INTERNATIONAL STANDARD

ISO/IEC 5055

First edition
2021-03

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

Reference number
ISO/IEC 5055:2021(E)

Documents
Document Sharing Hub
© ISO/IEC 2021

Full document details: ISO IEC 05055-2021

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2021

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
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO IEC 05055-2021

Contents

1 Scope ... 1
1.1 Purpose ... 1
1.2 Overview of Structural Quality Measurement in Software ... 1

2 Conformance ... 2

3 Normative References ... 3

4 Terms and Definitions ... 4

5 Symbols (and Abbreviated Terms) ... 7

6 Weaknesses Included in Quality Measures and Representation Metamodels ... 8
6.1 Purpose ... 8
6.2 Software Product Inputs ... 8
6.3 Automated Source Code Quality Measure Elements ... 8
6.4 Automated Source Code Maintainability Measure Element Descriptions ... 9
6.5 Automated Source Code Performance Efficiency Measure Element Descriptions ... 11
6.6 Automated Source Code Reliability Measure Element Descriptions ... 15
6.7 Automated Source Code Security Measure Element Descriptions ... 23
6.8 Introduction to the Specification of Quality Measure Elements ... 32
6.9 Knowledge Discovery Metamodel (KDM) ... 32
6.10 Software Patterns Metamodel Standard (SPMS) ... 36
6.11 Reading guide ... 37

7 List of ASCQM Weaknesses ... 38
7.1 Weakness Category Maintainability ... 38
7.1.1 CWE-407 Algorithmic Complexity ... 38
7.1.2 CWE-478 Missing Default Case in Switch Statement ... 38
7.1.3 Weakness CWE-480 Use of Incorrect Operator ... 38
7.1.4 CWE-484 Omitted Break Statement in Switch ... 39
7.1.5 CWE-561 Dead Code ... 39
7.1.6 CWE-570 Expression is Always False ... 39
7.1.7 CWE-571 Expression is Always True ... 39
7.1.8 CWE-783 Operator Precedence Logic Error ... 40
7.1.9 CWE-1075 Unconditional Control Flow Transfer Outside of Switch Block ... 40
7.1.10 CWE-1121 Excessive McCabe Cyclomatic Complexity Value ... 40
7.1.11 CWE-1054 Invocation of a Control Element at an Unnecessarily Deep Horizontal Layer (Layer-skipping Call) ... 41
7.1.12 CWE-1064 Invokable Control Element with Signature Containing an Excessive Number of Parameters ... 41
7.1.13 CWE-1084 Invokable Control Element with Excessive File or Data Access Operations ... 41
7.1.14 CWE-1051 Initialization with Hard-Coded Network Resource Configuration Data ... 42
7.1.15 CWE-1090 Method Containing Access of a Member Element from Another Class ... 42
7.1.16 CWE-1074 Class with Excessively Deep Inheritance ... 42
7.1.17 CWE-1086 Class with Excessive Number of Child Classes ... 43
7.1.18 CWE-1041 Use of Redundant Code (Copy-Paste) ... 43
7.1.19 CWE-1055 Multiple Inheritance from Concrete Classes ... 43
7.1.20 CWE-1045 Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor ... 44
7.1.21 CWE-1052 Excessive Use of Hard-Coded Literals in Initialization ... 44

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

7.1.22 CWE-1048 Invokable Control Element with Large Number of Outward Calls (Excessive Coupling or Fan-out) 44
7.1.23 CWE-1095 Loop Condition Value Update within the Loop 45
7.1.24 CWE-1085 Invokable Control Element with Excessive Volume of Commented-out Code 45
7.1.25 CWE-1047 Modules with Circular Dependencies 45
7.1.26 CWE-1080 Source Code File with Excessive Number of Lines of Code 46
7.1.27 CWE-1062 Parent Class Element with References to Child Class 46
7.1.28 CWE-1087 Class with Virtual Method without a Virtual Destructor 46
7.1.29 CWE-1079 Parent Class without Virtual Destructor Method 47
7.1.30 Maintainability Detection Patterns 47
7.2 Weakness Category Performance Efficiency 48
7.2.1 CWE-401 Improper Release of Memory Before Removing Last Reference ('Memory Leak') 48
7.2.2 Weakness CWE-404 Improper Resource Shutdown or Release 48
7.2.3 CWE-424 Improper Protection of Alternate Path 49
7.2.4 CWE-772 Missing Release of Resource after Effective Lifetime 49
7.2.5 CWE-775 Missing Release of File Descriptor or Handle after Effective Lifetime 49
7.2.6 CWE-1073 Non-SQL Invokable Control Element with Excessive Number of Data Resource Access 49
7.2.7 CWE-1057 Data Access Operations Outside of Designated Data Manager Component 50
7.2.8 CWE-1043 Storable and Member Data Element Excessive Number of Aggregated Storable and Member Data Elements 50
7.2.9 CWE-1072 Data Resource Access without use of Connection Pooling 50
7.2.10 CWE-1060 Excessive Number of Inefficient Server-Side Data Accesses 51
7.2.11 CWE-1091 Use of Object without Invoking Destructor Method 51
7.2.12 CWE-1046 Creation of Immutable Text Using String Concatenation 51
7.2.13 CWE-1042 Static Member Data Element outside of a Singleton Class Element 52
7.2.14 CWE-1049 Excessive Data Query Operations in a Large Data Table 52
7.2.15 CWE-1067 Excessive Execution of Sequential Searches of Data Resource 52
7.2.16 CWE-1089 Large Data Table with Excessive Number of Indices 53
7.2.17 CWE-1094 Excessive Index Range Scan for a Data Resource 53
7.2.18 CWE-1050 Excessive Platform Resource Consumption within a Loop 53
7.2.19 CWE-1060 Excessive Number of Inefficient Server-Side Data Accesses 54
7.2.20 Performance Efficiency Detection Patterns 54
7.3 Weakness Category Reliability 54
7.3.1 CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer 54
7.3.2 CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') 55
7.3.3 CWE-123 Write-what-where Condition 55
7.3.4 CWE-125 Out-of-bounds Read 56
7.3.5 CWE-130 Improper Handling of Length Parameter Inconsistency 56
7.3.6 CWE-131 Incorrect Calculation of Buffer Size 56
7.3.7 CWE-170 Improper Null Termination 57
7.3.8 CWE-194 Unexpected Sign Extension 57
7.3.9 CWE-195 Signed to Unsigned Conversion Error 57
7.3.10 CWE-196 Unsigned to Signed Conversion Error 58
7.3.11 CWE-197 Numeric Truncation Error 58
7.3.12 CWE-248 Uncaught Exception 58
7.3.13 CWE-252 Unchecked Return Value 59
7.3.14 CWE-366 Race Condition within a Thread 59
7.3.15 CWE-369 Divide by Zero 59

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 05055-2021

7.3.16 CWE-390 Detection of Error Condition Without Action ... 59
7.3.17 CWE-391 Unchecked Error Condition ... 60
7.3.18 CWE-392 Missing Report of Error Condition ... 60
7.3.19 CWE-394 Unexpected Status Code or Return Value ... 60
7.3.20 CWE-401 Improper Release of Memory Before Removing Last Reference ('Memory Leak') ... 61
7.3.21 CWE-404 Improper Resource Shutdown or Release ... 61
7.3.22 CWE-415 Double Free ... 61
7.3.23 CWE-416 Use After Free ... 62
7.3.24 CWE-424 Improper Protection of Alternate Path ... 62
7.3.25 CWE-456 Missing Initialization of a Variable ... 62
7.3.26 CWE-459 Incomplete Cleanup ... 63
7.3.27 CWE-476 NULL Pointer Dereference ... 63
7.3.28 CWE-480 Use of Incorrect Operator ... 63
7.3.29 CWE-484 Omitted Break Statement in Switch ... 64
7.3.30 CWE-543 Use of Singleton Pattern Without Synchronization in a Multithreaded Context ... 64
7.3.31 CWE-562 Return of Stack Variable Address ... 64
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 ... 65
7.3.34 CWE-597 Use of Wrong Operator in String Comparison ... 65
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 ... 67
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 ... 68
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 ... 69
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 ... 70
7.3.49 CWE-805 Buffer Access with Incorrect Length Value ... 71
7.3.50 CWE-820 Missing Synchronization ... 71
7.3.51 CWE-821 Incorrect Synchronization ... 71
7.3.52 7.3.52 CWE-822 Untrusted Pointer Dereference ... 72
7.3.53 7.3.53 CWE-823 Use of Out-of-range Pointer Offset ... 72
7.3.54 CWE-824 Access of Uninitialized Pointer ... 72
7.3.55 CWE-825 Expired Pointer Dereference ... 73
7.3.56 CWE-833 Deadlock ... 73
7.3.57 CWE-835 Loop with Unreachable Exit Condition ('Infinite Loop') ... 73
7.3.58 CWE-908 Use of Uninitialized Resource ... 74
7.3.59 CWE-1083 Data Access from Outside Designated Data Manager Component ... 74
7.3.60 CWE-1058 Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element ... 74
7.3.61 CWE-1096 Singleton Class Instance Creation without Proper Locking or Synchronization ... 75
7.3.62 CWE-1087 Class with Virtual Method without a Virtual Destructor ... 75
7.3.63 CWE-1079 Parent Class without Virtual Destructor Method ... 75

