Date: June 11, 2015

Automated Source Code Maintainability Measure

Beta 2

OMG Document Number: **ptc/2015-06-08**

Standard document URL:
http://www.omg.org/spec/ASCMM/20150608/AutomatedSourceMaintainabilityMeasure

Associated files: Normative:

http://www.omg.org/spec/ASCMM/20141211/AutomatedSourceCodeCISQTop20MaintainabilityMeasureSPMS.xmi

http://www.omg.org/spec/ASCMM/20141211/AutomatedSourceCodeCISQTop20MaintainabilityMeasureSMM.smm

---

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group (OMG) specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Consortium for IT Software Quality and it parent, Object Management Group, Inc. (OMG), a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. CISQ AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL CISDQ, THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE,

INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.287-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.287-19 or as specified in 48 C.F.R. 287-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 250 First Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

IMM®, MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , MOF™ , OMG Interface Definition Language (IDL)™ , and OMG SysML™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http:// www.omg.org*, under Documents, Report a Bug/Issue (*http://www.omg.org/report_issue.htm*).

Table of Contents

Table of Contents 4
Preface 6
About the Object Management Group 6
Issues 7
1. Scope 8
1.1 Purpose 8
1.2 CISQ Background 8
1.3 Overview of Software Quality Characteristic Measurement 8
1.4 Development of the Automated Source Code Maintainability Measure 9
1.5 Structure of the Automated Source Code Maintainability Measure 10
1.6 Using and Improving This Measure 11
2. Conformance 13
3. Normative References 13
3.1 Normative 13
4. Terms and Definitions 14
5. Symbols (and Abbreviated Terms) 15
6. Additional Information (Informative) 16
6.1 Software Product Inputs 16
6.2 Input Values for Thresholds in Measure Elements 16
6.3 Automated Source Code Maintainability Measure Elements 17
7. IPMSS Representation of the Quality Measure Elements (Normative) 24
7.1 Introduction 24
7.2 Category definition of CISQ Maintainability 27
7.3 Pattern definition of ASCMM-MNT-1: Control Flow Transfer Control Element outside Switch Block 27
7.4 Pattern definition of ASCMM-MNT-2: Class Element Excessive Inheritance of Class Elements with Concrete Implementation 28
7.5 Pattern definition of ASCMM-MNT-3: Storable and Member Data Element Initialization with Hard-Coded Literals 29
7.6 Pattern definition of ASCMM-MNT-4: Callable and Method Control Element Number of Outward Calls 30
7.7 Pattern definition of ASCMM-MNT-5: Loop Value Update within the Loop 32
7.8 Pattern definition of ASCMM-MNT-6: Commented-out Code Element Excessive Volume 33
7.9 Pattern definition of ASCMM-MNT-7: Inter-Module Dependency Cycles 34
7.10 Pattern definition of ASCMM-MNT-8: Source Element Excessive Size 35
7.11 Pattern definition of ASCMM-MNT-9: Horizontal Layer Excessive Number 36
7.12 Pattern definition of ASCMM-MNT-10: Named Callable and Method Control Element Multi-Layer Span 37

7.13 Pattern definition of ASCMM-MNT-11: Callable and Method Control Element Excessive Cyclomatic Complexity Value 38
7.14 Pattern definition of ASCMM-MNT-12: Named Callable and Method Control Element with Layer-skipping Call 39
7.15 Pattern definition of ASCMM-MNT-13: Callable and Method Control Element Excessive Number of Parameters 40
7.16 Pattern definition of ASCMM-MNT-14: Callable and Method Control Element Excessive Number of Control Elements involving Data Element from Data Manager or File Resource 42
7.17 Pattern definition of ASCMM-MNT-15: Public Member Element 43
7.18 Pattern definition of ASCMM-MNT-16: Method Control Element Usage of Member Element from other Class Element 44
7.19 Pattern definition of ASCMM-MNT-17: Class Element Excessive Inheritance Level 45
7.20 Pattern definition of ASCMM-MNT-18: Class Element Excessive Number of Children 46
7.21 Pattern definition of ASCMM-MNT-19: Named Callable and Method Control Element Excessive Similarity 47
7.22 Pattern definition of ASCMM-MNT-20: Unreachable Named Callable or Method Control Element 48
8. Calculation of the CISQ Automated Source Code Maintainability Measure and Functional Density (Normative) 50
8.1 Calculation of the Base Measure 50
8.2 Functional Density of Maintainability Violations 50
9. Alternative Weighted Measures and Uses (Informative) 51
9.1 Additional Derived Measures 51
10. References (Informative) 52
Appendix A: CISQ 53

