

DRAFT INTERNATIONAL STANDARD

ISO/IEC DIS 5055

ISO/IEC JTC 1

Secretariat: ANSI

Voting begins on:
2020-06-04

Voting terminates on:
2020-08-27

Information technology — Software measurement — Software quality measurement — Automated source code quality measures

ICS: 35.080

iTeh STANDARD PREVIEW (standards.iteh.ai)

[ISO/IEC DIS 5055](https://standards.iteh.ai/catalog/standards/sist/17e93555-8fbc-47c2-8495-32371a9a0d7c/iso-iec-dis-5055)

<https://standards.iteh.ai/catalog/standards/sist/17e93555-8fbc-47c2-8495-32371a9a0d7c/iso-iec-dis-5055>

THIS DOCUMENT IS A DRAFT CIRCULATED FOR COMMENT AND APPROVAL. IT IS THEREFORE SUBJECT TO CHANGE AND MAY NOT BE REFERRED TO AS AN INTERNATIONAL STANDARD UNTIL PUBLISHED AS SUCH.

IN ADDITION TO THEIR EVALUATION AS BEING ACCEPTABLE FOR INDUSTRIAL, TECHNOLOGICAL, COMMERCIAL AND USER PURPOSES, DRAFT INTERNATIONAL STANDARDS MAY ON OCCASION HAVE TO BE CONSIDERED IN THE LIGHT OF THEIR POTENTIAL TO BECOME STANDARDS TO WHICH REFERENCE MAY BE MADE IN NATIONAL REGULATIONS.

RECIPIENTS OF THIS DRAFT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

This document is circulated as received from the committee secretariat.

FAST TRACK PROCEDURE



Reference number
ISO/IEC DIS 5055:2020(E)

© ISO/IEC 2020

iTeh STANDARD PREVIEW (standards.iteh.ai)

ISO/IEC DIS 5055

<https://standards.iteh.ai/catalog/standards/sist/17e93555-8fbc-47c2-8495-32371a9a0d7c/iso-iec-dis-5055>



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2020

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Table of Contents

Preface	xi
1 Scope	1
1.1 Purpose	1
1.2 Overview of Structural Quality Measurement in Software	1
2 Conformance	2
3 Normative References.....	2
4 Terms and Definitions	2
5 Symbols (and Abbreviated Terms).....	4
6 Additional Information (Informative)	5
6.1 Software Product Inputs	5
6.2 Automated Source Code Quality Measure Elements.....	5
6.3 Automated Source Code Maintainability Measure Element Descriptions	5
6.4 Automated Source Code Performance Efficiency Measure Element Descriptions.....	8
6.5 Automated Source Code Reliability Measure Element Descriptions	10
6.6 Automated Source Code Security Measure Element Descriptions.....	17
6.7 Introduction to the Specification of Quality Measure Elements.....	25
6.8 Knowledge Discovery Metamodel (KDM).....	25
6.9 Software Patterns Metamodel Standard (SPMS)	29
6.10 Reading guide.....	30
7 List of ASCQM Weaknesses (Normative)	32
7.1 Weakness Category Maintainability	32
7.1.1 CWE-407 Algorithmic Complexity	32
7.1.2 CWE-478 Missing Default Case in Switch Statement.....	32
7.1.3 Weakness CWE-480 Use of Incorrect Operator	32
7.1.4 CWE-484 Omitted Break Statement in Switch	33
7.1.5 CWE-561 Dead Code.....	33
7.1.6 CWE-570 Expression is Always False.....	33
7.1.7 CWE-571 Expression is Always True	33
7.1.8 CWE-783 Operator Precedence Logic Error	34
7.1.9 CWE-1075 Unconditional Control Flow Transfer Outside of Switch Block	34
7.1.10 CWE-1121 Excessive McCabe Cyclomatic Complexity Value	34
7.1.11 CWE-1054 Invocation of a Control Element at an Unnecessarily Deep Horizontal Layer (Layer- skipping Call)	35
7.1.12 CWE-1064 Invokable Control Element with Signature Containing an Excessive Number of Parameters	35
7.1.13 CWE-1084 Invokable Control Element with Excessive File or Data Access Operations	36
7.1.14 CWE-1051 Initialization with Hard-Coded Network Resource Configuration Data	36
7.1.15 CWE-1090 Method Containing Access of a Member Element from Another Class.....	36
7.1.16 CWE-1074 Class with Excessively Deep Inheritance.....	37
7.1.17 CWE-1086 Class with Excessive Number of Child Classes.....	37
7.1.18 CWE-1041 Use of Redundant Code (Copy-Paste)	37
7.1.19 CWE-1055 Multiple Inheritance from Concrete Classes	38
7.1.20 CWE-1045 Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor	38
7.1.21 CWE-1052 Excessive Use of Hard-Coded Literals in Initialization	39
7.1.22 CWE-1048 Invokable Control Element with Large Number of Outward Calls (Excessive Coupling or Fan-out)	39
7.1.23 CWE-1095 Loop Condition Value Update within the Loop.....	39
7.1.24 CWE-1085 Invokable Control Element with Excessive Volume of Commented-out Code.....	40
7.1.25 CWE-1047 Modules with Circular Dependencies.....	40
7.1.26 CWE-1080 Source Code File with Excessive Number of Lines of Code	41
7.1.27 CWE-1062 Parent Class Element with References to Child Class	41
7.1.28 CWE-1087 Class with Virtual Method without a Virtual Destructor	41
7.1.29 CWE-1079 Parent Class without Virtual Destructor Method	42