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

7.3.64 CWE-1045 Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor 76
7.3.65 CWE-1051 Initialization with Hard-Coded Network Resource Configuration Data 76
7.3.66 CWE-1088 Synchronous Access of Remote Resource without Timeout 76
7.3.67 CWE-1066 Missing Serialization Control Element 77
7.3.68 CWE-1070 Serializable Storable Data Element with non-Serializable Item Elements 77
7.3.69 CWE-1097 Persistent Storable Data Element without Associated Comparison Control Element 77
7.3.70 CWE-1098 Data Element containing Pointer Item without Proper Copy Control Element 77
7.3.71 CWE-1082 Class Instance Self Destruction Control Element 78
7.3.72 CWE-1077 Floating Point Comparison with Incorrect Operator 78
7.3.73 CWE-665 Improper Initialization 78
7.3.74 CWE-457 Use of Uninitialized Variable 79
7.3.75 Reliability Detection Patterns 79
7.4 Weakness Category Security 80
7.4.1 Improper Restriction of Operations within the Bounds of a Memory Buffer 80
7.4.2 CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') 81
7.4.3 CWE-123 Write-what-where Condition 81
7.4.4 CWE-125 Out-of-bounds Read 82
7.4.5 CWE-129 Improper Validation of Array Index 82
7.4.6 CWE-130 Improper Handling of Length Parameter Inconsistency 82
7.4.7 CWE-131 Incorrect Calculation of Buffer Size 83
7.4.8 CWE-134 Use of Externally-Controlled Format String 83
7.4.9 CWE-194 Unexpected Sign Extension 83
7.4.10 CWE-195 Signed to Unsigned Conversion Error 83
7.4.11 CWE-196 Unsigned to Signed Conversion Error 84
7.4.12 CWE-197 Numeric Truncation Error 84
7.4.13 CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') 84
7.4.14 CWE-23 Relative Path Traversal 85
7.4.15 CWE-252 Unchecked Return Value 85
7.4.16 CWE-259 Use of Hard-coded Password 85
7.4.17 CWE-321 Use of Hard-coded Cryptographic Key 86
7.4.18 CWE-36 Absolute Path Traversal 86
7.4.19 CWE-366 Race Condition within a Thread 86
7.4.20 CWE-369 Divide by Zero 87
7.4.21 CWE-401 Improper Release of Memory Before Removing Last Reference ('Memory Leak') 87
7.4.22 CWE-404 Improper Resource Shutdown or Release 87
7.4.23 CWE-424 Improper Protection of Alternate Path 88
7.4.24 CWE-434 Unrestricted Upload of File with Dangerous Type 88
7.4.25 CWE-456 Missing Initialization of a Variable 88
7.4.26 CWE-457 Use of Uninitialized Variable 89
7.4.27 CWE-477 Use of Obsolete Function 89
7.4.28 CWE-480 Use of Incorrect Operator 89
7.4.29 CWE-502 Deserialization of Untrusted Data 90
7.4.30 CWE-543 Use of Singleton Pattern Without Synchronization in a Multithreaded Context 90
7.4.31 CWE-564 SQL Injection: Hibernate 90
7.4.32 CWE-567 Unsynchronized Access to Shared Data in a Multithreaded Context 90
7.4.33 CWE-570 Expression is Always False 91

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

7.4.34 CWE-571 Expression is Always True ... 91
7.4.35 CWE-606 Unchecked Input for Loop Condition ... 91
7.4.36 CWE-643 Improper Neutralization of Data within XPath Expressions ('XPath Injection') ... 92
7.4.37 CWE-652 Improper Neutralization of Data within XQuery Expressions ('XQuery Injection') ... 92
7.4.38 CWE-662 Improper Synchronization ... 92
7.4.39 CWE-665 Improper Initialization ... 93
7.4.40 CWE-667 Improper Locking ... 93
7.4.41 CWE-672 Operation on a Resource after Expiration or Release ... 94
7.4.42 CWE-681 Incorrect Conversion between Numeric Types ... 94
7.4.43 CWE-682 Incorrect Calculation ... 94
7.4.44 CWE-732 Incorrect Permission Assignment for Critical Resource ... 95
7.4.45 CWE-77 Improper Neutralization of Special Elements used in a Command ('Command Injection') ... 95
7.4.46 CWE-772 Missing Release of Resource after Effective Lifetime ... 95
7.4.47 CWE-775 Missing Release of File Descriptor or Handle after Effective Lifetime ... 96
7.4.48 CWE-778 Insufficient Logging ... 96
7.4.49 CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') ... 96
7.4.50 CWE-783 Operator Precedence Logic Error ... 97
7.4.51 CWE-786 Access of Memory Location Before Start of Buffer ... 97
7.4.52 CWE-787 Out-of-bounds Write ... 97
7.4.53 CWE-788 Access of Memory Location After End of Buffer ... 98
7.4.54 CWE-789 Uncontrolled Memory Allocation ... 98
7.4.55 CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') ... 98
7.4.56 CWE-798 Use of Hard-coded Credentials ... 98
7.4.57 CWE-805 Buffer Access with Incorrect Length Value ... 99
7.4.58 CWE-820 Missing Synchronization ... 99
7.4.59 CWE-821 Incorrect Synchronization ... 100
7.4.60 CWE-822 Untrusted Pointer Dereference ... 100
7.4.61 CWE-823 Use of Out-of-range Pointer Offset ... 100
7.4.62 CWE-824 Access of Uninitialized Pointer ... 101
7.4.63 CWE-825 Expired Pointer Dereference ... 101
7.4.64 CWE-835 Loop with Unreachable Exit Condition ('Infinite Loop') ... 101
7.4.65 CWE-88 Argument Injection or Modification ... 101
7.4.66 CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') ... 102
7.4.67 CWE-90 Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') ... 102
7.4.68 CWE-91 XML Injection (aka Blind XPath Injection) ... 102
7.4.69 CWE-99 Improper Control of Resource Identifiers ('Resource Injection') ... 103
7.4.70 CWE-611 Improper Restriction of XML External Entity Reference ('XXE') ... 103
7.4.71 CWE-1057 Data Access Control Element from Outside Designated Data Manager Component ... 103
7.4.72 CWE-415 Double Free ... 104
7.4.73 CWE-416 Use After Free ... 104
7.4.74 Security Detection Patterns ... 104

8 ASCQM Weakness Detection Patterns ... 107
8.1 Specification of Detection Patterns ... 107
8.2 ASCQM Check Index of Array Access ... 107

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

8.3 ASCQM Check Input of Memory Manipulation Primitives 108
8.4 ASCQM Ban String Manipulation Primitives without Boundary Checking Capabilities 109
8.5 ASCQM Check Input of String Manipulation Primitives with Boundary Checking Capabilities 109
8.6 ASCQM Ban Use of Expired Pointer 110
8.7 ASCQM Ban Input Acquisition Primitives without Boundary Checking Capabilities 111
8.8 ASCQM Check Offset used in Pointer Arithmetic 112
8.9 ASCQM Sanitize User Input used as Pointer 113
8.10 ASCQM Initialize Pointers before Use 114
8.11 ASCQM Check NULL Pointer Value before Use 115
8.12 ASCQM Ban Use of Expired Resource 116
8.13 ASCQM Ban Double Release of Resource 116
8.14 ASCQM Implement Copy Constructor for Class with Pointer Resource 117
8.15 ASCQM Ban Free Operation on Pointer Received as Parameter 118
8.16 ASCQM Ban Delete of VOID Pointer 119
8.17 ASCQM Ban Variable Increment or Decrement Operation in Operations using the Same Variable 119
8.18 ASCQM Ban Reading and Writing the Same Variable Used as Assignment Value 120
8.19 ASCQM Handle Return Value of Resource Operations 121
8.20 ASCQM Ban Incorrect Numeric Conversion of Return Value 122
8.21 ASCQM Handle Return Value of Must Check Operations 123
8.22 ASCQM Check Return Value of Resource Operations Immediately 124
8.23 ASCQM Ban Useless Handling of Exceptions 125
8.24 ASCQM Ban Incorrect Object Comparison 125
8.25 ASCQM Ban Assignment Operation Inside Logic Blocks 126
8.26 ASCQM Ban Comparison Expression Outside Logic Blocks 126
8.27 ASCQM Ban Incorrect String Comparison 127
8.28 ASCQM Ban Logical Operation with a Constant Operand 128
8.29 ASCQM Implement Correct Object Comparison Operations 128
8.30 ASCQM Ban Comma Operator from Delete Statement 129
8.31 ASCQM Release in Destructor Memory Allocated in Constructor 130
8.32 ASCQM Release Memory after Use with Correct Operation 131
8.33 ASCQM Implement Required Operations for Manual Resource Management 132
8.34 ASCQM Release Platform Resource after Use 133
8.35 ASCQM Release Memory After Use 134
8.36 ASCQM Implement Virtual Destructor for Classes Derived from Class with Virtual Destructor 135
8.37 ASCQM Implement Virtual Destructor for Parent Classes 135
8.38 ASCQM Release File Resource after Use in Operation 136
8.39 ASCQM Implement Virtual Destructor for Classes with Virtual Methods 137
8.40 ASCQM Ban Self Destruction 138
8.41 ASCQM Manage Time-Out Mechanisms in Blocking Synchronous Calls 138
8.42 ASCQM Ban Non-Final Static Data in Multi-Threaded Context 139
8.43 ASCQM Ban Non-Serializable Elements in Serializable Objects 140
8.44 ASCQM Ban Hard-Coded Literals used to Connect to Resource 141
8.45 ASCQM Ban Unintended Paths 142
8.46 ASCQM Ban Incorrect Float Number Comparison 143
8.47 ASCQM Singleton Creation without Proper Locking in Multi-Threaded Context 143
8.48 ASCQM Ban Incorrect Numeric Implicit Conversion 145
8.49 ASCQM Data Read and Write without Proper Locking in Multi-Threaded Context 146
8.50 ASCQM Ban Incorrect Synchronization Mechanisms 147

