List of Weaknesses Included in the CISQ Automated Source Code Reliability Measure

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 a table. This document lists the weaknesses in the Reliability measure. 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 table 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 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 the table. 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: 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.

Master list of quality measure weaknesses: https://www.it-cisq.org/coding-rules/index.htm
Master list PDF: https://www.it-cisq.org/pdf/cisq-weaknesses-in-ascqm.pdf