7.1.30 Maintainability detection patterns	42
7.2 Weakness Category Performance Efficiency	44
7.2.1 CWE-401 Improper Release of Memory Before Removing Last Reference ('Memory Leak')	44
7.2.2 Weakness CWE-404 Improper Resource Shutdown or Release.....	44
7.2.3 CWE-424 Improper Protection of Alternate Path	45
7.2.4 CWE-772 Missing Release of Resource after Effective Lifetime	45
7.2.5 CWE-775 Missing Release of File Descriptor or Handle after Effective Lifetime	45
7.2.6 CWE-1073 Non-SQL Invokable Control Element with Excessive Number of Data Resource Access.....	46
7.2.7 CWE-1057 Data Access Operations Outside of Designated Data Manager Component	46
7.2.8 CWE-1043 Storable and Member Data Element Excessive Number of Aggregated Storable and Member Data Elements.....	47
7.2.9 CWE-1072 Data Resource Access without use of Connection Pooling	47
7.2.10 CWE-1060 Excessive Number of Inefficient Server-Side Data Accesses	47
7.2.11 CWE-1091 Use of Object without Invoking Destructor Method	48
7.2.12 CWE-1046 Creation of Immutable Text Using String Concatenation	48
7.2.13 CWE-1042 Static Member Data Element outside of a Singleton Class Element.....	48
7.2.14 CWE-1049 Excessive Data Query Operations in a Large Data Table	49
7.2.15 CWE-1067 Excessive Execution of Sequential Searches of Data Resource	49
7.2.16 CWE-1089 Large Data Table with Excessive Number of Indices	50
7.2.17 CWE-1094 Excessive Index Range Scan for a Data Resource	50
7.2.18 CWE-1050 Excessive Platform Resource Consumption within a Loop	50
7.2.19 CWE-1060 Excessive Number of Inefficient Server-Side Data Accesses	51
7.2.20 Performance Efficiency Detection Patterns	51
7.3 Weakness Category Reliability	52
7.3.1 CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer	52
7.3.2 CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	52
7.3.3 CWE-123 Write-what-where Condition	53
7.3.4 CWE-125 Out-of-bounds Read	53
7.3.5 CWE-130 Improper Handling of Length Parameter Inconsistency	54
7.3.6 CWE-131 Incorrect Calculation of Buffer Size	54
7.3.7 CWE-170 Improper Null Termination	54
7.3.8 CWE-194 Unexpected Sign Extension	55
7.3.9 CWE-195 Signed to Unsigned Conversion Error	55
7.3.10 CWE-196 Unsigned to Signed Conversion Error.....	55
7.3.11 CWE-197 Numeric Truncation Error	56
7.3.12 CWE-248 Uncaught Exception	56
7.3.13 CWE-252 Unchecked Return Value.....	57
7.3.14 CWE-366 Race Condition within a Thread	57
7.3.15 CWE-369 Divide By Zero.....	57
7.3.16 CWE-390 Detection of Error Condition Without Action	58
7.3.17 CWE-391 Unchecked Error Condition.....	58
7.3.18 CWE-392 Missing Report of Error Condition	58
7.3.19 CWE-394 Unexpected Status Code or Return Value.....	59
7.3.20 CWE-401 Improper Release of Memory Before Removing Last Reference ('Memory Leak')	59
7.3.21 CWE-404 Improper Resource Shutdown or Release.....	60
7.3.22 CWE-415 Double Free.....	60
7.3.23 CWE-416 Use After Free.....	61
7.3.24 CWE-424 Improper Protection of Alternate Path	61
7.3.25 CWE-456 Missing Initialization of a Variable	61
7.3.26 CWE-459 Incomplete Cleanup	62
7.3.27 CWE-476 NULL Pointer Dereference	62
7.3.28 CWE-480 Use of Incorrect Operator	62
7.3.29 CWE-484 Omitted Break Statement in Switch	63
7.3.30 CWE-543 Use of Singleton Pattern Without Synchronization in a Multithreaded Context	63
7.3.31 CWE-562 Return of Stack Variable Address	63
7.3.32 CWE-567 Unsynchronized Access to Shared Data in a Multithreaded Context.....	64
7.3.33 CWE-595 Comparison of Object References Instead of Object Contents.....	64
7.3.34 CWE-597 Use of Wrong Operator in String Comparison	64
7.3.35 CWE-662 Improper Synchronization	65

