condition"
        Reads "true"
        TrueFlow "tf1"
        FalseFlow "ff1"
    ActionElement id="tf1" ...
    ...
    Flows "ae2"
ActionElement id="ff1" ...
```

KDM elements absent from the application model

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

```
ActionElement id="ae1" kind="Compound"
    ActionElement id="ae2" kind="Condition"
         ...
         TrueFlow "tf1"
         ...
         ActionElement id="tf1" ...
             Flows "ae3"
         ActionElement id="ae3"
             Flows "e1"
         ActionElement id="e1" kind="Goto"
            Flows "ff1"
     ...
  ActionElement id="ff1" ...
```

What to report
Roles to report:
- the <WhileTrueLoop> "while true" loop

8.20 ASCQM Ban Unmodified Loop Variable Within Loop

Descriptor
ASCQM Ban Unmodified Loop Variable Within Loop(WhileLoop)

Description
Identify occurrences in the application model where:
- the <WhileLoop> while loop
- lacks an update of the condition value within the loop

KDM outline illustration
KDM elements present in the application model

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

```
BooleanType id="booleanType"
StorableUnit id="su1"  type="booleanType"
ActionElement id="ae1" kind="Compound"
     ...
     ActionElement id="ae2" kind="Condition"
         Reads "su1"
         ...
     ...
...
```

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

```
ActionElement id="ae1" kind="Compound"
     ...
    ActionElement id="ae3" kind="Assign|Incr|Decr"
        Writes "su1"
        ...
    ...
```

What to report
Roles to report:
- the <WhileLoop> while loop

8.21 ASCQM Release File Resource after Use in Class

Descriptor
ASCQM Release File Resource after Use in Class(Class, FileResourceOpenStatement)

Description
Identify occurrences in application model where:
- the <Class> class, ...
- uses the <FileResourceOpenStatement> file resource open statement
- without releasing the file resource in any of its methods

The path to exit the function, procedure, method, includes calls to other functions, procedures, methods, ...

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

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

...
CodeModel
     ...
    ClassUnit id="cu1"
         ...
          ActionElement id="ae1" kind="PlatformAction"
          ...
...
```

KDM elements absent from the application model

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

```
ClassUnit id="cu1"
     ...
    ActionElement id="ae2" kind="PlatformAction"
...
```

What to report

Roles to report:
- the <Class> class
- the <FileResourceOpenStatement> file resource open statement

8.22 ASCQM Catch Exceptions

Descriptor

ASCQM Catch Exceptions(Method, Exception, MethodCall)

Description

Identify occurrences in application model where:
- the <Method> method
- declared as throwwing the <Exception> exception
- is called in the <MethodCall> method call
- which doesn't catch exceptions of type <Exception>

KDM outline illustration

KDM elements present in the application model

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

```
...
ClassUnit id="cu1"
...
MethodUnit id="mu1" type="mu1_signature"
    Signature id="mu1_signature"
        ParameterUnit id="pu1" type="cu1" kind="throws"
     ...
...
ActionElement id="ae1" kind="MethodCall"
    Calls "mu1"
...
```

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

```
...
TryUnit id="t1"
     ...
     ActionElement id="ae1" kind="MethodCall"
          Calls "mu1"
     ...
     ExceptionFlow  "c1"
...
```

What to report
Roles to report are:
- the <Method> method
- the <Exception> exception
- the <MethodCall> method call

8.23 ASCQM Ban Empty Exception Block

Descriptor
ASCQM Ban Empty Exception Block(CatchBlock)

Description
Identify occurrences in application model where:
- the <CatchBlock> catch block
- is empty

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

```
...
CatchUnit id="cu1"
     ActionElement id="ae1" kind="Nop"
...
```

What to report
Roles to report are:
- the <CatchBlock> catch block

8.24 ASCQM Ban Incompatible Lock Acquisition Sequences

Descriptor
ASCQM Ban Incompatible Lock Acquisition Sequences(LockAcquisitionSequence, ReverseLockAcquisitionSequence)

Description
Identify occurrences in application model where:
- the <LockAcquisitionSequence> sequence of lock acquisition
- is the reverse of the <ReverseLockAcquisitionSequence> sequence of lock acquisition

The locking mechanism is technology, framework, and language dependent.