Preface

About the Object Management Group

OMG

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

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

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Formal Specifications are available from this URL: *http://www.omg.org/spec*
Specifications are organized by the following categories:

Business Modeling Specifications Middleware Specifications

- CORBA/IIOP
- Data Distribution Services
- Specialized CORBA

IDL/Language Mapping Specifications

Modeling and Metadata Specifications

- UML, MOF, CWM, XMI
- UML Profile

Modernization Specifications

Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

- CORBAServices

- CORBAFacilities

CORBA Embedded Intelligence Specifications

CORBA Security Specifications

OMG Domain Specifications

Signal and Image Processing Specifications

All of OMG‟s formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
109 Highland Avenue
Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO/IEC standards. Please consult http://www.iso.org

Issues

The reader is encouraged to report and technical or editing issues/problems with this specification to http://www.omg.org

1. Scope

1.1 Purpose

The purpose of this specification is to establish a standard measure of Maintainability based on detecting violations of good architectural and coding practices that could result in unreliable operation such as outages, data corruption, and lengthy recovery from system failures. Establishing a standard for this measure is important because such measures are being used in outsourcing and system development contracts without having an approved international standard to reference. They are also critical to other software-intensive OMG initiatives such as The Internet of Things. The Consortium for IT Software Quality (CISQ) was formed as a special interest group of OMG to create specifications for automating standard measures of software quality attributes and submit them to OMG for approval

1.2 CISQ Background

This specification defines a method for automating the measurement of Maintainability from violations of architectural and coding practice that affect an application's understandability and ease of change. The violations included in the CISQ measure were selected from a large set of candidate violations related to Maintainability issues. The final set of violations were chosen through a voting process among CISQ member organizations that resulted in a limited set of violations that member organizations believed were sufficiently severe that they had to be remediated. This process will be described more fully in a subsequent sub-clause.

1.3 Overview of Software Quality Characteristic Measurement

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

Violations of Good Architectural and Coding Practice—The most recent advance in measuring the structural quality of software is based on the analysis and measurement of violations of good architectural and coding practice that can be detected by statically analyzing the source code. The CWE/SANS 25 and OWASP Top Ten lists of security weaknesses are examples of this approach. These lists are drawn from the Common Weakness Enumeration (CWE) repository maintained by MITRE Corporation. CWE contains

descriptions of over 800 weaknesses that represent violations of good architectural and coding practice in software that can be exploited to gain unauthorized entry into a system. The Software Assurance community has been a leader in this area of measurement by championing the detection of code weaknesses as a way of improving one aspect of software quality—software security.

Unfortunately there are no equivalent repositories of weaknesses for Reliability, Performance Efficiency, or Maintainability. Knowledge of these weaknesses is spread across software engineering textbooks, expert blogs, and information sharing sites such as github. The CISQ measure for Maintainability can fill the void for a consensus body of knowledge about the most egregious Maintainability problems that should be detected and remediated in source code. Currently, no standards or guidelines have been developed for calculating component or application-level Maintainability measures that aggregate weaknesses detected through static code analysis into application-level Maintainability measures. CISQ will be providing recommendations for these aggregation and scaling techniques. However, these techniques are not part of this standard since different measurement objectives are best served by different scoring techniques.

Using violations of good architectural and coding practices in software quality metrics presents several challenges for establishing baselines. Growth in the number of unique violations to be detected could continually raise the bar for measuring quality, reducing the validity of baseline comparisons. Further, different vendors will detect different sets of violations, making comparisons difficult across commercial software quality measurement offerings. One solution to this problem is to create a stable list of violations that are used for computing a baseline for each quality characteristic. The Automated Source Code Maintainability Measure was developed by a team of industry experts to form the basis for a stable baseline measure.