7.3.36	CWE-667 Improper Locking	66
7.3.37	CWE-672 Operation on a Resource after Expiration or Release	66
7.3.38	CWE-681 Incorrect Conversion between Numeric Types	67
7.3.39	CWE-682 Incorrect Calculation	67
7.3.40	CWE-703 Improper Check or Handling of Exceptional Conditions.....	67
7.3.41	CWE-704 Incorrect Type Conversion or Cast	68
7.3.42	CWE-758 Reliance on Undefined, Unspecified, or Implementation-Defined Behavior.....	68
7.3.43	CWE-764 Multiple Locks of a Critical Resource	69
7.3.44	CWE-772 Missing Release of Resource after Effective Lifetime	69
7.3.45	CWE-775 Missing Release of File Descriptor or Handle after Effective Lifetime	70
7.3.46	CWE-786 Access of Memory Location Before Start of Buffer	70
7.3.47	CWE-787 Out-of-bounds Write.....	70
7.3.48	CWE-788 Access of Memory Location After End of Buffer	71
7.3.49	CWE-805 Buffer Access with Incorrect Length Value.....	71
7.3.50	CWE-820 Missing Synchronization.....	72
7.3.51	CWE-821 Incorrect Synchronization.....	72
7.3.52	CWE-822 Untrusted Pointer Dereference	72
7.3.53	CWE-823 Use of Out-of-range Pointer Offset.....	73
7.3.54	CWE-824 Access of Uninitialized Pointer.....	73
7.3.55	CWE-825 Expired Pointer Dereference.....	74
7.3.56	CWE-833 Deadlock.....	74
7.3.57	CWE-835 Loop with Unreachable Exit Condition ('Infinite Loop')	74
7.3.58	CWE-908 Use of Uninitialized Resource.....	75
7.3.59	CWE-1083 Data Access from Outside Designated Data Manager Component	75
7.3.60	CWE-1058 Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element	75
7.3.61	CWE-1096 Singleton Class Instance Creation without Proper Locking or Synchronization	76
7.3.62	CWE-1087 Class with Virtual Method without a Virtual Destructor	76
7.3.63	CWE-1079 Parent Class without Virtual Destructor Method	77
7.3.64	CWE-1045 Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor	77
7.3.65	CWE-1051 Initialization with Hard-Coded Network Resource Configuration Data	77
7.3.66	CWE-1088 Synchronous Access of Remote Resource without Timeout	78
7.3.67	CWE-1066 Missing Serialization Control Element.....	78
7.3.68	CWE-1070 Serializable Storable Data Element with non-Serializable Item Elements	79
7.3.69	CWE-1097 Persistent Storable Data Element without Associated Comparison Control Element	79
7.3.70	CWE-1098 Data Element containing Pointer Item without Proper Copy Control Element.....	79
7.3.71	CWE-1082 Class Instance Self Destruction Control Element.....	80
7.3.72	CWE-1077 Floating Point Comparison with Incorrect Operator	80
7.3.73	CWE-665 Improper Initialization	81
7.3.74	CWE-457 Use of Uninitialized Variable	81
7.3.75	Reliability Detection Patterns	81
7.4	Weakness Category Security	82
7.4.1	Improper Restriction of Operations within the Bounds of a Memory Buffer	82
7.4.2	CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	83
7.4.3	CWE-123 Write-what-where Condition	84
7.4.4	CWE-125 Out-of-bounds Read.....	84
7.4.5	CWE-129 Improper Validation of Array Index.....	84
7.4.6	CWE-130 Improper Handling of Length Parameter Inconsistency.....	85
7.4.7	CWE-131 Incorrect Calculation of Buffer Size.....	85
7.4.8	CWE-134 Use of Externally-Controlled Format String	85
7.4.9	CWE-194 Unexpected Sign Extension	86
7.4.10	CWE-195 Signed to Unsigned Conversion Error.....	86
7.4.11	CWE-196 Unsigned to Signed Conversion Error.....	87
7.4.12	CWE-197 Numeric Truncation Error.....	87
7.4.13	CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	87
7.4.14	CWE-23 Relative Path Traversal	88
7.4.15	CWE-252 Unchecked Return Value.....	88
7.4.16	CWE-259 Use of Hard-coded Password	88
7.4.17	CWE-321 Use of Hard-coded Cryptographic Key.....	89