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

8.51 ASCQM Ban Resource Access without Proper Locking in Multi-Threaded Context. 148
8.52 ASCQM Ban Incorrect Type Conversion ... 149
8.53 ASCQM Ban Return of Local Variable Address ... 150
8.54 ASCQM Ban Storage of Local Variable Address in Global Variable ... 151
8.55 ASCQM Ban While TRUE Loop Without Path to Break ... 151
8.56 ASCQM Ban Unmodified Loop Variable Within Loop ... 152
8.57 ASCQM Check and Handle ZERO Value before Use as Divisor ... 153
8.58 ASCQM Ban Creation of Lock on Private Non-Static Object to Access Private Static Data ... 154
8.59 ASCQM Release Lock After Use ... 154
8.60 ASCQM Ban Sleep Between Lock Acquisition and Release ... 155
8.61 ASCQM Ban Creation of Lock on Non-Final Object ... 157
8.62 ASCQM Ban Creation of Lock on Inappropriate Object Type ... 157
8.63 ASCQM NULL Terminate Output of String Manipulation Primitives ... 158
8.64 ASCQM Release File Resource after Use in Class ... 159
8.65 ASCQM Use Break in Switch Statement ... 160
8.66 ASCQM Catch Exceptions ... 161
8.67 ASCQM Ban Empty Exception Block ... 162
8.68 ASCQM Initialize Resource before Use ... 162
8.69 ASCQM Ban Incompatible Lock Acquisition Sequences ... 164
8.70 ASCQM Ban Use of Thread Control Primitives with Known Deadlock Issues ... 164
8.71 ASCQM Ban Buffer Size Computation Based on Bitwise Logical Operation ... 165
8.72 ASCQM Ban Buffer Size Computation Based on Array Element Pointer Size ... 166
8.73 ASCQM Ban Buffer Size Computation Based on Incorrect String Length Value ... 167
8.74 ASCQM Ban Sequential Acquisitions of Single Non-Reentrant Lock ... 168
8.75 ASCQM Initialize Variables ... 169
8.76 ASCQM Ban Allocation of Memory with Null Size ... 170
8.77 ASCQM Ban Double Free on Pointers ... 170
8.78 ASCQM Initialize Variables before Use ... 171
8.79 ASCQM Ban Self Assignment ... 172
8.80 ASCQM Check Boolean Variables are Updated in Different Conditional Branches before Use ... 173
8.81 ASCQM Ban Not Operator on Operand of Bitwise Operation ... 174
8.82 ASCQM Ban Not Operator on Non-Boolean Operand of Comparison Operation ... 174
8.83 ASCQM Ban Incorrect Joint Comparison ... 175
8.84 ASCQM Secure XML Parsing with Secure Options ... 176
8.85 ASCQM Secure Use of Unsafe XML Processing with Secure Parser ... 178
8.86 ASCQM Sanitize User Input used in Path Manipulation ... 179
8.87 ASCQM Sanitize User Input used in SQL Access ... 180
8.88 ASCQM Sanitize User Input used in Document Manipulation Expression ... 181
8.89 ASCQM Sanitize User Input used in Document Navigation Expression ... 183
8.90 ASCQM Sanitize User Input used to access Directory Resources ... 184
8.91 ASCQM Sanitize Stored Input used in User Output ... 185
8.92 ASCQM Sanitize User Input used in User Output ... 187
8.93 ASCQM Sanitize User Input used in System Command ... 188
8.94 ASCQM Ban Use of Deprecated Libraries ... 189
8.95 ASCQM Sanitize User Input used as Array Index ... 190
8.96 ASCQM Check Input of Memory Allocation Primitives ... 191
8.97 ASCQM Sanitize User Input used as String Format ... 192
8.98 ASCQM Sanitize User Input used in Loop Condition ... 193
8.99 ASCQM Sanitize User Input used as Serialized Object ... 195
8.100 ASCQM Log Caught Security Exceptions ... 196
8.101 ASCQM Ban File Creation with Default Permissions ... 197

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

8.102 ASCQM Ban Use of Prohibited Low-Level Resource Management Functionality 198
8.103 ASCQM Ban Excessive Size of Index on Columns of Large Tables 200
8.104 ASCQM Implement Index Required by Query on Large Tables 201
8.105 ASCQM Ban Excessive Number of Index on Columns of Large Tables 202
8.106 ASCQM Ban Excessive Complexity of Data Resource Access 202
8.107 ASCQM Ban Expensive Operations in Loops 203
8.108 ASCQM Limit Number of Aggregated Non-Primitive Data Types 205
8.109 ASCQM Ban Excessive Number of Data Resource Access from non-stored SQL Procedure 206
8.110 ASCQM Ban Excessive Number of Data Resource Access from non-SQL Code 207
8.111 ASCQM Ban Incremental Creation of Immutable Data 208
8.112 ASCQM Ban Static Non-Final Data Element Outside Singleton 208
8.113 ASCQM Ban Conversion References to Child Class 209
8.114 ASCQM Ban Circular Dependencies between Modules 210
8.115 ASCQM Limit Volume of Commented-Out Code 211
8.116 ASCQM Limit Size of Operations Code 212
8.117 ASCQM Limit Volume of Similar Code 212
8.118 ASCQM Limit Algorithmic Complexity via Cyclomatic Complexity Value 213
8.119 ASCQM Limit Number of Data Access 214
8.120 ASCQM Ban Excessive Number of Children 215
8.121 ASCQM Ban Excessive Number of Inheritance Levels 216
8.122 ASCQM Ban Usage of Data Elements from Other Classes 216
8.123 ASCQM Ban Control Flow Transfer 217
8.124 ASCQM Ban Loop Value Update within Incremental and Decremental Loop 218
8.125 ASCQM Limit Number of Parameters 219
8.126 ASCQM Ban Unreferenced Dead Code 220
8.127 ASCQM Ban Excessive Number of Concrete Implementations to Inherit From 220
8.128 ASCQM Limit Number of Outward Calls 221
8.129 ASCQM Sanitize User Input used in Expression Language Statement 221
8.130 ASCQM Ban Hard-Coded Literals used to Initialize Variables 223
8.131 ASCQM Ban Logical Dead Code 223
8.132 ASCQM Ban Exception Definition without Ever Throwing It 224
8.133 ASCQM Ban Switch in Switch Statement 225
8.134 ASCQM Limit Algorithmic Complexity via Module Design Complexity Value 226
8.135 ASCQM Limit Algorithmic Complexity via Essential Complexity Value 227
8.136 ASCQM Use Default Case in Switch Statement 228

9 Calculation of the Quality Measures 229

Annex A Consortium for IT Software Quality (CISQ) 230

Annex B Common Weakness Enumeration (CWE) 231

Annex C Related Quality Measures 232

Annex D Relationship to ISO/IEC 25000 Series Standards 233

References 235

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see http://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by the Object Management Group (OMG) (as Automated Source Code Quality Measures [ASCQM], Version 1.0) and drafted in accordance with its editorial rules. It was adopted, under the JTC 1 PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

Introduction

Computers are being used in an increasing variety of application areas where their correct operation is critical for business accuracy or human safety. Developing or selecting trustworthy, dependable software products is of prime importance. Quantitative evaluation of software product quality is key to ensuring correct performance. This assurance requires rigorous measures of software product quality characteristics. Defining these measures in an international standard is important for standardizing the use of such measures by public and commercial organizations.

