CISQ

Consortium for Information & Software Quality ™

List of Weaknesses Included in the CISQ Automated Source Code Quality Measures

June 2019

Overview of Structural Quality Measurement in Software

Measurement of the structural quality characteristics of software has a long history in software engineering. 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. This specification provides automated measures for four structural quality characteristics listed in the ISO/IEC 25010 software quality model that can be calculated from 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. Good architectural and coding practices can be stated as rules for engineering software products. Violations of these rules will be called weaknesses to be consistent with terms used in the Common Weakness Enumeration which lists the weaknesses used in these measures.

The four Automated Source Code Quality Measures are calculated from counts of what industry experts have determined to be most severe weaknesses. Consequently, they provide strong indicators of the quality of a software system and the probability of operational or cost problems related to each measure’s domain.

The weaknesses comprising each CISQ Automated Source Code Quality Measure are grouped by measure in the Tables 1-4. 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 (an ITU standard) 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. The title and description of CWEs is taken from information in the online CWE repository (cwe.mitre.org). Each weakness will be described as a ‘quality measure element’ to remain consistent with the structure of software quality measures enumerated in ISO/IEC 25020.

Some weaknesses drawn from the CWE repository (parent weaknesses) have related weaknesses listed as ‘contributing weaknesses’ (‘child weaknesses’ 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 in this standard. Compliance to the CISQ measures is assessed at the level of the parent weakness. A technology must be able to detect at least one of the contributing weaknesses to be assessed compliant on the parent weakness.

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.

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*).
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 a large number of 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.
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*).

The cells containing CWE IDs for parents are presented in a dark blue.
The cells containing contributing weaknesses are presented in a light blue.

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 15 parent weaknesses and 3 contributing weaknesses (children in the CWE) that represent variants of these weaknesses. The CWE numbers for contributing weaknesses is presented in light blue cells immediately below the parent weakness whose CWE number is in a dark blue cell.

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

CWE #	Descriptor	Weakness Description
CWE-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.
CWE-1049	**Excessive Data Query Operations in a Large Data Table**	The software performs a data query with a large number of joins and sub-queries on a large data table. (default thresholds are 5 joins, 3 sub-queries, and 1,000,000 rows for a large table, *alternate thresholds for all three parameters can be set prior to analysis*).
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*).

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 a large number of data accesses/queries that are sent through a data manager, i.e., does not use efficient database capabilities. (default threshold for the maximum number of data queries is 2, *alternate threshold can be set prior to analysis*).
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 a large number of rows. (default threshold for the index range is 10, *alternate threshold can be set prior to analysis).*

The cells containing CWE IDs for parents are presented in a dark blue.
The cells containing contributing weaknesses are presented in a light blue.

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 is presented in light blue cells immediately below the parent weakness whose CWE number is in a dark blue cell.

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

CWE #	Descriptor	Weakness description
CWE-119	**Improper 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 has the ability to 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.
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.
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 Cleanup**	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.
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 actually 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 of 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.
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.
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.
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 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.
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: □□□The dedicated data access component can be either client-side or server-side, which means that data access components can be developed using non-SQL language. □□□If there is no dedicated data access component, every data access is a violation. □□□For some embedded software that requires access to data from anywhere, the whole software is defined as a data access component. This condition must be identified as input to the analysis.
CWE-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.

The cells containing CWE IDs for parents are presented in a dark blue.
The cells containing contributing weaknesses are presented in a light blue.

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 38 contributing weaknesses (children in the CWE) that represent variants of these weaknesses. The CWE numbers for contributing weaknesses are presented in light blue cells immediately below the parent weakness whose CWE number is in a dark blue cell.

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

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

CWE-79	**Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')**	The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users. 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.
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 has the ability to 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.
CWE-788	**Access of Memory Location After End of Buffer**	The software reads or writes to a buffer using an index or pointer that references a memory location after the end of the buffer. This typically occurs when a pointer or its index is decremented to a position before the buffer; when pointer arithmetic results in a position before the buffer; or when a negative index is used, which generates a position before the buffer.
CWE-805	**Buffer Access with Incorrect Length Value**	The software uses a sequential operation to read or write a buffer, but it uses an incorrect length value that causes it to access memory that is outside of the bounds of the buffer.

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

The cells containing CWE IDs for parents are presented in a dark blue.
The cells containing contributing weaknesses are presented in a light blue.

Calculation of Quality and Functional Density Measures

Calculation of the Base Measures

After reviewing several alternatives, a count of total violations of quality rules was selected as the best option for a base measure for each of the four software quality characteristics covered in this specification. Software quality characteristic measures have frequently been scored at the component level and then aggregated to develop an overall score for the application. However, scoring at the component level was rejected because many violations of quality rules cannot be isolated to a single component, but rather involve interactions among several components. Therefore, each Automated Source Code Quality Measure score is computed as the sum of its quality measure elements counted across an entire application.