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

```
PlatformModel
     DeployedResource  id="dr1"
          ...
          LockResource id="lr1"
          LockResource id="lr2"
          ...
```

```
        PlatformAction id="pa1" kind="lock" implementation="ae1 ae12"
            ManagesResource|ReadsResource|WritesResource "lr1"
        PlatformAction id="pa2" kind="lock" implementation="ae3 ae10"
            ManagesResource|ReadsResource|WritesResource "lr2"
...
CodeModel
     ...
     ActionElement id="ae1" kind="PlatformAction"
         Flows "ae2"
     ActionElement id="ae2" ...
         Flows "ae3"
     ActionElement id="ae3" kind="PlatformAction"
         Flows "ae4"
     ActionElement id="ae4" ...
     ...
     ActionElement id="ae10" kind="PlatformAction"
         Flows "ae11"
     ActionElement id="ae11" ...
         Flows "ae12"
     ActionElement id="ae12" kind="PlatformAction"
         Flows "ae13"
     ActionElement id="ae13"  ...
```

What to report

Roles to report are:
- the <LockAcquisitionSequence> sequence of lock acquisition
- the <ReverseLockAcquisitionSequence> sequence of lock acquisition

8.25 ASCQM Ban Use of Thread Control Primitives with Known Deadlock Issues

Descriptor

ASCQM Ban Use of Thread Control Primitives with Known Deadlock Issues(ThreadControlPrimitiveCall)

Description

Identify occurrences in application model where:
- the <ThreadControlPrimitiveCall> call to a thread control function, procedure, method, ... with known deadlock issues.

The list of primitives is technology, framework, language dependant. For example, in Java: java.lang.Thread.suspend(), java.lang.Thread.resume(), java.lang.ThreadGroup.suspend(), java.lang.ThreadGroup.resume() and dependent methods java.lang.ThreadGroup.allowThreadSuspension().

KDM outline illustration

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

```
ControlElement id="ce1"
name="java.lang.Thread.suspend|java.lang.Thread.resume|..."
    ...
...
ActionElement id="ae3" kind="Call|PtrCall|MethodCall|VirtualCall"
     ...
     Calls "ce1"
```

What to report

Roles to report:
- the <ThreadControlPrimitiveCall> call to a thread control function, procedure, method, ... with known deadlock issues.

8.26 ASCQM Ban Use of Prohibited Low-Level Resource Management Functionality

Descriptor

ASCQM Ban Use of Prohibited Low-Level Resource Management Functionality(ResourceManagementPrimitiveCall, TechnologyStack)

Description

Identify occurrences in application model where:
- the <ResourceManagementPrimitiveCall> low-level resource management primitive call
- which is bypassing the resource management primitives provided by the <TechnologyStack> technology stack

KDM outline illustration

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

```
CodeModel
    Package id="p1"
name="javax.ejb|javax.servlet|javax.jms|javax.faces|javax.xml.bind|javax.activati
on|javax.mail|..."
    ...
    Package id="p2" name="java.sql"
        ClassUnit id="cu2" name="DriverManager"
            MethodUnit id="mu2"
    ...
    CompilationUnit id="cu2"
        Imports "p1"
        Imports "p2"
        ...
        ActionElement id="ae1" kind="MethodCall"
            Calls "mu2"
```

or

```
CodeModel
     Package id="p1" name="javax.servlet"
     ...
     Package id="p2" name="java.net"
      ClassUnit id="cu2" name="Socket|ServerSocket"
          MethodUnit id="mu2"
     ...
     CompilationUnit id="cu2"
         Imports "p1"
         Imports "p2"
         ...
         ActionElement id="ae1" kind="MethodCall"
             Calls "mu2"
```