ISO/IEC 5055:2020 Automated Source Code Quality Measures defines measures at the software product level (source code) for four of the quality characteristics defined in the software product quality model presented in ISO/IEC 25010-2. These measures conform to the definitions of quality characteristics in ISO/IEC 25010-2 that they quantify. Each measure is calculated from counts of severe weaknesses in the source code that affect the quality characteristic being measured. Each weakness is associated with one or more detection patterns that can guide the development of automated tools for quality analysis of software products. The weaknesses included in each measure were selected by an international team of software engineering experts based on the severity of their impact on a software product quality characteristic.

ISO/IEC 25023 defines software measures for the quality model in ISO/IEC 25010-2, but most all of these measures are at the behavioral level For instance, a measure for availability is defined as mean downtime, which does not identify the problems in the software that caused the downtime. ISO/IEC 5055:2020 therefore supplements ISO/IEC 25023 by defining software measures at the level of weaknesses in the source code. Thus, availability would be measured by the existence of software weaknesses that cause downtime such as poor error handling or missing timeouts. ISO/IEC 5055:2020 adds strong product level measurement to support the ISO/IEC 25000 software product quality standards.

ISO/IEC 5055:2020 may be revised from time to time based on new severe weaknesses being added or existing weaknesses in the model becoming less severe because of advances in computer science. Thus, this will be a standard that adapts to changes in the technology of computing.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

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

1 Scope

1.1 Purpose

The measures in this standard 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 provide only a small set of 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 most of them are measures of external behavior and do not sufficiently 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.

© ISO/IEC 2021 – All rights reserved

Documents Document Sharing Hub
Full document details: ISO IEC 05055-2021

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 score for each of the four Automated Source Code Quality Measures from their quality measure elements is calculated 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 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***—Compliance with this specification requires that an implementation shall 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 shall be clearly described and itemized so that they can be audited by a third party.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

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, formal/2017-11-01, https://www.omg.org/spec/SPMS/1.2/
- ISO/IEC 19506:2012 – Object Management Group Architecture Driven Modernization (ADM) – Knowledge Discovery Metamodel (KDM). Also, Knowledge Discovery Metamodel, version 1.4 (KDM), formal/2016-09-01, https://www.omg.org/spec/KDM/1.4/
- MOF/XMI Mapping, version 2.5.1 (XMI), https://www.omg.org/spec/XMI/2.5.1/
- ISO/IEC 25010:2011 Systems and software engineering – System and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models
- ISO/IEC 25020:2019 Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Measurement reference model and guide
- 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.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

4 Terms and Definitions

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

4.1
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)

4.2
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)

4.3
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)

4.4
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)

4.5
Detection Pattern
an abstract representation of a set of parsed program elements and their relations described in a formal representation language that provides guidance for detecting a specific weakness in the software source code.

4.6
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)

4.7
Maintainability
capability of a product to be modified by the intended maintainers with effectiveness and efficiency (ISO/IEC 25010)

4.8
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)

4.9
Performance Efficiency
capability of a product to use an appropriate amount of resources under stated conditions (ISO/IEC 25010)

4.10

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

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)

4.11
Reliability
capability a product, to perform specified functions under specified conditions for a specified period of time (ISO/IEC 25010)

4.12
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)

4.13
Software Product
a set of computer programs, procedures, and possibly associated documentation and data. (ISO/IEC 25010)

4.14
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)

4.15
Software Quality
degree to which a software product satisfies stated and implied needs when used under specified conditions. (ISO/IEC 25010)

4.16
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)

4.17
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)

4.18
Software Quality Characteristic Measure
a software quality measure derived from measuring the attributes related to a specific software quality characteristic. (ISO/IEC 25020)

4.19
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)

4.20
Software Quality Measurement

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

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

4.21
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)

4.22
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.

4.23
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)

4.24
Structural Element
a component of software code that can be uniquely identified and counted such as a token, decision, variable, etc.

4.25
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.

4.26
Weakness
is a specific structure of program elements in the software source code, sometimes referred to as a software anti-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)

4.27
Weakness Pattern
A formal description constructed from arranging the elements output from parsing software source code into a software pattern metamodel structure that is identified as a weakness.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

5 Symbols (and Abbreviated Terms)

AFP — Automated Function Points

ASCMM — Automated Source Code Maintainability Measure

ASCPEM — Automated Source Code Performance Efficiency Measure

ASCQM — Automated Source Code Quality Measure

ASCRM — Automated Source Code Reliability Measure

ASCSM — Automated Source Code Security Measure

CWE — Common Weakness Enumeration

CISQ — Consortium for IT Software Quality

KDM — Knowledge Discovery Metamodel

SMM — Structured Metrics Metamodel

SPMS — Structured Pattern Metamodel Standard

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

6 Weaknesses Included in Quality Measures and Representation Metamodels

6.1 Purpose

The purpose of this clause is to enumerate the weaknesses constituting each measure and to provide a simple description of each measure. Clause 6.2 describes the input needed for analyzing the source code to detect the weaknesses. Clause 6.2 introduces the concept of weaknesses.as quality measure elements in calculating the measures. Clauses 6.3 to 6.6 enumerate the weaknesses included in each measure. These clauses also indicate the status of each weakness as a parent weakness or a contributor weakness that represents a specific instantiation of a parent weakness. Clause 6.8 introduces how weaknesses will be represented in formal metalanguages. Clauses 6.9 and 10 describe the metamodels that will be used for specifying weaknesses and their detection patterns. Clause 6.11 provides a reading guide for understanding the content of Clauses 6.9 and 6.10.

6.2 Software Product Inputs

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

- The entire source code for the application being analyzed.
- All materials and information required to prepare the application for production.
- A list of vetted libraries that are being used to sanitize data against potential attacks.
- What routines/API calls are being used for remote authentication, to any custom initialization and clean up routines, to synchronize resources, or to neutralize accepted file types or the names of resources.

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

6.3 Automated Source Code Quality Measure Elements

The weaknesses violating software quality rules that compose the CISQ Automated Source Code Quality Measures are grouped by quality measure in the clauses 6.4 through 6.7. Some of the weaknesses are included in more than one quality measure because they can cause several types of problems. The Common Weakness Enumeration repository (CWE, Annex B) has recently been expanded to include weaknesses from quality characteristics beyond security. All weaknesses included in these measures are identified by their CWE number from the repository. In most cases the description of CWEs is taken from information in the online repository (cwe.mitre.org). The mappings of the weaknesses from the previous CISQ measures to the current measures are presented in Annex 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. The parent contributor relationships are informative material and are presented for informational purposes only. All weaknesses listed in these tables, regardless of whether they are parents or contributors, are to be treated as Quality Measure Elements in calculating the measures presented in 6.4 through 6.7.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

6.4 Automated Source Code Maintainability Measure Element Descriptions

The quality measure elements (weaknesses violating software quality rules) that compose the CISQ Automated Source Code Maintainability Measure are presented in Table 1. This measure contains 29 parent weaknesses and no contributing weaknesses. Thus, Maintainability contains 29 Quality Measure Elements.

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

CWE #	Descriptor	Weakness Description
CWE-407	Algorithmic Complexity	An algorithm in a product has an inefficient worst-case computational complexity that may be detrimental to system performance and can be triggered by an attacker, typically using crafted manipulations that ensure that the worst case is being reached.
CWE-478	Missing Default Case in Switch Statement	The code does not have a default case in a switch statement, which might lead to complex logical errors and resultant weaknesses.
CWE-480	Use of Incorrect Operator	The programmer accidentally uses the wrong operator, which changes the application logic in security-relevant ways.
CWE-484	Omitted Break Statement in Switch	The program omits a break statement within a switch or similar construct, causing code associated with multiple conditions to execute. This can cause problems when the programmer only intended to execute code associated with one condition.
CWE-561	Dead code	The software contains dead code that can never be executed. (Thresholds are set at 5% logically dead code or 0% for code that is structurally dead. Code that exists in the source but not in the object does not count.)
CWE-570	Expression is Always False	The software contains an expression that will always evaluate to false.
CWE-571	Expression is Always True	The software contains an expression that will always evaluate to true.
CWE-783	Operator Precedence Logic Error	The program uses an expression in which operator precedence causes incorrect logic to be used.
CWE-1041	Use of Redundant Code (Copy-Paste)	The software has multiple functions, methods, procedures, macros, etc. that contain the same code. (The default threshold for each instance of copy-pasted code sets the maximum number of allowable copy-pasted instructions at 10% of the total instructions in the instance, *alternate thresholds can be set prior to analysis*).