1.4 Development of the Automated Source Code Maintainability Measure

The original 24 CISQ member companies provided experts to working groups whose charter was to define CISQ measures. Violations of good architectural and coding practice that a high probability of causing Maintainability problems were selected by an international team of experts drawn from the 24 organizations that joined CISQ in 2010. These organizations 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 Maintainability weaknesses and then pare it down to a set they felt had to be remediated to avoid serious operational problems.

The work group began by defining Maintainability issues, quality rules for avoiding these issues, and measures based on counting violations of these rules. They developed lists of issues and quality rules by drawing information from company defect logs, their career experience in different environments, and industry sources such as books and blogs. In order to reduce the work group's initial list to a critical set of Maintainability violations, work group members individually evaluated the severity of each violation. High severity violations were judged to be those that must be fixed in a future release because of their operational risk or cost impact. The work group went through several rounds of eliminating lower severity

violations and re-rating the severity of remaining violations until a final list was established as the quality measure elements to be incorporated into this specification.

1.5 Structure of the Automated Source Code Maintainability Measure

ISO/IEC 25010 defines a quality characteristic as being composed from several quality sub-characteristics. This framework for software product quality is presented in Figure 1 for the eight quality characteristics presented in 25010. The quality characteristics and their sub-characteristics selected for source code measurement by CISQ are indicated in blue.

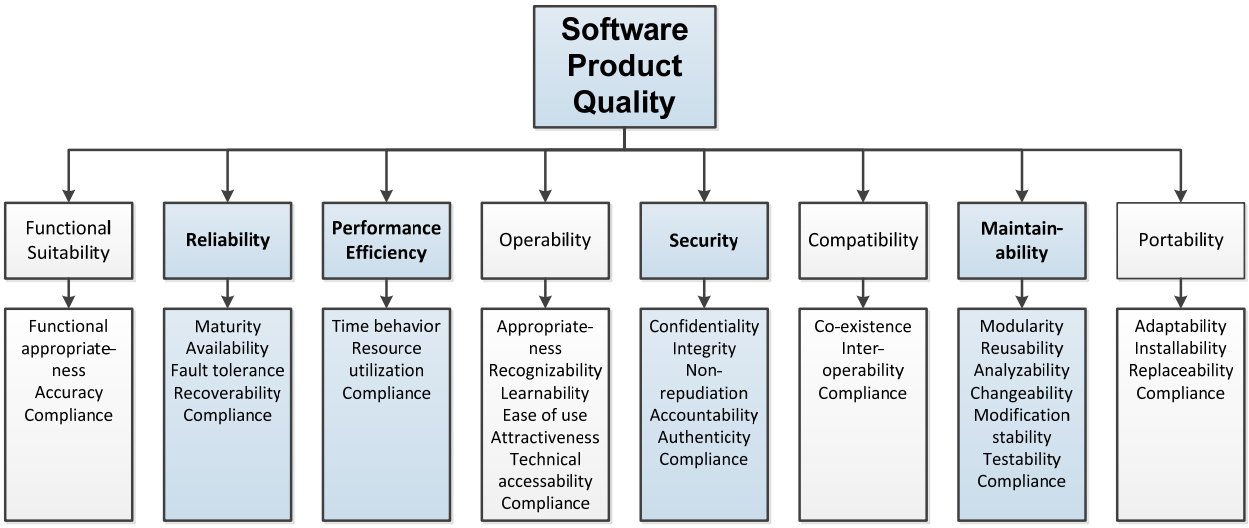

Figure 1. Software Quality Characteristics from ISO/IEC 25010 with CISQ focal areas highlighted.

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 Maintainability Measure.

The non-normative portion of this specification begins by listing the Maintainability issues that can plague software developed with poor architectural and coding practices. Quality rules written as architectural or coding practices are conventions that avoided the problem described in the Maintainability issue. These quality rules were then transformed into software quality measure elements by counting violations of these architectural and coding practices and conventions.

The normative portion of this specification represents each quality measure element developed from a Maintainability rule using the Structured Patterns Meta-model Standard (SPMS). The code-based elements in these patterns are represented using the Knowledge Discovery Meta-model (KDM). The calculation of the Automated Source Code Maintainability Measure from its quality measure elements is

then represented in the Structured Metrics Meta-model (SMM). This calculation is presented as the simple sum of quality measure elements without being adjusted by a weighting scheme.