7.4.18	CWE-36 Absolute Path Traversal	89
7.4.19	CWE-366 Race Condition within a Thread	90
7.4.20	CWE-369 Divide by Zero	90
7.4.21	CWE-401 Improper Release of Memory Before Removing Last Reference ('Memory Leak')	90
7.4.22	CWE-404 Improper Resource Shutdown or Release	91
7.4.23	CWE-424 Improper Protection of Alternate Path	91
7.4.24	CWE-434 Unrestricted Upload of File with Dangerous Type	92
7.4.25	CWE-456 Missing Initialization of a Variable	92
7.4.26	CWE-457 Use of Uninitialized Variable	92
7.4.27	CWE-477 Use of Obsolete Function	93
7.4.28	CWE-480 Use of Incorrect Operator	93
7.4.29	CWE-502 Deserialization of Untrusted Data	93
7.4.30	CWE-543 Use of Singleton Pattern Without Synchronization in a Multithreaded Context	94
7.4.31	CWE-564 SQL Injection: Hibernate	94
7.4.32	CWE-567 Unsynchronized Access to Shared Data in a Multithreaded Context	95
7.4.33	CWE-570 Expression is Always False	95
7.4.34	CWE-571 Expression is Always True	95
7.4.35	CWE-606 Unchecked Input for Loop Condition	96
7.4.36	CWE-643 Improper Neutralization of Data within XPath Expressions ('XPath Injection')	96
7.4.37	CWE-652 Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')	96
7.4.38	CWE-662 Improper Synchronization	97
7.4.39	CWE-665 Improper Initialization	97
7.4.40	CWE-667 Improper Locking	98
7.4.41	CWE-672 Operation on a Resource after Expiration or Release	98
7.4.42	CWE-681 Incorrect Conversion between Numeric Types	99
7.4.43	CWE-682 Incorrect Calculation	99
7.4.44	CWE-732 Incorrect Permission Assignment for Critical Resource	100
7.4.45	CWE-77 Improper Neutralization of Special Elements used in a Command ('Command Injection')	100
7.4.46	CWE-772 Missing Release of Resource after Effective Lifetime	100
7.4.47	CWE-775 Missing Release of File Descriptor or Handle after Effective Lifetime	101
7.4.48	CWE-778 Insufficient Logging	101
7.4.49	CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	102
7.4.50	CWE-783 Operator Precedence Logic Error	102
7.4.51	CWE-786 Access of Memory Location Before Start of Buffer	102
7.4.52	CWE-787 Out-of-bounds Write	103
7.4.53	CWE-788 Access of Memory Location After End of Buffer	103
7.4.54	CWE-789 Uncontrolled Memory Allocation	103
7.4.55	CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	104
7.4.56	CWE-798 Use of Hard-coded Credentials	104
7.4.57	CWE-805 Buffer Access with Incorrect Length Value	104
7.4.58	CWE-820 Missing Synchronization	105
7.4.59	CWE-821 Incorrect Synchronization	105
7.4.60	CWE-822 Untrusted Pointer Dereference	106
7.4.61	CWE-823 Use of Out-of-range Pointer Offset	106
7.4.62	CWE-824 Access of Uninitialized Pointer	106
7.4.63	CWE-825 Expired Pointer Dereference	107
7.4.64	CWE-835 Loop with Unreachable Exit Condition ('Infinite Loop')	107
7.4.65	CWE-88 Argument Injection or Modification	107
7.4.66	CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	108
7.4.67	CWE-90 Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	108
7.4.68	CWE-91 XML Injection (aka Blind XPath Injection)	109
7.4.69	CWE-99 Improper Control of Resource Identifiers ('Resource Injection')	109
7.4.70	CWE-611 Improper Restriction of XML External Entity Reference ('XXE')	109
7.4.71	CWE-1057 Data Access Control Element from Outside Designated Data Manager Component	110
7.4.72	CWE-415 Double Free	110
7.4.73	CWE-416 Use After Free	110
7.4.74	Security Detection Patterns	111
8	ASCQM Weakness Detection Patterns (Normative)	113