© ISO/IEC 2021 – All rights reserved

Documents Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness Description
CWE-1045	Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor	A parent class has a virtual destructor method, but the parent has a child class that does not have a virtual destructor.
CWE-1047	Modules with Circular Dependencies	The software contains modules in which one module has references that cycle back to itself, i.e., there are circular dependencies.
CWE-1048	Invokable Control Element with Large Number of Outward Calls (Excessive Coupling or Fan-out)	The code contains callable control elements that contain an excessively large number of references to other application objects external to the context of the callable, i.e. a Fan-Out value that is excessively large. (default threshold for the maximum number of references is 5, *alternate threshold can be set prior to analysis*)
CWE-1051	Initialization with Hard-Coded Network Resource Configuration Data	The software initializes data using hard-coded values that act as network resource identifiers.
CWE-1052	Excessive Use of Hard-Coded Literals in Initialization	The software initializes a data element using a hard-coded literal that is not a simple integer or static constant element.
CWE-1054	Invocation of a Control Element at an Unnecessarily Deep Horizontal Layer (Layer-skipping Call)	The code at one architectural layer invokes code that resides at a deeper layer than the adjacent layer, i.e., the invocation skips at least one layer, and the invoked code is not part of a vertical utility layer that can be referenced from any horizontal layer.
CWE-1055	Multiple Inheritance from Concrete Classes	The software contains a class with inheritance from more than one concrete class.
CWE-1062	Parent Class Element with References to Child Class	The code has a parent class that contains references to a child class, its methods, or its members.
CWE-1064	Invokable Control Element with Signature Containing an Excessive Number of Parameters	The software contains a function, subroutine, or method whose signature has an unnecessarily large number of parameters/arguments. (default threshold for the maximum number of parameters is 7, *alternate threshold can be set prior to analysis*).
CWE-1074	Class with Excessively Deep Inheritance	A class has an inheritance level that is too high, i.e., it has many parent classes. (default threshold for maximum Inheritance levels is 7, *alternate threshold can be set prior to analysis*).
CWE-1075	Unconditional Control Flow Transfer outside of Switch Block	The software performs unconditional control transfer (such as a "goto") in code outside of a branching structure such as a switch block.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness Description
CWE-1079	Parent Class without Virtual Destructor Method	A parent class contains one or more child classes, but the parent class does not have a virtual destructor method.
CWE-1080	Source Code File with Excessive Number of Lines of Code	A source code file has too many lines of code. (default threshold for the maximum lines of code is 1000, *alternate threshold can be set prior to analysis*).
CWE-1084	Invokable Control Element with Excessive File or Data Access Operations	A function or method contains too many operations that utilize a data manager or file resource. (default threshold for the maximum number of SQL or file operations is 7, *alternate threshold can be set prior to analysis*).
CWE-1085	Invokable Control Element with Excessive Volume of Commented-out Code	A function, method, procedure, etc. contains an excessive amount of code that has been commented out within its body. (default threshold for the maximum percent of commented-out instructions is 2%, *alternate threshold can be set prior to analysis*).
CWE-1086	Class with Excessive Number of Child Classes	A class contains an unnecessarily large number of children. (default threshold for the maximum number of children of a class is 10, *alternate threshold can be set prior to analysis*).
CWE-1087	Class with Virtual Method without a Virtual Destructor	A class contains a virtual method, but the method does not have an associated virtual destructor.
CWE-1090	Method Containing Access of a Member Element from Another Class	A method for a class performs an operation that directly accesses a member element from another class.
CWE-1095	Loop Condition Value Update within the Loop	The software uses a loop with a control flow condition based on a value that is updated within the body of the loop.
CWE-1121	Excessive McCabe Cyclomatic Complexity	A module, function, method, procedure, etc. contains McCabe cyclomatic complexity that exceeds a desirable maximum. (default threshold for Cyclomatic Complexity is 20, *alternate threshold can be set prior to analysis*).

6.5 Automated Source Code Performance Efficiency Measure Element Descriptions

The quality measure elements (weaknesses violating software quality rules) that compose the CISQ Automated Source Code Performance Efficiency Measure are presented in Table 2. This measure contains 16 parent weaknesses and 3 contributing weaknesses (children in the CWE) that represent variants of these weaknesses. The CWE numbers for contributing weaknesses are presented in lighter gray cells immediately below the parent weakness whose CWE number is in a darker gray cell. Performance Efficiency contains 19 Quality Measure Elements.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

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

CWE #	Descriptor	Weakness Description
CWE-404	Improper Resource Shutdown or Release	The program does not release or incorrectly releases a resource before it is made available for re-use.
CWE-401	Improper Release of Memory Before Removing Last Reference ('Memory Leak')	The software does not sufficiently track and release allocated memory after it has been used, which slowly consumes remaining memory.
CWE-772	Missing Release of Resource after Effective Lifetime	The software does not release a resource after its effective lifetime has ended, i.e., after the resource is no longer needed.
CWE-775	Missing Release of File Descriptor or Handle after Effective Lifetime	The software does not release a file descriptor or handle after its effective lifetime has ended, i.e., after the file descriptor/handle is no longer needed. When a file descriptor or handle is not released after use (typically by explicitly closing it), attackers can cause a denial of service by consuming all available file descriptors/handles, or otherwise preventing other system processes from obtaining their own file descriptors/handles.
CWE-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.
CWE-1042	Static Member Data Element outside of a Singleton Class Element	The code contains a member element that is declared as static (but not final), in which its parent class element is not a singleton class - that is, a class element that can be used only once in the 'to' association of a Create action.
CWE-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*).
CWE-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.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness Description
CWE-1049	Excessive Data Query Operations in a Large Data Table	The software performs a data query with many 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*).
CWE-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*).
CWE-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.
CWE-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*).
CWE-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*).

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness Description
CWE-1072	Data Resource Access without Use of Connection Pooling	The software accesses a data resource through a database without using a connection pooling capability. (the use of a connection pool is technology dependent; for example, connection pooling is disabled with the addition of 'Pooling=false' to the connection string with ADO.NET or the value of a 'com.sun.jndi.ldap.connect.pool' environment parameter in Java).
CWE-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 many 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*).
CWE-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*).
CWE-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.
CWE-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 many rows. (default threshold for the index range is 10, *alternate threshold can be set prior to analysis).*

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

6.6 Automated Source Code Reliability Measure Element Descriptions

The quality measure elements (weaknesses violating software quality rules) that compose the CISQ Automated Source Code Reliability Measure are presented in Table 3. This measure contains 35 parent weaknesses and 39 contributing weaknesses (children in the CWE) that represent variants of these weaknesses. The CWE numbers for contributing weaknesses are presented in lighter gray cells immediately below the parent weakness whose CWE number is in a darker gray cell. Reliability contains 74 Quality Measure Elements.