or

```
CodeModel
    Package id="p1" name="javax.ejb"
    ...
    Package id="p2" name="java.net"
        ClassUnit id="cu2" name="Socket|ServerSocket"
            MethodUnit id="mu2"
    ...
    Package id="p3" name="java.lang"
        ClassUnit id="cu3"name= ClassLoader"
            MethodUnit id="mu3"
    ...
    Package id="p4" name="java.io"
        ClassUnit id="cu4" name="File"
```

```
            MethodUnit id="mu4"
    ...
    Package id="p5" name="java.awt"
        ClassUnit id="cu5"
            MethodUnit id="mu5"
    ...
    CompilationUnit id="cu2"
        Imports "p1"
        Imports "p2"
        ...
        ActionElement id="ae1" kind="MethodCall"
            Calls "mu2|mu3|mu4|mu5"
```

or

```
CodeModel
    Package id="p1" name="javax.ejb"
    ...
    ...
    CompilationUnit id="cu2"
        Imports "p1"
        Imports "p2"
        ...
        ActionElement id="ae1" kind="MethodCall" attribute="synchronized"
            ...
```

What to report

Roles to report:
- the <ResourceManagementPrimitiveCall> low-level resource management primitive call
- the <TechnologyStack> technology stack

8.27 ASCQM Ban Excessive Size of Index on Columns of Large Tables

Descriptor

ASCQM Ban Excessive Size of Index on Columns of Large Tables(Table, TotalSizeOfIndexes, MaxTotalSizeOfIndexes, MinNumberOfRows)

Description

Identify occurrences in application model where:
- the <Table> table
- with <TotalSizeOfIndexes> number of indexes
- which is greater than <MaxTotalSizeOfIndexes>
- and with more than <MinNumberOfRows>

The <MaxTotalSizeOfIndexes> value is a measurement parameter. Its default value is: 30 The <MinNumberOfRows> value is a measurement parameter. Its default value is: 1000000

KDM outline illustration

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

```
DataModel
    RelationalSchema
        RelationalTable id="rt1"
            Index id="i1" implementation="iu1"
            Index id="i2" implementation="iu1 iu2"
            ...
            itemUnit id="iu1" type="dt1"
            itemUnit id="iu2" type="dt2"
...
CodeModel
     DataType id="dt1"
```

```
    DataType id="dt2"
    ...
```

The size of an Index is the size in bytes of the data types of the columns it relies on.

What to report

Roles to report:
- the <Table> table
- the <TotalSizeOfIndexes> value
- the <MaxTotalSizeOfIndexes> value
- the <MinNumberOfRows> value

8.28 ASCQM Ban Excessive Number of Index on Columns of Large Tables

Descriptor

ASCQM Ban Excessive Number of Index on Columns of Large Tables(Table, NumberOfIndexes, MaxNumberOfIndexes, MinNumberOfRows)

Description

Identify occurrences in application model where:
- the <Table> table
- with <NumberOfIndexes> number of indexes
- which is greater than <MaxNumberOfIndexes>
- and with more than <MinNumberOfRows>

The <MaxNumberOfIndexes> value is a measurement parameter. Its default value is: 3
The <MinNumberOfRows> value is a measurement parameter. Its default value is: 1000000

KDM outline illustration

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

```
DataModel
    RelationalSchema
        RelationalTable id="rt1"
            Index id="i1"
            Index id="i2"
            Index id="i3"
            Index id="i4"
            Index id="i5"
            Index id="i6"
  ...
```

What to report

Roles to report:
- the <Table> table
- the <NumberOfIndexes> value
- the <MaxNumberOfIndexes> value
- the <MinNumberOfRows> value