8.1	Specification of Detection Patterns.....	113
8.2	Detection Patterns.....	113
8.2.1	ASCQM Check Index of Array Access.....	113
8.2.2	ASCQM Check Input of Memory Manipulation Primitives.....	114
8.2.3	ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities.....	115
8.2.4	ASCQM Check Input of String Manipulation Primitives with Boundary Checking Capabilities.....	116
8.2.5	ASCQM Ban Use of Expired Pointer.....	117
8.2.6	ASCQM Ban Input Acquisition Primitives without Boundary Checking Capabilities.....	118
8.2.7	ASCQM Check Offset used in Pointer Arithmetic.....	118
8.2.8	ASCQM Sanitize User Input used as Pointer.....	119
8.2.9	ASCQM Initialize Pointers before Use.....	120
8.2.10	ASCQM Check NULL Pointer Value before Use.....	121
8.2.11	ASCQM Ban Use of Expired Resource.....	122
8.2.12	ASCQM Ban Double Release of Resource.....	123
8.2.13	ASCQM Implement Copy Constructor for Class With Pointer Resource.....	124
8.2.14	ASCQM Ban Free Operation on Pointer Received as Parameter.....	124
8.2.15	ASCQM Ban Delete of VOID Pointer.....	125
8.2.16	ASCQM Ban Variable Increment or Decrement Operation in Operations using the Same Variable.....	126
8.2.17	ASCQM Ban Reading and Writing the Same Variable Used as Assignment Value.....	127
8.2.18	ASCQM Handle Return Value of Resource Operations.....	127
8.2.19	ASCQM Ban Incorrect Numeric Conversion of Return Value.....	129
8.2.20	ASCQM Handle Return Value of Must Check Operations.....	130
8.2.21	ASCQM Check Return Value of Resource Operations Immediately.....	131
8.2.22	ASCQM Ban Useless Handling of Exceptions.....	132
8.2.23	ASCQM Ban Incorrect Object Comparison.....	132
8.2.24	ASCQM Ban Assignment Operation Inside Logic Blocks.....	133
8.2.25	ASCQM Ban Comparison Expression Outside Logic Blocks.....	134
8.2.26	ASCQM Ban Incorrect String Comparison.....	134
8.2.27	ASCQM Ban Logical Operation with a Constant Operand.....	135
8.2.28	ASCQM Implement Correct Object Comparison Operations.....	135
8.2.29	ASCQM Ban Comma Operator from Delete Statement.....	136
8.2.30	ASCQM Release in Destructor Memory Allocated in Constructor.....	137
8.2.31	ASCQM Release Memory after Use with Correct Operation.....	138
8.2.32	ASCQM Implement Required Operations for Manual Resource Management.....	139
8.2.33	ASCQM Release Platform Resource after Use.....	140
8.2.34	ASCQM Release Memory After Use.....	141
8.2.35	ASCQM Implement Virtual Destructor for Classes Derived from Class with Virtual Destructor.....	142
8.2.36	ASCQM Implement Virtual Destructor for Parent Classes.....	143
8.2.37	ASCQM Release File Resource after Use in Operation.....	144
8.2.38	ASCQM Implement Virtual Destructor for Classes with Virtual Methods.....	145
8.2.39	ASCQM Ban Self Destruction.....	145
8.2.40	ASCQM Manage Time-Out Mechanisms in Blocking Synchronous Calls.....	146
8.2.41	ASCQM Ban Non-Final Static Data in Multi-Threaded Context.....	147
8.2.42	ASCQM Ban Non-Serializable Elements in Serializable Objects.....	147
8.2.43	ASCQM Ban Hard-Coded Literals used to Connect to Resource.....	149
8.2.44	ASCQM Ban Unintended Paths.....	149
8.2.45	ASCQM Ban Incorrect Float Number Comparison.....	150
8.2.46	ASCQM Singleton Creation without Proper Locking in Multi-Threaded Context.....	151
8.2.47	ASCQM Ban Incorrect Numeric Implicit Conversion.....	152
8.2.48	ASCQM Data Read and Write without Proper Locking in Multi-Threaded Context.....	153
8.2.49	ASCQM Ban Incorrect Synchronization Mechanisms.....	155
8.2.50	ASCQM Ban Resource Access without Proper Locking in Multi-Threaded Context.....	155
8.2.51	ASCQM Ban Incorrect Type Conversion.....	157
8.2.52	ASCQM Ban Return of Local Variable Address.....	157
8.2.53	ASCQM Ban Storage of Local Variable Address in Global Variable.....	158
8.2.54	ASCQM Ban While TRUE Loop Without Path To Break.....	159
8.2.55	ASCQM Ban Unmodified Loop Variable Within Loop.....	160
8.2.56	ASCQM Check and Handle ZERO Value before Use as Divisor.....	161
8.2.57	ASCQM Ban Creation of Lock On Private Non-Static Object to Access Private Static Data.....	161