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

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

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

and

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

Functional Density of Weaknesses

In order to compare quality results among different applications, the Automated Source Code Quality Measures can be normalized by size to create a density measure. There are several size measures with which the density of quality violations can be normalized, such as lines of code and Function Points. These size measures, if properly standardized, can be used for creating a density measure for use in benchmarking the quality of applications. OMG's Automated Function Points (AFP) measure (ISO, 2019) offers an automatable size measure that, as an OMG Supported Specification, is standardized. AFP was adapted from the International Function Point User Group's (IFPUG) counting guidelines, and is commercially supported. Although other size measures can be used to evaluate the density of security violations, the following density measure for quality violations is derived from OMG supported specifications for Automated Function Points and the Automated Source Code Security Measure. Thus, the functional density of Security violations is a simple division expressed as follows.

ASCxM-density = ASCxM / AFP
where x = a software quality characteristic (R, S, PE, M)

Additional Derived Measures

There are many additional weighting schemes that can be applied to the Automated Source Code Quality Measures or to the quality measure elements that composing them. Table 5 presents several weighted measure candidates and their potential uses. However, these weighting schemes are not derived from any existing standards and are therefore not normative.

Table 5. Weighting Schemes for Automated Source Code Quality Measures

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

Appendix A: Consortium for Information & Software Quality (CISQ)

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

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

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

An international team of experts drawn from CISQ's 24 original companies formed into working groups to define CISQ measures. Weaknesses that had a high probability of causing reliability, security, performance efficiency, or maintainability problems were selected for inclusion in the four measures. The original CISQ members included IT departments in Fortune 200 companies, system integrators/ outsourcers, and vendors that provide quality-related products and services to the IT market. The experts met several times per year for two years in the US, France, and India to develop a broad list of candidate weaknesses. This list was pared down to a set of weaknesses they believed had to be remediated to avoid serious operational or cost problems. These weaknesses became the foundation of the original specifications of the automated source code measures for Reliability, Security, Performance Efficiency, and Maintainability.

Appendix B: Common Weakness Enumeration (CWE)

The Common Weakness Enumeration (CWE) repository (http://cwe.mitre.org/) maintained by The MITRE Corporation is a collection of over 800 weaknesses in software architecture and source code that malicious actors have used to gain unauthorized entry into systems or to cause malicious actions. The CWE is a widely used industry source (http://cwe.mitre.org/community/citations.html) that provides a foundation for the ITU-T X.1524 and ISO/IEC standard, in addition to 2 ISO/IEC technical reports:

- SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY Cybersecurity information exchange – Vulnerability/state exchange - Common weakness enumeration (CWE)
- ISO/IEC 29147:2014 Information Technology -- Security Techniques -- Vulnerability Disclosure"
- ISO/IEC TR 24772:2013 Information technology -- Programming languages -- Guidance to avoiding vulnerabilities in programming languages through language selection and use
- ISO/IEC Technical Report is ISO/IEC TR 20004:2012 Information Technology -- Security Techniques -- Refining Software Vulnerability Analysis under ISO/IEC 15408 and ISO/IEC 18045

The CWE/SANS Institute Top 25 Most Dangerous Software Errors is a list of the 25 most widespread and frequently exploited security weaknesses in the CWE repository. The previous version of the CISQ Automated Source Code Security Measure (ASCSM) was based on 22 of the CWE/SANS Top 25 that could be detected and counted in source code. In this revision, the number of security weaknesses is being expanded beyond the CWE/SANS Top 25 since there are other weaknesses severe enough to be incorporated in the CISQ measure. In addition, many CWEs also cause reliability problems and are therefore included in the CISQ Reliability measure. Wherever a CWE is included in any of the 4 CISQ structural quality measures, its CWE identifier will be noted.

Since the CWE is recognized as the primary industry repository of security weaknesses, it is supported by the majority of vendors providing tools and technology in the software security domain (http://cwe.mitre.org/compatible/compatible.html), such as Coverity, HP Fortify, Klockwork, IBM, CAST, Veracode, and others. These vendors already have capabilities for detecting many of the CWEs. Industry experts who developed the CWE purposely worded the CWEs to be language and application agnostic in order to allow vendors to develop detectors specific to a wide range of languages and application types beyond the scope that could be covered in the CWE. Since some of the CWEs may not be relevant in some languages, the reduced opportunity for anti-patterns in those cases will be reflected in the scores.