8.29 ASCQM Ban Excessive Complexity of Data Resource Access

Descriptor

ASCQM Ban Excessive Complexity of Data Resource Access(Query, NumberOfTables, MaxNumberOfTables, NumberOfSubqueries, MaxNumberOfSubqueries, MinNumberOfRows)

Description
Identify occurrences in application model where:
- the <Query> query
- with <NumberOfTables> number of tables or views
- which is greater than <MaxNumberOfTables>
- and with <NumberOfSubqueries> number of subqueries
- which is greater than <MaxNumberOfSubqueries>
- with at least one table or view with more than <MinNumberOfRows>

The <MaxNumberOfTables> value is a measurement parameter. Its default value is: 5
The <MaxNumberOfSubqueries> value is a measurement parameter. Its default value is:3
The <MinNumberOfRows> value is a measurement parameter. Its default value is: 1000000

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

```
DataModel
    RelationalSchema
        RelationalTable|RelationalView id="cs1"
        RelationalTable|RelationalView id="cs2"
        RelationalTable|RelationalView id="cs3"
        RelationalTable|RelationalView id="cs4"
        RelationalTable|RelationalView id="cs5"
        RelationalTable|RelationalView id="cs6"
        ...
        DataAction id="da1" kind="Select|Insert|Update|Delete"
            ...
            ReadsColumnSet|WritesColumnSet  "cs1"
            ReadsColumnSet|WritesColumnSet "cs2"
            ReadsColumnSet|WritesColumnSet "cs3"
            ReadsColumnSet|WritesColumnSet "cs4"
            ReadsColumnSet|WritesColumnSet "cs5"
            ReadsColumnSet|WritesColumnSet  "cs6"
            ...
            DataAction id="da2" kind="Select"
                ...
            DataAction id="da3" kind="Select"
                ...
            DataAction id="da4" kind="Select"
                ...
            DataAction id="da5" kind="Select"
                ...
            ...
         ...
     ...
  ...
```

What to report
Roles to report:
- the <Query> query
- the <NumberOfTables> value
- the <MaxNumberOfTables> value
- the <NumberOfSubqueries> value
- the <MaxNumberOfSubqueries> value
- the <MinNumberOfRows> value

8.30 ASCQM Ban Expensive Operations in Loops

Descriptor
ASCQM Ban Expensive Operations in Loops(ResourceConsummingStatement, Loop)

Description
Identify occurrences in application model where:

- the <ResourceConsummingStatement> resource consuming statement
- is used within the <Loop> loop.

KDM outline illustration

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

```
ActionElement id="ae1" kind="New|NewArray"

or

ActionElement id="ae1" kind="SizeOf|InstanceOf|DynCast|TypeCast"

or

ActionElement id="ae1" kind="New|NewArray"

or

PlatformModel
     ...
     MarshalledResource|NamingResource|DataManager  id="pr1"
      ...
     PlatformAction  id="pa1" implementation="ae1"
         ManagesResource|WritesResource|ReadsResource  "pr1"
...
CodeModel
     ...
    ActionElement id="ae1" kind="PlatformAction"
....

with (while loops)

BooleanType id="booleanType"
Value id="true" name="true" type="booleanType"
ActionElement id="ae2" kind="Compound"
    ActionElement id="ae3" kind="Condition"
        Reads "true"
        TrueFlow "tf1"
        FalseFlow "ff1"
    ActionElement id="tf1" ...
     ...
        Flows "ae1"
     ...
    Flows "ae3"
ActionElement id="ff1" ...

or (for loops)

ActionElement id="ae2" kind="compound"
    ActionElement id="ae3" kind="Assign"
        Reads ...
        Writes "LoopVariable"
        Flows "ae4"
    ActionElement id="ae4"
kind="LessThan|LessThanOrEqual|GreaterThan|GreaterThanOrEqual"
        Reads "LoopVariable"
        Reads ...
        TrueFlow "ae5"
        FalseFlow "ae7"
    ActionElement id="ae5" kind=...
    ...
        Flows "ae1"
    ...
   ActionElement id="ae6" kind="Incr|Decr"
```

```
        Addresses "LoopVariable"
        Flows "ae4"
   ActionElement id="ae7" kind="Nop"
...
```

What to report

Roles to report are:

- the <ResourceConsummingStatement> resource consuming statement
- the <Loop> loop.