8.2.58	ASCQM Release Lock After Use	162
8.2.59	ASCQM Ban Sleep Between Lock Acquisition and Release	163
8.2.60	ASCQM Ban Creation of Lock On Non-Final Object	164
8.2.61	ASCQM Ban Creation of Lock On Inappropriate Object Type	165
8.2.62	ASCQM NULL Terminate Output Of String Manipulation Primitives	166
8.2.63	ASCQM Release File Resource after Use in Class	167
8.2.64	ASCQM Use Break in Switch Statement	168
8.2.65	ASCQM Catch Exceptions	169
8.2.66	ASCQM Ban Empty Exception Block	170
8.2.67	ASCQM Initialize Resource before Use	170
8.2.68	ASCQM Ban Incompatible Lock Acquisition Sequences	172
8.2.69	ASCQM Ban Use of Thread Control Primitives with Known Deadlock Issues	173
8.2.70	ASCQM Ban Buffer Size Computation Based on Bitwise Logical Operation	173
8.2.71	ASCQM Ban Buffer Size Computation Based on Array Element Pointer Size	174
8.2.72	ASCQM Ban Buffer Size Computation Based on Incorrect String Length Value	175
8.2.73	ASCQM Ban Sequential Acquisitions of Single Non-Reentrant Lock	176
8.2.74	ASCQM Initialize Variables	177
8.2.75	ASCQM Ban Allocation of Memory with Null Size	178
8.2.76	ASCQM Ban Double Free On Pointers	179
8.2.77	ASCQM Initialize Variables before Use	180
8.2.78	ASCQM Ban Self Assignment	181
8.2.79	ASCQM Check Boolean Variables are Updated in Different Conditional Branches before Use	181
8.2.80	ASCQM Ban Not Operator On Operand Of Bitwise Operation	182
8.2.81	ASCQM Ban Not Operator On Non-Boolean Operand Of Comparison Operation	183
8.2.82	ASCQM Ban Incorrect Joint Comparison	184
8.2.83	ASCQM Secure XML Parsing with Secure Options	185
8.2.84	ASCQM Secure Use of Unsafe XML Processing with Secure Parser	186
8.2.85	ASCQM Sanitize User Input used in Path Manipulation	187
8.2.86	ASCQM Sanitize User Input used in SQL Access	189
8.2.87	ASCQM Sanitize User Input used in Document Manipulation Expression	190
8.2.88	ASCQM Sanitize User Input used in Document Navigation Expression	191
8.2.89	ASCQM Sanitize User Input used to access Directory Resources	193
8.2.90	ASCQM Sanitize Stored Input used in User Output	194
8.2.91	ASCQM Sanitize User Input used in User Output	195
8.2.92	ASCQM Sanitize User Input used in System Command	197
8.2.93	ASCQM Ban Use of Deprecated Libraries	198
8.2.94	ASCQM Sanitize User Input used as Array Index	198
8.2.95	ASCQM Check Input of Memory Allocation Primitives	200
8.2.96	ASCQM Sanitize User Input used as String Format	201
8.2.97	ASCQM Sanitize User Input used in Loop Condition	202
8.2.98	ASCQM Sanitize User Input used as Serialized Object	204
8.2.99	ASCQM Log Caught Security Exceptions	205
8.2.100	ASCQM Ban File Creation with Default Permissions	206
8.2.101	ASCQM Ban Use of Prohibited Low-Level Resource Management Functionality	207
8.2.102	ASCQM Ban Excessive Size of Index on Columns of Large Tables	209
8.2.103	ASCQM Implement Index Required by Query on Large Tables	210
8.2.104	ASCQM Ban Excessive Number of Index on Columns of Large Tables	211
8.2.105	ASCQM Ban Excessive Complexity of Data Resource Access	211
8.2.106	ASCQM Ban Expensive Operations in Loops	213
8.2.107	ASCQM Limit Number of Aggregated Non-Primitive Data Types	214
8.2.108	ASCQM Ban Excessive Number of Data Resource Access from non-stored SQL Procedure	215
8.2.109	ASCQM Ban Excessive Number of Data Resource Access from non-SQL Code	216
8.2.110	ASCQM Ban Incremental Creation of Immutable Data	217
8.2.111	ASCQM Ban Static Non-Final Data Element Outside Singleton	218
8.2.112	ASCQM Ban Conversion References to Child Class	219
8.2.113	ASCQM Ban Circular Dependencies between Modules	219
8.2.114	ASCQM Limit Volume of Commented-Out Code	221
8.2.115	ASCQM Limit Size of Operations Code	221
8.2.116	ASCQM Limit Volume of Similar Code	222