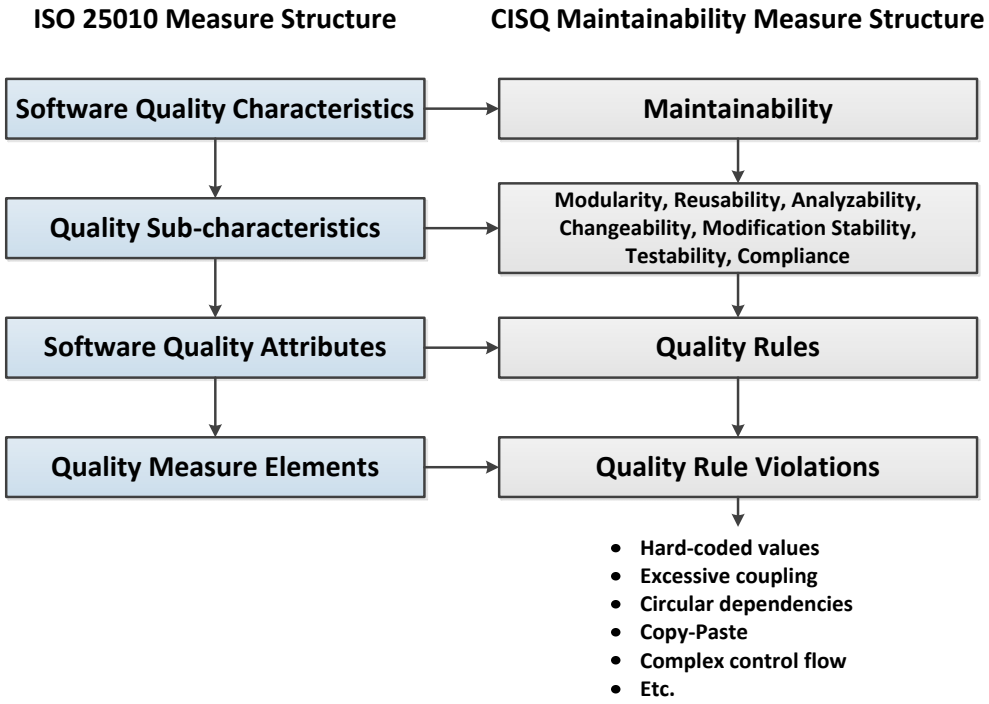

Figure 2. ISO/IEC 25010 Framework for Software Quality Characteristics Measurement

There are several weighting schemes that can be applied in aggregating violation counts into structural quality measures. The most effective weighting often depends on the measure's use such as assessing operational risk or estimating maintenance costs. The quality measure elements included in this specification were considered to be severe violations of secure architectural and coding practices that would need to be remediated. Therefore, weightings based on severity would add little useful information to the measure since the variance among weights would be small. In order to support benchmarking among applications, this specification includes a measure of the violation density. This measure is created by dividing the total number of violations detected by a count of Automated Function Points (Object Management Group, 2014).

1.6 Using and Improving This Measure

The Automated Source Code Maintainability Measure is a correlated measure rather than an absolute measure. That is, since it does not measure all possible Maintainability-related weaknesses it does not provide an absolute measure of Maintainability. However, since it includes counts of what industry experts considered high severity Maintainability weaknesses, it provides a strong indicator of Maintainability that will be highly correlated with the absolute Maintainability of a software system and with the probability that it can experience outages, data corruption, and related problems.

Since the impact and frequency of specific violations in the Automated Source Code Maintainability Measure could change over time, this approach allows specific violations to be included, excluded, amplified, or diminished over time in order to support the most effective benchmarking, diagnostic, and predictive use. This specification will be adjusted through controlled OMG specification revision processes to reflect changes in Maintainability engineering while retaining the ability to compare baselines. Vendors of static analysis and measurement technology can compute this standard baseline measure, as well as their own extended measures that include other Maintainability weaknesses not included as measure elements in this specification.

2. Conformance

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

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

3. Normative References

3.1 Normative

The following normative documents contain provisions, which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of any of these publications do not apply.