8.31 ASCQM Limit Number of Aggregated Non-Primitive Data Types

Descriptor

ASCQM Limit Number of Aggregated Non-Primitive Data Types(Class, NumberOfNonPrimitiveMembers, MaxNumberOfNonPrimitiveMembers)

Description

Identify occurrences in application model where :

- the <Class> class
- with <NumberOfNonPrimitiveMembers> number of non-primitive members
- which is greater than <MaxNumberOfNonPrimitiveMembers>

The <MaxNumberOfNonPrimitiveMembers> value is a measurement parameter. Its default value is: 5

KDM outline illustration

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

```
ClassUnit id="cu1"
ClassUnit id="cu2"
ClassUnit id="cu3"
ClassUnit id="cu4"
ClassUnit id="cu5"
ClassUnit id="cu6"
...
ClassUnit id="cu0"
    MemberUnit id="mu1" type="cu1"
    MemberUnit id="mu2" type="cu2"
    MemberUnit id="mu3" type="cu3"
    MemberUnit id="mu4" type="cu4"
    MemberUnit id="mu5" type="cu5"
    MemberUnit id="mu6" type="cu6"
    ...
```

What to report

Roles to report :

- the <Class> class
- the <NumberOfNonPrimitiveMembers> value
- the <MaxNumberOfNonPrimitiveMembers> value

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

Descriptor

ASCQM Ban Excessive Number of Data Resource Access from non-stored SQL Procedure(Function, NumberOfDataAccess, MaxNumberOfDataAccess)

Description

Identify occurrences in application model where:
- the <Function> SQL function is not a stored procedure
- with <NumberOfDataAccess> accesses to data resources
- which is greater than <MaxNumberOfDataAccess>

The <MaxNumberOfDataAccess> value is a measurement parameter. Its default value is: 5

KDM outline illustration

KDM elements present in the application model

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

```
DataModel
    RelationSchema  id="rs1"
        ...
        CallableUnit id="cu1"
            ...
            ActionElement id="da1" kind="Select|Insert|Update|Delete"
            ...
            ActionElement id="da2" kind="Select|Insert|Update|Delete"
            ...
            ActionElement id="da3" kind="Select|Insert|Update|Delete"
            ...
            ActionElement id="da4" kind="Select|Insert|Update|Delete"
            ...
            ActionElement id="da5" kind="Select|Insert|Update|Delete"
            ...
            ActionElement id="da6" kind="Select|Insert|Update|Delete"
            ...
```

KDM elements absent from the application model

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

```
DataModel
    RelationSchema  id="rs1"
        ...

        CallableUnit id="cu1" kind="stored"
            ...
```

What to report

Roles to report:
- the <Function> function
- the <NumberOfDataAccess> value
- the <MaxNumberOfDataAccess> value

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

Descriptor

ASCQM Ban Excessive Number of Data Resource Access from non-SQL Code(FunctionProcedureOrMethod, NumberOfDataAccess, MaxNumberOfDataAccess)

Description

Identify occurrences in application model where:
- the <FunctionProcedureOrMethod> function, procedure, method, ...
- with <NumberOfDataAccess> accesses to data resources
- which is greater than <MaxNumberOfDataAccess>

The <MaxNumberOfDataAccess> value is a measurement parameter. Its default value is: 2

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

```
DataModel
    RelationSchema  id="rs1"
        ...
        ActionElement id="da1" kind="Select|Insert|Update|Delete"
implementation="i1"
        ActionElement id="da2" kind="Select|Insert|Update|Delete"
implementation="i2"
        ActionElement id="da3" kind="Select|Insert|Update|Delete"
implementation="i3"
...
CodeModel
     ...
    CallableUnit id="cu1" | MethodUnit id="mu1"
        ...
        ActionElement  id="i1"
        ...
        ActionElement  id="i2"
        ...
        ActionElement  id="i3"
        ...
...
```