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

CWE #	Descriptor	Weakness description
CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	The software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer.
CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.
CWE-123	Write-what-where condition	Any condition where the attacker can write an arbitrary value to be written to an arbitrary location, often as the result of a buffer overflow.
CWE-125	Out-of-bounds read	The software reads data past the end, or before the beginning, of the intended buffer.
CWE-130	Improper Handling of Length Parameter Inconsistency	The software parses a formatted message or structure, but it does not handle or incorrectly handles a length field that is inconsistent with the actual length of the associated data.
CWE-786	Access of Memory Location Before Start of Buffer	The software reads or writes to a buffer using an index or pointer that references a memory location prior to the beginning of the buffer. This typically occurs when a pointer or its index is decremented to a position before the buffer, when pointer arithmetic results in a position before the beginning of the valid memory location, or when a negative index is used.
CWE-787	Out-of-bounds Write	The software writes data past the end, or before the beginning, of the intended buffer. The software may modify an index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-788	Access of Memory Location After End of Buffer	The software reads or writes to a buffer using an index or pointer that references a memory location after the end of the buffer. This typically occurs when a pointer or its index is decremented to a position before the buffer; when pointer arithmetic results in a position before the buffer; or when a negative index is used, which generates a position before the buffer.
CWE-805	Buffer Access with Incorrect Length Value	The software uses a sequential operation to read or write a buffer, but it uses an incorrect length value that causes it to access memory that is outside of the bounds of the buffer.
CWE-822	Untrusted Pointer Dereference	The program obtains a value from an untrusted source, converts this value to a pointer, and dereferences the resulting pointer. There are several variants of this weakness, including but not necessarily limited to: The untrusted value is directly invoked as a function call. In OS kernels or drivers where there is a boundary between "userland" and privileged memory spaces, an untrusted pointer might enter through an API or system call (see CWE-781 for one such example). Inadvertently accepting the value from an untrusted control sphere when it did not have to be accepted as input at all. This might occur when the code was originally developed to be run by a single user in a non-networked environment, and the code is then ported to or otherwise exposed to a networked environment.
CWE-823	Use of Out-of-range Pointer Offset	The program performs pointer arithmetic on a valid pointer, but it uses an offset that can point outside of the intended range of valid memory locations for the resulting pointer. While a pointer can contain a reference to any arbitrary memory location, a program typically only intends to use the pointer to access limited portions of memory, such as contiguous memory used to access an individual array. Programs may use offsets to access fields or sub-elements stored within structured data. The offset might be out-of-range if it comes from an untrusted source, is the result of an incorrect calculation, or occurs because of another error.
CWE-824	Access of Uninitialized Pointer	The program accesses or uses a pointer that has not been initialized. If the pointer contains an uninitialized value, then the value might not point to a valid memory location.
CWE-825	Expired Pointer Dereference	The program dereferences a pointer that contains a location for memory that was previously valid, but is no longer valid.
CWE-170	Improper Null Termination	The software does not terminate or incorrectly terminates a string or array with a null character or equivalent terminator.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-252	Unchecked Return Value	The software does not check the return value from a method or function, which can prevent it from detecting unexpected states and conditions.
CWE-390	Detection of Error Condition Without Action	The software detects a specific error, but takes no actions to handle the error. For instance, where an exception handling block (such as Catch and Finally blocks) do not contain any instruction, making it impossible to accurately identify and adequately respond to unusual and unexpected conditions.
CWE-394	Unexpected Status Code or Return Value	The software does not properly check when a function or operation returns a value that is legitimate for the function, but is not expected by the software.
CWE-404	Improper Resource Shutdown or Release	The program does not release or incorrectly releases a resource before it is made available for re-use.
CWE-401	Improper Release of Memory Before Removing Last Reference ('Memory Leak')	The software does not sufficiently track and release allocated memory after it has been used, which slowly consumes remaining memory.
CWE-772	Missing Release of Resource after Effective Lifetime	The software does not release a resource after its effective lifetime has ended, i.e., after the resource is no longer needed.
CWE-775	Missing Release of File Descriptor or Handle after Effective Lifetime	The software does not release a file descriptor or handle after its effective lifetime has ended, i.e., after the file descriptor/handle is no longer needed. When a file descriptor or handle is not released after use (typically by explicitly closing it), attackers can cause a denial of service by consuming all available file descriptors/handles, or otherwise preventing other system processes from obtaining their own file descriptors/handles.
CWE-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.
CWE-459	Incomplete Clean-up	The software does not properly "clean up" and remove temporary or supporting resources after they have been used.
CWE-476	NULL Pointer Dereference	A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-480	Use of Incorrect Operator	The programmer accidentally uses the wrong operator, which changes the application logic in security-relevant ways.
CWE-484	Omitted Break Statement in Switch	The program omits a break statement within a switch or similar construct, causing code associated with multiple conditions to execute. This can cause problems when the programmer only intended to execute code associated with one condition.
CWE-562	Return of Stack Variable Address	A function returns the address of a stack variable, which will cause unintended program behavior, typically in the form of a crash. Because local variables are allocated on the stack, when a program returns a pointer to a local variable, it is returning a stack address. A subsequent function call is likely to re-use this same stack address, thereby overwriting the value of the pointer, which no longer corresponds to the same variable since a function's stack frame is invalidated when it returns. At best this will cause the value of the pointer to change unexpectedly. In many cases it causes the program to crash the next time the pointer is dereferenced.
CWE-595	Comparison of Object References Instead of Object Contents	The program compares object references instead of the contents of the objects themselves, preventing it from detecting equivalent objects.
CWE-597	Use of Wrong Operator in String Comparison	The software uses the wrong operator when comparing a string, such as using "==" when the equals() method should be used instead. In Java, using == or != to compare two strings for equality compares two objects for equality, not their values.
CWE-1097	Persistent Storable Data Element without Associated Comparison Control Element	The software uses a storable data element that does not have all the associated functions or methods that are necessary to support comparison. Remove instances where the persistent data has missing or improper dedicated comparison operations. Note: * In case of technologies with classes, this means situations where a persistent field is from a class that is made persistent while it does not implement methods from the list of required comparison operations (a JAVA example is the list composed of {'hashCode()','equals()'} methods)
CWE-662	Improper Synchronization	The software attempts to use a shared resource in an exclusive manner, but does not prevent or incorrectly prevents use of the resource by another thread or process.
CWE-366	Race Condition within a Thread	If two threads of execution use a resource simultaneously, there exists the possibility that resources may be used while invalid, in turn making the state of execution undefined.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context	The software uses the singleton pattern when creating a resource within a multithreaded environment.
CWE-567	Unsynchronized Access to Shared Data in a Multithreaded Context	The product does not properly synchronize shared data, such as static variables across threads, which can lead to undefined behavior and unpredictable data changes.
CWE-667	Improper Locking	The software does not properly acquire a lock on a resource, or it does not properly release a lock on a resource, leading to unexpected resource state changes and behaviors.
CWE-764	Multiple Locks of a Critical Resource	The software locks a critical resource more times than intended, leading to an unexpected state in the system.
CWE-820	Missing Synchronization	The software utilizes a shared resource in a concurrent manner but does not attempt to synchronize access to the resource.
CWE-821	Incorrect Synchronization	The software utilizes a shared resource in a concurrent manner, but it does not correctly synchronize access to the resource.
CWE-1058	Invokable Control Element in Multi-Thread Context with non-Final Static Storable or Member Element	The code contains a function or method that operates in a multi-threaded environment but owns an unsafe non-final static storable or member data element.
CWE-1096	Singleton Class Instance Creation without Proper Locking or Synchronization	The software implements a Singleton design pattern but does not use appropriate locking or other synchronization mechanism to ensure that the singleton class is only instantiated once.
CWE-665	Improper Initialization	The software does not initialize or incorrectly initializes a resource, which might leave the resource in an unexpected state when it is accessed or used.
CWE-456	Missing Initialization of a Variable	The software does not initialize critical variables, which causes the execution environment to use unexpected values.
CWE-457	Use of uninitialized variable	The code uses a variable that has not been initialized, leading to unpredictable or unintended results.
CWE-672	Operation on a Resource after Expiration or Release	The software uses, accesses, or otherwise operates on a resource after that resource has been expired, released, or revoked.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-415	Double Free	The product calls free() twice on the same memory address, potentially leading to modification of unexpected memory locations.
CWE-416	Use After Free	Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code.
CWE-681	Incorrect Conversion between Numeric Types	When converting from one data type to another, such as long to integer, data can be omitted or translated in a way that produces unexpected values. If the resulting values are used in a sensitive context, then dangerous behaviors may occur. For instance, if the software declares a variable, field, member, etc. with a numeric type, and then updates it with a value from a second numeric type that is incompatible with the first numeric type.
CWE-194	Unexpected Sign Extension	The software performs an operation on a number that causes it to be sign-extended when it is transformed into a larger data type. When the original number is negative, this can produce unexpected values that lead to resultant weaknesses.
CWE-195	Signed to Unsigned Conversion Error	The software uses a signed primitive and performs a cast to an unsigned primitive, which can produce an unexpected value if the value of the signed primitive cannot be represented using an unsigned primitive.
CWE-196	Unsigned to Signed Conversion Error	The software uses an unsigned primitive and performs a cast to a signed primitive, which can produce an unexpected value if the value of the unsigned primitive cannot be represented using a signed primitive.
CWE-197	Numeric Truncation Error	Truncation errors occur when a primitive is cast to a primitive of a smaller size and data is lost in the conversion. When a primitive is cast to a smaller primitive, the high order bits of the large value are lost in the conversion, potentially resulting in an unexpected value that is not equal to the original value. This value may be required as an index into a buffer, a loop iterator, or simply necessary state data. In any case, the value cannot be trusted and the system will be in an undefined state. While this method may be employed viably to isolate the low bits of a value, this usage is rare, and truncation usually implies that an implementation error has occurred.
CWE-682	Incorrect Calculation	The software performs a calculation that generates incorrect or unintended results that are later used in security-critical decisions or resource management.
CWE-131	Incorrect Calculation of Buffer Size	The software does not correctly calculate the size to be used when allocating a buffer, which could lead to a buffer overflow.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-369	Divide by Zero	The product divides a value by zero.
CWE-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.
CWE-248	Uncaught Exception	An exception is thrown from a function, but it is not caught.
CWE-391	Unchecked Error Condition	Ignoring exceptions and other error conditions may allow an attacker to induce unexpected behavior unnoticed.
CWE-392	Missing Report of Error Condition	The software encounters an error but does not provide a status code or return value to indicate that an error has occurred.
CWE-704	Incorrect Type Conversion or Cast	The software does not correctly convert an object, resource, or structure from one type to a different type.
CWE-758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	The software uses an API function, data structure, or other entity in a way that relies on properties that are not always guaranteed to hold for that entity.
CWE-833	Deadlock	The software contains multiple threads or executable segments that are waiting for each other to release a necessary lock, resulting in deadlock.
CWE-835	Loop with Unreachable Exit Condition ('Infinite Loop')	The program contains an iteration or loop with an exit condition that cannot be reached, i.e., an infinite loop.
CWE-908	Use of Uninitialized Resource	The software uses a resource that has not been properly initialized.
CWE-1045	Parent Class with a Virtual Destructor and a Child Class without a Virtual Destructor	A parent class has a virtual destructor method, but the parent has a child class that does not have a virtual destructor.
CWE-1051	Initialization with Hard-Coded Network Resource Configuration Data	The software initializes data using hard-coded values that act as network resource identifiers.
CWE-1066	Missing Serialization Control Element	The software contains a serializable data element that does not have an associated serialization method.
CWE-1070	Serializable Data Element Containing Non-Serializable Item Elements	The software contains a serializable, storable data element such as a field or member, but the data element contains member elements that are not serializable.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-1077	Floating Point Comparison with Incorrect Operator	The code performs a comparison such as an equality test between two float (floating point) values, but it uses comparison operators that do not account for the possibility of loss of precision. Numeric calculation using floating point values can generate imprecise results because of rounding errors. As a result, two different calculations might generate numbers that are mathematically equal, but have slightly different bit representations that do not translate to the same mathematically equal values. As a result, an equality test or other comparison might produce unexpected results. (an example in JAVA, is the use of '= =' or '! =') instead of being checked for precision.
CWE-1079	Parent Class without Virtual Destructor Method	A parent class contains one or more child classes, but the parent class does not have a virtual destructor method.
CWE-1082	Class Instance Self Destruction Control Element	The code contains a class instance that calls the method or function to delete or destroy itself. (an example of a self-destruction in C++ is 'delete this')
CWE-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: 1) 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. 2) If there is no dedicated data access component, every data access is a violation. 3) For some embedded software that requires access to data from anywhere, the whole software is defined as a data access component. This condition must be identified as input to the analysis.
CWE-1087	Class with Virtual Method without a Virtual Destructor	A class contains a virtual method, but the method does not have an associated virtual destructor.
CWE-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.
CWE-1098	Data Element containing Pointer Item without Proper Copy Control Element	The code contains a data element with a pointer that does not have an associated copy or constructor method.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