- Structured Patterns Meta-model Standard, admtf/14-02-01
- Knowledge Discovery Meta-model, version 1.3 (KDM), formal/2011-08-04
- Structured Metrics Meta-model, version 1.0 (SMM), formal/2012-01-05
- MOF/XMI Mapping, version 2.4.1 (XMI), formal/2011-08-09
- Automated Function Points (AFP), formal/2014-01-03
- ISO/IEC 25010 Systems and software engineering – System and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models

4. Terms and Definitions

Automated Function Points—a specification for automating the counting of Function Points that mirrors as closely as possible the counting guidelines of the International Function Point User Group. (OMG, formal 2014-01-03)

Cyclomatic Complexity—A measure of control flow complexity developed by Thomas McCabe based on a graph-theoretic analysis that reduces the control flow of a computer program to a set of edges, vertices, and their attributes that can be quantified. (McCabe, 1976)

Internal Software Quality—the degree to which a set of static attributes of a software product satisfy stated and implied needs for the software product to be used under specified conditions. This will be referred to as software structural quality, or simply structural quality in this specification. (ISO/IEC 25010)

Maintainability—degree of effectiveness or efficiency with which a product or system can be modified by the intended maintainers. (ISO/IEC 25010)

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

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

Software Maintainability—degree of effectiveness or efficiency with which a software product or system can be modified by the intended maintainers. (ISO/IEC 25010)

Software Maintainability Measure Element—a measure defined in terms of a quality attribute of software that affects Its maintainabiilty and the measurement method for quantifying it, including optionally the transformation by a mathematical function. (adapted from ISO/IEC 25023)

Software Product Quality Model—a model that categorizes product quality properties into eight characteristics (functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability). Each characteristic is composed of a set of related sub-characteristics. (ISO/IEC 25010)

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

Software Quality Attribute—an inherent property or characteristic of software that can be distinguished quantitatively or qualitatively by human or automated means. (derived from ISO/IEC 25010)

Software Quality Characteristic—a category of software quality attributes that bears on software quality. (ISO/IEC 25010)

Software Quality Characteristic Measure—a software quality measure derived from measuring the attributes related to a specific software quality characteristic.

Software Quality Issue—architectural or coding practices that are known to cause problems in software development, maintenance, or operations and for which software quality rules can be defined that help avoid problems created by the issue.

Software Quality Measure—a measure that is defined as a measurement function of two or more values of software quality measure elements. (ISO/IEC 25010)

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

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

Software Quality Property—measurable component of software quality. (derived from ISO/IEC 25010)

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

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

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

Violation—a pattern or structure in the code that is inconsistent with good architectural and coding practices and can lead to problems in operation or maintenance.

5. Symbols (and Abbreviated Terms)

CISQ – Consortium for IT Software Quality

KDM – Knowledge Discovery Meta-model

SPMS – Structured Patterns Meta-model Standard

SMM – Structured Metrics Meta-model

6. Additional Information (Informative)

6.1 Software Product Inputs

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

- The entire source code for the application being analyzed
- All materials and information required to prepare the application for production
- A description of the architecture and layer boundaries of the application, including an assignment of modules to layers

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

6.2 Input Values for Thresholds in Measure Elements

Several of the weaknesses in the Automated Source Code Maintainability measure detect violations of good architectural or coding practice based on threshold values for a construct being exceeded. Table 1 lists the default threshold value used in specifying this measure. In using this measure, threshold values can be adjusted to different levels. However, when the threshold values are adjusted the results cannot be compared or benchmarked to data from other analyses that used the default values. In such cases it may be good to compute values for both the default and adjusted values.

Table 1. Input Values for Thresholds in Measure Elements

KDM Element in a Measure Element	Threshold Value for a Measure Element
<ArchitectureModel> defining the application's architectural blueprint. <NumberOfHorizontalLayerThresholdMinimalValue> minimal value <NumberOfHorizontalLayerThresholdMaximalValue> maximal value.	The default value for <NumberOfHorizontalLayerThresholdMinimalValue> is 4. The default value for <NumberOfHorizontalLayerThresholdMaximalValue> is 8.
<NumberOfInheritenceLevelsThresholdValue> maximum value of number of parent class units	The default value for <NumberOfInheritenceLevelsThresholdValue> is 7.
<NumberOfChildrenThresholdValue> maximum value of number of child classes	The default value for <NumberOfChildrenThresholdValue> is 10.