What to report
Roles to report:
- the <FunctionProcedureOrMethod> function, procedure, method, ...
- the <NumberOfDataAccess> value
- the <MaxNumberOfDataAccess> value

8.34 ASCQM Ban Incremental Creation of Immutable Data

Descriptor
ASCQM Ban Incrementral Creation of Immutable Data(StringConcatenationStatement)

Description
Identify occurrences in the application model where:
- a text variable is incrementaly updated in the <StringConcatenationStatement> string concatenation statement

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

```
...
StringType id="st1"
StorableUnit id="su1" type="st1"
...
ActionElement id="ae1" kind="Append"
    Reads "su1"
    Writes "su1"
    ...
...
```

What to report
Roles to report are:
- the <StringConcatenationStatement> string concatenation statement

8.35 ASCQM Ban Unboxing in Loops

Descriptor

Ban Unboxing in Loops(Unboxing, Loop)

Description

Identify occurrences in application model where
- the <Unboxing> unboxing statement
- is used within the <Loop> loop.

KDM outline illustration

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

```
IntegerType|DecimalType|BooleanType|BitType|CharType|

...
id="dt1"
...
StorableUnit|ItemUnit|MemberUnit|Value
      id="de1"
      type="dt1"
      ...
ClassUnit| ...
      id="dt2"
MemberUnit
      id="fu1"
            type="dt1" ...
BooleanType id="booleanType"
      Value id="true"
      name="true"
      type="booleanType"
ActionElement
      id="ae2"
      kind="Compound"
ActionElement
      id="ae3"
      kind="Condition"
Reads "true"
      TrueFlow "tf1"
      FalseFlow "ff1"
ActionElement
      id="tf1" ...
      ...
StorableUnit|ItemUnit|MemberUnit
      id="de2"
      type="dt2"
      ...
ActionElement
      id="ae1"
      kind="Assign"
Writes "de1"
Reads "de2"
   ...
Flows "ae3"
ActionElement
      id="ff1" ...

or

IntegerType|DecimalType|BooleanType|BitType|CharType| ...
      id="dt1"
...
StorableUnit|ItemUnit|MemberUnit|Value
      id="de1"
      type="dt1"
```

```
...
ClassUnit|...
      id="dt2"
MemberUnit
      id="fu1"
      type="dt1"
     ...
ActionElement
      id="ae2"
      kind="compound"
ActionElement
      id="ae3"
      kind="Assign"
Reads ...
Writes "LoopVariable"
Flows "ae4"
ActionElement
      id="ae4"
      kind="LessThan|LessThanOrEqual|GreaterThan|GreaterThanOrEqual"
Reads "LoopVariable"
Reads ...
      TrueFlow "ae5"
      FalseFlow "ae7"
ActionElement
      id="ae5"
      kind=...
...
StorableUnit|ItemUnit|MemberUnit
      id="de2"
      type="dt2"
...
ActionElement
      id="ae1"
      kind="Assign"
Writes "de1"
Reads "de2"
...
ActionElement
      id="ae7"
      kind="Nop"
...
```

What to report

Roles to report are
- the <Unboxing> unboxing statement
- the <Loop> loop.

8.36 ASCQM Ban Autoboxing in Loops

Descriptor

ASCQM Ban Autoboxing in Loops(Autoboxing, Loop)

Description

Identify occurrences in application model where - the <Autoboxing> autoboxing statement - is used within the <Loop> loop.