8.2.117	ASCQM Limit Algorithmic Complexity via Cyclomatic Complexity Value.....	223
8.2.118	ASCQM Limit Number of Data Access.....	223
8.2.119	ASCQM Ban Excessive Number of Children.....	225
8.2.120	ASCQM Ban Excessive Number of Inheritance Levels	226
8.2.121	ASCQM Ban Usage of Data Elements from Other Classes	226
8.2.122	ASCQM Ban Control Flow Transfer.....	227
8.2.123	ASCQM Ban Loop Value Update within Incremental and Decremental Loop	228
8.2.124	ASCQM Limit Number of Parameters.....	229
8.2.125	ASCQM Ban Unreferenced Dead Code.....	230
8.2.126	ASCQM Ban Excessive Number of Concrete Implementations to Inherit From	230
8.2.127	ASCQM Limit Number of Outward Calls	231
8.2.128	ASCQM Ban Public Data Elements.....	232
8.2.129	ASCQM Ban Hard-Coded Literals used to Initialize Variables	232
8.2.130	ASCQM Ban Logical Dead Code	233
8.2.131	ASCQM Ban Exception Definition without Ever Throwing It.....	234
8.2.132	ASCQM Ban Switch in Switch Statement	235
8.2.133	ASCQM Limit Algorithmic Complexity via Module Design Complexity Value	236
8.2.134	ASCQM Limit Algorithmic Complexity via Essential Complexity Value.....	237
8.2.135	ASCQM Use Default Case in Switch Statement	237
9	Calculation of Quality and Functional Density Measures.....	239
9.1	Calculation of the Base Measures (Normative)	239
9.2	Functional Density of Weaknesses (Non-normative)	239
10	Alternative Weighted Measures and Uses (Informative)	241
10.1	Additional Derived Measures	241
11	References (Informative).....	243
Annex A:	Consortium for IT Software Quality (CISQ) (Informative)	245
Annex B:	Common Weakness Enumeration (CWE) (Informative)	247
Annex C:	Disposition of Weaknesses from the Original CISQ Measures to This Specification	249
Annex D:	Relationship of the CISQ Structural Quality Measures to ISO 25000 Series Standards (SQuaRE) (Informative)	255

Table of Figures

iTeh STANDARD PREVIEW
(standards.iteh.ai)

Figure 1 Software Quality Characteristics from ISO/IEC 25010 with CISQ measure areas highlighted.....245
Figure 2 ISO/IEC 25020 Framework for Software Quality Characteristics Measurement.....246

Table of Tables

Table 1 Quality Measure Elements for Automated Source Code Maintainability Measure.....5
Table 2 Quality Measure Elements for Automated Source Code Performance Efficiency Measure.....8
Table 3 Quality Measure Elements for Automated Source Code Reliability Measure.....10
Table 4 Quality Measure Elements for Automated Source Code Security Measure.....17
Table 5 Software elements translated into KDM wording.....25
Table 6 Informative Weighting Schemes for Security Measurement.....231

Preface

About the OMG

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

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

More information on the OMG is available at <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
109 Highland Avenue
Suite 300
Needham, MA 02494
USA

<https://standards.iteh.ai/catalog/standards/sist/17e93555-8fbc-47c2-8495-32371a9a0d7c/iso-iec-dis-5055>

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 the OMG main web page <https://www.omg.org>, under Documents, Report a Bug/Issue.

1 Scope

1.1 Purpose

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

- Automated Source Code Maintainability Measure (ASCMM) <https://www.omg.org/spec/ASCMM/1.0/>
- Automated Source Code Performance Efficiency Measure (ASCPEM) <https://www.omg.org/spec/ASCPEM/1.0/>
- Automated Source Code Reliability Measure (ASCRM) <https://www.omg.org/spec/ASCRM/1.0/>
- Automated Source Code Security Measure (ASCSM) <https://www.omg.org/spec/ASCSM/1.0/>

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

A primary objective of updating these measures was to extend their applicability to embedded software, which is especially important for the growing implementation of embedded devices and the Internet of Things. Functionality that has traditionally been implemented in IT applications is now being moved to embedded chips. Since the weaknesses included in the measures specified in this document have been found to be applicable to all forms of software, embedded software is not treated separately in this specification.