Appendix C: Relationship of the CISQ Structural Quality Measures to ISO 25000 Series Standards (SQuaRE)

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

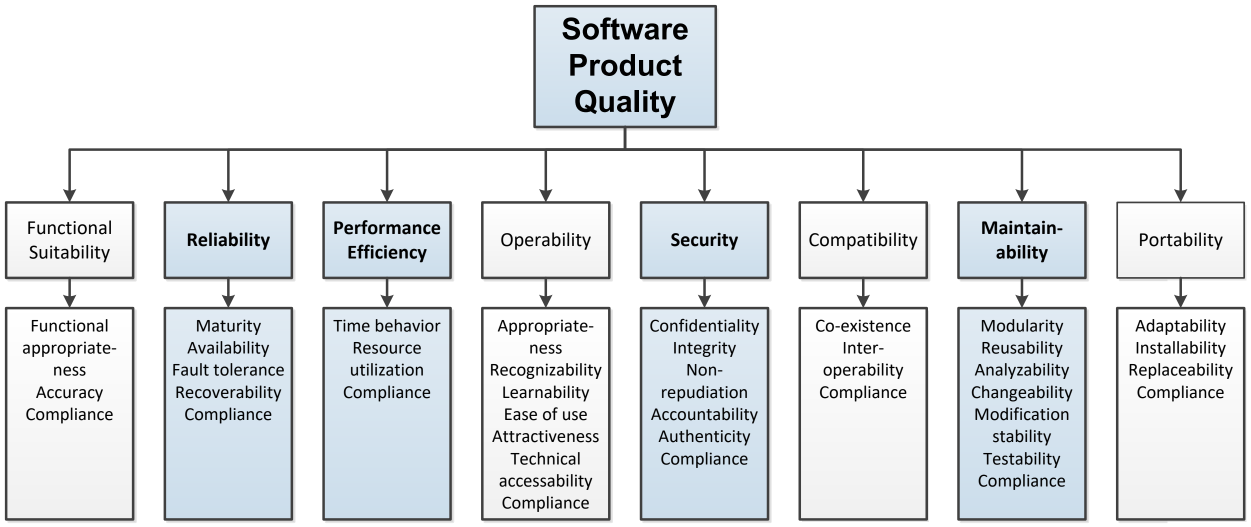
Software Product Quality
Functional Suitability
Reliability
Performance Efficiency
Operability
Security
Compatibility
Maintain-ability
Portability
Functional appropriate-ness Accuracy Compliance
Maturity Availability Fault tolerance Recoverability Compliance
Time behavior Resource utilization Compliance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

Figure 1. Software Quality Characteristics from ISO/IEC 25010 with CISQ measure areas highlighted.

ISO/IEC 25023 establishes a framework of software quality characteristic measures wherein each quality sub-characteristic consists of a collection of quality attributes that can be quantified as quality measure elements. A quality measure element quantifies a unitary measurable attribute of software, such as the violation of a quality rule. Figure 2 presents an example of the ISO/IEC 25023 quality measurement framework using a partial decomposition for the Automated Source Code Security Measure.

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

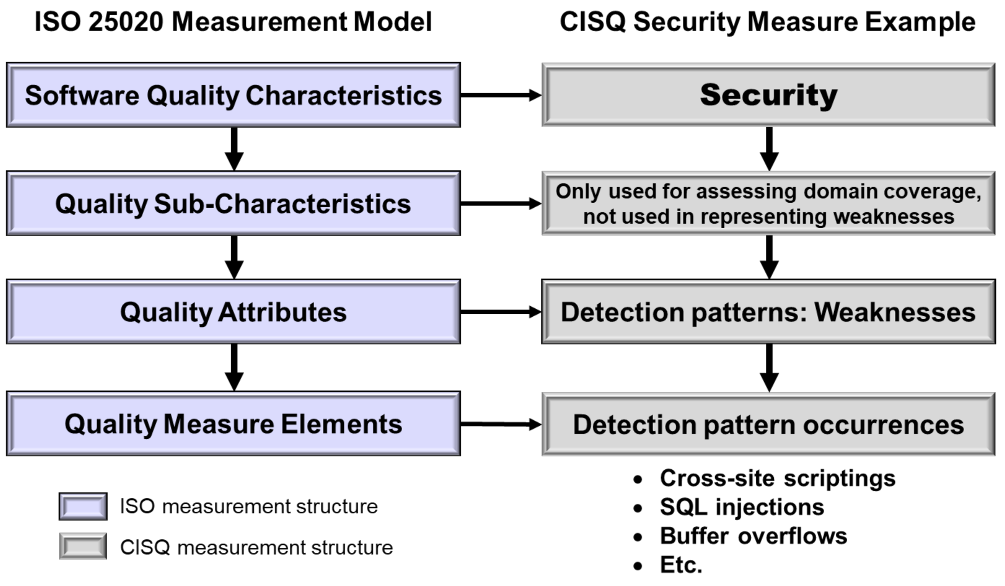
ISO 25020 Measurement Model
CISQ Security Measure Example
Software Quality Characteristics
Security
Quality Sub-Characteristics
Only used for assessing domain coverage, not used in representing weaknesses
Quality Attributes
Detection patterns: Weaknesses
Quality Measure Elements
Detection pattern occurrences
• Cross-site scriptings
• SQL injections
• Buffer overflows
• Etc.
ISO measurement structure
CISQ measurement structure

Figure 2. ISO/IEC 25020 Framework for Software Quality Characteristics Measurement

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