KDM outline illustration

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

```
IntegerType|DecimalType|BooleanType|BitType|CharType| ...
      id="dt1"
...
```

```
ClassUnit|...
      id="dt2"
MemberUnit
      id="fu1"
      type="dt1"
...
StorableUnit|ItemUnit|MemberUnit
      id="de2"
      type="dt2"
...
BooleanType
      id="booleanType"
Value
      id="true"
      name="true"
      type="booleanType"
ActionElement
      id="ae2"
      kind="Compound"
ActionElement
      id="ae3"
      kind="Condition"
Reads "true"
      TrueFlow "tf1"
      FalseFlow "ff1"
ActionElement
      id="tf1" ...
...
StorableUnit|ItemUnit|MemberUnit|Value
      id="de1"
      type="dt1"
...
ActionElement
      id="ae1"
      kind="Assign"
Writes "de2"
Reads "de1"    ...
Flows "ae3"
ActionElement
      id="ff1"
...

or

IntegerType|DecimalType|BooleanType|BitType|CharType|...

...
ClassUnit|...
      id="dt2"
MemberUnit
       id="fu1"
       type="dt1"
        ...
ActionElement
      id="ae2"
      kind="compound"


ActionElement
      id="ae3"
      kind="Assign"
Reads ...
Writes "LoopVariable"
Flows "ae4"
ActionElement
```

```
      id="ae4"
      kind="LessThan|LessThanOrEqual|GreaterThan|GreaterThanOrEqual"
Reads "LoopVariable"
Reads ...
TrueFlow "ae5"
FalseFlow "ae7"
ActionElement
      id="ae5"
      kind=...
...
StorableUnit|ItemUnit|MemberUnit|Value
      id="de1"
      type="dt1"
      ...
ActionElement
      id="ae1"
      kind="Assign"
Writes "de2"
Reads "de1"
...
ActionElement
      id="ae7"
      kind="Nop"
      ...
```

What to report

Roles to report are
- the <Autoboxing> autoboxing statement
- the <Loop> loop.

8.37 ASCQM Implement Index Required by Query on Large Tables

Descriptor

ASCQM Implement Index Required by Query on Large Tables(Query, Table, Column, MinNumberOfRows)

Description

Identify occurrences in application model where:
- the <Query> query
- queries the <Table> table
- using the <Column> column(s)
- where the <Table> table has more than <MinNumberOfRows>
- but lacks a proper index

The <MinNumberOfRows> value is a measurement parameter. Its default value is: 1000000

KDM outline illustration

KDM elements present in the application model

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

```
DataModel
    RelationalSchema
        RelationalTable id="rt1"
            itemUnit id="iu1"


         ...
         DataAction id="da1" kind="Select|Insert|Update|Delete"
             ...
            Reads "iu1"
            ...
        ...
     ...
...
```

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

```
DataModel
    RelationalSchema
        RelationalTable id="rt1"
            Index id="i1" implementation="iu1"
            itemUnit id="iu1"
...
```

What to report
Roles to report:
- the <Query> query
- the <Table> table
- the <Column> column (list)
- the <MinNumberOfRows> value

8.38 ASCQM Release Memory after Use with Correct Reference

Descriptor
ASCQM Release Memory after Use with Correct Reference(MemoryAllocationStatement, AllocationReference, MemoryReleaseStatement, ReleaseReference)