6.7 Automated Source Code Security Measure Element Descriptions

The quality measure elements (weaknesses violating software quality rules) that compose the CISQ Automated Source Code Security Measure are presented in Table 4. This measure contains 36 parent weaknesses and 37 contributing weaknesses (children in the CWE) that represent variants of these weaknesses. The CWE numbers for contributing weaknesses are presented in lighter gray cells immediately below the parent weakness whose CWE number is in a darker gray cell. Security contains 73 Quality Measure Elements.

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

CWE #	Descriptor	Weakness description
CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	The software uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory, but the software does not properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory.
CWE-23	Relative Path Traversal	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize sequences such as ".." that can resolve to a location that is outside of that directory.
CWE-36	Absolute Path Traversal	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly neutralize absolute path sequences such as "/abs/path" that can resolve to a location that is outside of that directory.
CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	The software constructs all or part of a command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended command when it is sent to a downstream component.
CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	The software constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.
CWE-88	Argument Injection or Modification	The software does not sufficiently delimit the arguments being passed to a component in another control sphere, allowing alternate arguments to be provided, leading to potentially security-relevant changes.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users. Cross-site scripting (XSS) vulnerabilities occur when: 1. Untrusted data enters a web application, typically from a web request. 2. The web application dynamically generates a web page that contains this untrusted data. 3. During page generation, the application does not prevent the data from containing content that is executable by a web browser, such as JavaScript, HTML tags, HTML attributes, mouse events, Flash, ActiveX, etc. 4. A victim visits the generated web page through a web browser, which contains malicious script that was injected using the untrusted data. 5. Since the script comes from a web page that was sent by the web server, the victim's web browser executes the malicious script in the context of the web server's domain. 6. This effectively violates the intention of the web browser's same-origin policy, which states that scripts in one domain should not be able to access resources or run code in a different domain.
CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.
CWE-564	SQL Injection: Hibernate	Using Hibernate to execute a dynamic SQL statement built with user-controlled input can allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands.
CWE-90	Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')	The software constructs all or part of an LDAP query using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended LDAP query when it is sent to a downstream component.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-91	XML Injection (aka Blind XPath Injection)	The software does not properly neutralize special elements that are used in XML, allowing attackers to modify the syntax, content, or commands of the XML before it is processed by an end system.
CWE-99	Improper Control of Resource Identifiers ('Resource injection')	The software receives input from an upstream component, but it does not restrict or incorrectly restricts the input before it is used as an identifier for a resource that may be outside the intended sphere of control.
CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	The software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer.
CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.
CWE-123	Write-what-where condition	Any condition where the attacker can write an arbitrary value to an arbitrary location, often as the result of a buffer overflow.
CWE-125	Out-of-bounds Read	The software reads data past the end, or before the beginning, of the intended buffer.
CWE-130	Improper Handling of Length Parameter Inconsistency	The software parses a formatted message or structure, but it does not handle or incorrectly handles a length field that is inconsistent with the actual length of the associated data.
CWE-786	Access of Memory Location Before Start of Buffer	The software reads or writes to a buffer using an index or pointer that references a memory location prior to the beginning of the buffer. This typically occurs when a pointer or its index is decremented to a position before the buffer, when pointer arithmetic results in a position before the beginning of the valid memory location, or when a negative index is used.
CWE-787	Out-of-bounds Write	The software writes data past the end, or before the beginning, of the intended buffer. The software may modify an index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-788	Access of Memory Location After End of Buffer	The software reads or writes to a buffer using an index or pointer that references a memory location after the end of the buffer. This typically occurs when a pointer or its index is decremented to a position before the buffer; when pointer arithmetic results in a position before the buffer; or when a negative index is used, which generates a position before the buffer.
CWE-805	Buffer Access with Incorrect Length Value	The software uses a sequential operation to read or write a buffer, but it uses an incorrect length value that causes it to access memory that is outside of the bounds of the buffer.
CWE-822	Untrusted Pointer Dereference	The program obtains a value from an untrusted source, converts this value to a pointer, and dereferences the resulting pointer. There are several variants of this weakness, including but not necessarily limited to: The untrusted value is directly invoked as a function call. In OS kernels or drivers where there is a boundary between "userland" and privileged memory spaces, an untrusted pointer might enter through an API or system call (see CWE-781 for one such example). Inadvertently accepting the value from an untrusted control sphere when it did not have to be accepted as input at all. This might occur when the code was originally developed to be run by a single user in a non-networked environment, and the code is then ported to or otherwise exposed to a networked environment.
CWE-823	Use of Out-of-range Pointer Offset	The program performs pointer arithmetic on a valid pointer, but it uses an offset that can point outside of the intended range of valid memory locations for the resulting pointer. While a pointer can contain a reference to any arbitrary memory location, a program typically only intends to use the pointer to access limited portions of memory, such as contiguous memory used to access an individual array. Programs may use offsets to access fields or sub-elements stored within structured data. The offset might be out-of-range if it comes from an untrusted source, is the result of an incorrect calculation, or occurs because of another error.
CWE-824	Access of Uninitialized Pointer	The program accesses or uses a pointer that has not been initialized. If the pointer contains an uninitialized value, then the value might not point to a valid memory location.
CWE-825	Expired Pointer Dereference	The program dereferences a pointer that contains a location for memory that was previously valid, but is no longer valid.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-129	Improper Validation of Array Index	The product uses untrusted input when calculating or using an array index, but the product does not validate or incorrectly validates the index to ensure the index references a valid position within the array.
CWE-134	Use of Externally Controlled Format String	The software uses a function that accepts a format string as an argument, but the format string originates from an external source.
CWE-252	Unchecked Return Value	The software does not check the return value from a method or function, which can prevent it from detecting unexpected states and conditions.
CWE-404	Improper Resource Shutdown or Release	The program does not release or incorrectly releases a resource before it is made available for re-use.
CWE-401	Improper Release of Memory Before Removing Last Reference ('Memory Leak')	The software does not sufficiently track and release allocated memory after it has been used, which slowly consumes remaining memory.
CWE-772	Missing Release of Resource after Effective Lifetime	The software does not release a resource after its effective lifetime has ended, i.e., after the resource is no longer needed.
CWE-775	Missing Release of File Descriptor or Handle after Effective Lifetime	The software does not release a file descriptor or handle after its effective lifetime has ended, i.e., after the file descriptor/handle is no longer needed. When a file descriptor or handle is not released after use (typically by explicitly closing it), attackers can cause a denial of service by consuming all available file descriptors/handles, or otherwise preventing other system processes from obtaining their own file descriptors/handles.
CWE-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.
CWE-434	Unrestricted Upload of File with Dangerous Type	The software allows the upload or transfer files of dangerous types that can be automatically processed within the product's environment.
CWE-477	Use of Obsolete Function	The code uses deprecated or obsolete functions, which suggests that the code has not been actively reviewed or maintained.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-480	Use of Incorrect Operator	The programmer accidentally uses the wrong operator, which changes the application logic in security-relevant ways.
CWE-502	Deserialization of Untrusted Data	The application deserializes untrusted data without sufficiently verifying that the resulting data will be valid.
CWE-570	Expression is Always False	The software contains an expression that will always evaluate to false.
CWE-571	Expression Is Always True	The software contains an expression that will always evaluate to true.
CWE-606	Unchecked Input for Loop Condition	The product does not properly check inputs that are used for loop conditions, potentially leading to a denial of service because of excessive looping.
CWE-611	Improper Restriction of XML External Entity Reference ('XXE')	The software processes an XML document that can contain XML entities with URIs that resolve to documents outside of the intended sphere of control, causing the product to embed incorrect documents into its output.
CWE-643	Improper Neutralization of Data within XPath Expressions ('XPath Injection')	The software uses external input to dynamically construct an XPath expression used to retrieve data from an XML database, but it does not neutralize or incorrectly neutralizes that input. This allows an attacker to control the structure of the query.
CWE-652	CWE-652 Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')	The software uses external input to dynamically construct a XQuery expression used to retrieve data from an XML database, but it does not neutralize or incorrectly neutralizes that input. This allows an attacker to control the structure of the query.
CWE-665	Improper Initialization	The software does not initialize or incorrectly initializes a resource, which might leave the resource in an unexpected state when it is accessed or used.
CWE-456	Missing Initialization of a Variable	The software does not initialize critical variables, which causes the execution environment to use unexpected values.
CWE-457	Use of uninitialized variable	The software uses a variable that has not been initialized leading to unpredictable or unintended results.
CWE-662	Improper Synchronization	The software attempts to use a shared resource in an exclusive manner, but does not prevent or incorrectly prevents use of the resource by another thread or process.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-366	Race Condition within a Thread	If two threads of execution use a resource simultaneously, there exists the possibility that resources may be used while invalid, in turn making the state of execution undefined.
CWE-543	Use of Singleton Pattern Without Synchronization in a Multithreaded Context	The software uses the singleton pattern when creating a resource within a multithreaded environment.
CWE-567	Unsynchronized Access to Shared Data in a Multithreaded Context	The product does not properly synchronize shared data, such as static variables across threads, which can lead to undefined behavior and unpredictable data changes.
CWE-667	Improper Locking	The software does not properly acquire a lock on a resource, or it does not properly release a lock on a resource, leading to unexpected resource state changes and behaviors.
CWE-820	Missing Synchronization	The software utilizes a shared resource in a concurrent manner but does not attempt to synchronize access to the resource.
CWE-821	Incorrect Synchronization	The software utilizes a shared resource in a concurrent manner, but it does not correctly synchronize access to the resource.
CWE-672	Operation on a Resource after Expiration or Release	The software uses, accesses, or otherwise operates on a resource after that resource has been expired, released, or revoked.
CWE-415	Double Free	The product calls free() twice on the same memory address, potentially leading to modification of unexpected memory locations.
CWE-416	Use After Free	Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code.
CWE-681	Incorrect Conversion between Numeric Types	When converting from one data type to another, such as long to integer, data can be omitted or translated in a way that produces unexpected values. If the resulting values are used in a sensitive context, then dangerous behaviors may occur. For instance, if the software declares a variable, field, member, etc. with a numeric type, and then updates it with a value from a second numeric type that is incompatible with the first numeric type.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-194	Unexpected Sign Extension	The software performs an operation on a number that causes it to be sign-extended when it is transformed into a larger data type. When the original number is negative, this can produce unexpected values that lead to resultant weaknesses.
CWE-195	Signed to Unsigned Conversion Error	The software uses a signed primitive and performs a cast to an unsigned primitive, which can produce an unexpected value if the value of the signed primitive cannot be represented using an unsigned primitive.
CWE-196	Unsigned to Signed Conversion Error	The software uses an unsigned primitive and performs a cast to a signed primitive, which can produce an unexpected value if the value of the unsigned primitive cannot be represented using a signed primitive.
CWE-197	Numeric Truncation Error	Truncation errors occur when a primitive is cast to a primitive of a smaller size and data is lost in the conversion. When a primitive is cast to a smaller primitive, the high order bits of the large value are lost in the conversion, potentially resulting in an unexpected value that is not equal to the original value. This value may be required as an index into a buffer, a loop iterator, or simply necessary state data. In any case, the value cannot be trusted and the system will be in an undefined state. While this method may be employed viably to isolate the low bits of a value, this usage is rare, and truncation usually implies that an implementation error has occurred.
CWE-682	Incorrect Calculation	The software performs a calculation that generates incorrect or unintended results that are later used in security-critical decisions or resource management.
CWE-131	Incorrect Calculation of Buffer Size	The software does not correctly calculate the size to be used when allocating a buffer, which could lead to a buffer overflow.
CWE-369	Divide by Zero	The product divides a value by zero.
CWE-732	Incorrect Permission Assignment for Critical Resource	The software specifies permissions for a security-critical resource in a way that allows that resource to be read or modified by unintended actors.
CWE-778	Insufficient Logging	When a security-critical event occurs, the software either does not record the event or omits important details about the event when logging it.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