1.2 Overview of Structural Quality Measurement in Software

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

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

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

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

ISO/IEC 25020. Each quality measure element will be represented as one or more Detection Patterns. Many quality measure elements (weaknesses) will share one 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 for each weakness, and then summing these numbers for all the weaknesses included in the specific quality characteristic measure.

2 Conformance

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

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

3 Normative References

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
- 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* <https://www.omg.org/spec/AFP/>. Needham, MA: Object Management Group.
- 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.

- Automated Function Points**—a specification for automating the counting of Function Points that mirrors as closely as possible the counting guidelines of the International Function Point User Group. (OMG, formal 2014-01-03)
- Common Weakness Enumeration**—a repository maintained by MITRE Corporation of known weaknesses in software that can be exploited to gain unauthorized entry into a software system. (cwe.mitre.org)
- Contributing Weakness**—a weakness that is represented as a child of a parent weakness in the Common Weakness Enumeration, that is, a variant instantiation of the parent weakness (cwe.mitre.org)
- Cyclomatic Complexity**—A measure of control flow complexity developed by Thomas McCabe based on a graph-theoretic analysis that reduces the control flow of a computer program to a set of edges, vertices, and their attributes that can be quantified. (McCabe, 1976)
- Detection Pattern**—a collection of parsed program elements and their relations that constitute a weakness in the software.
- Internal Software Quality**—the degree to which a set of static attributes of a software product satisfy stated and implied needs for the software product to be used under specified conditions. This will be referred to as software structural quality, or simply structural quality in this specification. (ISO/IEC 25010)
- Maintainability**—capability of a product to be modified by the intended maintainers with effectiveness and efficiency (ISO/IEC 25010)
- Parent Weakness**—a weakness in the Common Weakness Enumeration that has numerous possible instantiations in software that are represented by its relation to child CWEs (cwe.mitre.org)
- Performance Efficiency**—capability of a product to use an appropriate amount of resources under stated conditions (ISO/IEC 25010)
- Quality Measure Element**—a measure defined in terms of a software quality attribute and the measurement method for quantifying it, including optionally the transformation by a mathematical function (ISO/IEC 25010)
- Reliability**—capability a product, to perform specified functions under specified conditions for a specified period of time (ISO/IEC 25010)
- Security**— capability of a product to protect information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization, and to defend against attack patterns by malicious actors (ISO/IEC 25010)
- Software Product**—a set of computer programs, procedures, and possibly associated documentation and data. (ISO/IEC 25010)
- Software Product Quality Model**—a model that categorizes product quality properties into eight characteristics (functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability). Each characteristic is composed of a set of related sub-characteristics. (ISO/IEC 25010)
- Software Quality**—degree to which a software product satisfies stated and implied needs when used under specified conditions. (ISO/IEC 25010)
- Software Quality Attribute**—an inherent property or characteristic of software that can be distinguished quantitatively or qualitatively by human or automated means. (derived from ISO/IEC 25010)
- Software Quality Characteristic**—a set of software quality attributes that affect a specific category of software quality outcomes. (similar to but more specific than ISO/IEC 25010)
- Software Quality Characteristic Measure**—a software quality measure derived from measuring the attributes related to a specific software quality characteristic. (ISO/IEC 25020)
- Software Quality Measure**—a measure that is defined as a measurement function of two or more values of software quality measure elements. (ISO/IEC 25010)

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

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

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

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

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

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

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

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

ITIH STANDARD PREVIEW
(standards.iteh.ai)

5 Symbols (and Abbreviated Terms)

<https://standards.iteh.ai/catalog/standards/sist/17e93555-8fbc-47c2-8495-32371a9a0d7c/iso-iec-dis-5055>

AFP — Automated Function Points

ASCMM — Automated Source Code Maintainability Measure

ASCPEM — Automated Source Code Performance Efficiency Measure

ASCQM — Automated Source Code Quality Measure

ASCRM — Automated Source Code Reliability Measure

ASCSM — Automated Source Code Security Measure

CWE — Common Weakness Enumeration

CISQ — Consortium for IT Software Quality

KDM — Knowledge Discovery Metamodel

SPMS — Structured Pattern Metamodel Standard

SMM — Structured Metrics Metamodel