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

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

ControlElement
      id="ce1"
      name="malloc|calloc|
     ...
|New|NewArray|..."
...
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" ...
StorableUnit
      id="su2"
      type="pt1"
     ...
ActionElement
      id="ae2"
      type="add"
Reads "su1"
...
Writes "su2" ...
ControlElement
       id="ce2"
name="free|...
|delete|delete[]|..."
...
ActionElement
      id="ae3"
      kind="Call"
Addresses "su2"
Calls "ce2"
```

What to report

Roles to report are

- the <MemoryAllocationStatement> allocation statement
- the <AllocationReference> reference - the <MemoryReleaseStatement> release statement
- the <ReleaseReference> reference

This page intentionally left blank.

9 Calculation of ASCRSM 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 Automated Source Code Quality Measure (ASCRSM). Software quality characteristic measures have frequently been scored at the software component level and then aggregated to develop an overall score for an application. However, scoring at the software 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, the ASCRSM score is computed as the sum of its quality measure elements counted across an entire application.

The calculation of an ASCRSM score progresses as follows:

- One or more Detection Pattern Scores are calculated for each weakness as the total occurrences of each Detection Pattern associated with the weakness.
- Weakness Scores are calculated for each weakness as the total sum of Detection Pattern Scores associated with the Weakness.
- ASCRSM is calculated as the sum of its Weakness Scores.

That is,

Detection Pattern Score $_{x,y} = \sum_{y=1}^{n}$ Occurrences $_{y}$

where x = a specific CWE weakness (e.g., CWE-248, CWE-252, etc.)

y = the n^{th} detection pattern associated with weakness x

Weakness Score $_{x} = \sum_{y=1}^{n}$ Detection Pattern Score $_{y}$

where x = a specific CWE weakness (CWE-248, CWE-252, etc.)

y = a Detection Pattern associated with Weakness x

ASCRSM = $\sum_{x=1}^{n}$ Weakness Score $_{x}$

where x = a specific CWE weakness (e.g., CWE-248, CWE-252, etc.)

Furthermore, total counts of occurrences for each Detection Pattern can be calculated as:

Total Detection Pattern Score $_{y} = \sum_{x=1}^{n}$ Detection Pattern Score $_{x,y}$

where x = a specific CWE weakness (e.g., CWE-248, CWE-252,, etc.)

y = a specific Detection Pattern

9.2 Functional Destiny of Weanessess (Informative)

In order to compare quality results among different applications, the Automated Source Code Resource Sustainability 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 resource sustainability of applications. OMG's Automated Function Points (AFP) measure (ISO, 2019) 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 Resource Sustainability Measure. Thus, the functional density of Resource Sustainability weaknesses is a simple division expressed as follows.

ASCRSM-density = ASCRSM / AFP

10 Alternative Weighted Measures and Uses (Informative)

There are many additional weighting schemes that can be applied to the Automated Source Code Resource Sustainability Measure or to the quality measure elements that composing it. 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 3: Informative Weighting Schemes for Security Measurement

Weighting scheme	Potential uses
Weight each Total Detection Pattern score by the risk it presents	Identifying training needs for avoiding patterns underlying risky weaknesses
Weight each weakness 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 resource sustainability weaknesses	Prioritizing modules or application components for corrective maintenance or replacement

This page intentionally left blank.

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 . Milford, 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 (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.

International Organization for Standards (2019*). ISO/IEC 19515:2019, Automated Function Points. Information technology -- Object Management Group Automated Function Points (AFP), 1.0*. Geneva, Switzerland. Also, Object Management Group (2014). *Automated Function Points. formal 2014-01-03* http://www.omg.org/spec/AFP/ . Needham, MA: Object Management Group.

International Telecommunications Union (2012). *ITU-T X.1524 – Series X: Data Networks, Open System Communications and Security – Cybersecurity information exchange – Vulnerability/state exchange – Common weakness enumeration*. Geneva:, Switzerland.

Martin, R.A. & Barnum, S. (2006). *Status update: The Common Weakness Enumeration*. NIST Static Analysis Summit, Gaithersburg, MD, June 29, 2006.

This page intentionally left blank.

Annex A
Consortium for IT Software Quality (CISQ)

(informative)

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

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

Companies interested in joining CISQ held executive forums in Frankfurt, Germany; Arlington, VA; and Bangalore, India to set strategy and direction for the consortium. In these forums four quality characteristics were selected as the most important targets for automation—reliability, security, performance efficiency, and maintainability. These attributes cover four of the eight quality characteristics described in ISO/IEC 25010.

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.

This page intentionally left blank.

Annex B
Common Weakness Enumeration (CWE)

(informative)

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 the ITU-T X.1524 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.