CWE #	Descriptor	Weakness description
CWE-783	Operator Precedence Logic Error	The program uses an expression in which operator precedence causes incorrect logic to be used. While often just a bug, operator precedence logic errors can have serious consequences if they are used in security-critical code, such as making an authentication decision.
CWE-789	Uncontrolled Memory Allocation	The product allocates memory based on an untrusted size value, but it does not validate or incorrectly validates the size, allowing arbitrary amounts of memory to be allocated.
CWE-798	Use of Hard-coded Credentials	The software contains hard-coded credentials, such as a password or cryptographic key, which it uses for its own inbound authentication, outbound communication to external components, or encryption of internal data.
CWE-259	Use of Hard-coded Password	The software contains a hard-coded password, which it uses for its own inbound authentication or for outbound communication to external components.
CWE-321	Use of Hard-coded Cryptographic Key	The use of a hard-coded cryptographic key significantly increases the possibility that encrypted data may be recovered.
CWE-835	Loop with Unreachable Exit Condition ('Infinite Loop')	The program contains an iteration or loop with an exit condition that cannot be reached, i.e., an infinite loop.
CWE-917	Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')	The software constructs all or part of an expression language (EL) statement in a Java Server Page (JSP) using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended EL statement before it is executed.
CWE-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.

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

6.8 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.

6.9 Knowledge Discovery Metamodel (KDM)

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

1. The descriptions include information such as (MethodUnit), (Reads), (ManagesResource), … to identify the KDM entities included in the pattern definition.
2. The descriptions only describe the salient aspects of the pattern since the specifics can be technology or language dependent.

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 clause. However, for readers not familiar with KDM, Table 5 presents a primer

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 05055-2021

Table 5 — Software elements translated into KDM wording

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

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

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

SQL stored procedures	`DataModel`
	`  RelationalSchema ...`
	`  CallableUnit id="cu1" kind="stored" ...`

return code value definition and use	`CallableUnit|MethodUnit id="ce1" type="ce1_signature" ...`
	`  Signature "ce1_signature"`
	`  ParameterUnit id="pu1" kind="return" ...`
	`Value|StorableUnit|MemberUnit id="de1" ...`
	`ActionElement id="ae1"`
	`kind="Call|PtrCall|MethodCall|VirtualCall" ...`
	`  Calls "ce1"`
	`  Reads "de1"`

exception	`CallableUnit|MethodUnit id="ce1" type="ce1_signature" ...`
	`  Signature "ce1_signature"`
	`  ParameterUnit id="pu1" kind="exception" ...`

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

Software element	KDM Outline
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 ...

© ISO/IEC 2021 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 05055-2021

Software element	KDM Outline
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"

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

Software element	KDM Outline

checked	`Value|StorableUnit|MemberUnit id="de1" ...`
	`ActionElement id="ae1"`
	`kind="Equals|NotEqualTo|GreaterThan|GreaterThanOrEqual|LessThan|LessThanOrEqual" ...`
	`  Reads "de1"`

6.10 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.

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

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"

6.11 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 modeling associated with a weakness pattern for a specific weakness associated with the Software Quality characteristic.

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) detection patterns with reference numbers to subclauses in clause 8
    - 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

For each numbered sub-clause in clause 8:

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

© ISO/IEC 2021 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021

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

7 List of ASCQM Weaknesses

7.1 Weakness Category Maintainability

7.1.1 CWE-407 Algorithmic Complexity

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

Roles
- the <ControlFlow>

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

7.1.2 CWE-478 Missing Default Case in Switch Statement

Reference
https://cwe.mitre.org/data/definitions/478

Roles
- the <SwitchStatement>

Detection Patterns
8.2.135 ASCQM Use Default Case in Switch Statement

7.1.3 Weakness CWE-480 Use of Incorrect Operator

Reference

https://cwe.mitre.org/data/definitions/480

Roles
- the <Operator>

© ISO/IEC 2021 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 05055-2021