<NumberOfConcreteClasseInheritencesThresholdValue> maximum value of number of inheritence of concrete classes	Default value for <NumberOfConcreteClasseInheritencesThresholdValue> is 1.
<NumberOfOutwardReferencesThresholdValue> maximum value of number of references to other objects	Default value for <NumberOfOutwardReferencesThresholdValue> threshold value is 5.
<PercentageOfCommentedOutInstructionsThresholdValue> maximum value of percentage of instructions that are in comments	Default value for <PercentageOfCommentedOutInstructionsThresholdValue> threshold value is 2%.
<NumberOfLinesOfCodeThresholdValue> maximum value of number of lines of code	The default value for <NumberOfLinesOfCodeThresholdValue> is 1000.
<CyclomaticComplexityThresholdValue> maximum value of distinct path through the control element	Default value for <CyclomaticComplexityThreshold> is 20.
<NumberOfDataOperationsThresholdValue> maximum value of data operation in control element	Default value for <NumberOfDataOperationsThresholdValue> is 7.
<ParameterNumberThreshold> maximum value of parameters in signature	Default value for the <ParameterNumberThreshold> is 7.

6.3 Automated Source Code Maintainability Measure Elements

The violations of good architectural and coding practice incorporated into the Automated Source Code Maintainability Measure are listed in Table 2. The pattern label appears in the first column. The consequences that can be caused by the pattern are listed in column 2. A coding or architectural rule that helps avoid the pattern are listed as objectives in column 3. Finally, a textual description of the pattern that constitutes the attribute underlying the measure element is provided in column 4.

Table 2. Maintainability Patterns, Consequences, Objectives, and Maintainability Measure Elements

Maintainability Pattern	Consequence	Objective	Measure Element
ASCMM-MNT-1: Control Flow Transfer Control Element outside Switch Block	Software that does not follow the principles of structured programming degrades comprehensibility	Avoid the unconditional transfer of control flow outside of switch structures	Number of instances where an unconditional transfer of control is located outside the branching based on the value of a storable element
ASCMM-MNT-2: Class Element Excessive Inheritance of Class Elements with	Software that does not follow the principles of reuse requires more maintenance effort in	Avoid the multiple inheritance of classes with concrete implementations	Number of instances where the number of inheritances of concrete classes of a

Concrete Implementation	order to propagate changes to all instances of duplicated code		class element is considered too large, based on exceeding a threshold value. Default value for the number of concrete class Inheritances is 1.
ASCMM-MNT-3: Storable and Member Data Element Initialization with Hard-Coded Literals	Software featuring hard-coded pieces of information within its own code reduces adaptability	Avoid hard-coded non-trivial values in the code	Number of instances where a literal value element is used to intialize a storable data element or member data element via a 'Write' action; exceptions are simple integers and static constant storable or member data elements
ASCMM-MNT-4: Callable and Method Control Element Number of Outward Calls	Software that does not follow the principles of modularity causes excessive propagation of modification impacts	Avoid overly complex outward dependencies	Number of instances where a named callable control element or method control element has a Fan-Out value that is too large, that is, its number of references to other objects within the application exceeds a threshold value (the application determines the scope of the search for the referenced objects). Default threshold value for the number of references to other objects within the application is 5
ASCMM-MNT-5: Loop Value Update within the Loop	Software that does not follow the principles of modularity causes excessive propagation of modification impacts	Avoid overly complex behaviors of loop indices	Number of instances where a value of a local storable data element used in the condition of the loop control flow is updated within the

			'Write' action located in the loop body block
ASCMM-MNT-6: Commented-out Code Element Excessive Volume	Software that does contain commented-out code that can mistakenly be considered as active code and that can hide a lack of comments causes excessive modification effort	Avoid code blocks found in comments	Number of instances where a named callable control element or method control element contains too many commented-out code items compared to a threshold that is based on the percentage of instructions in the callable control element or method control element that are in comments. Default threshold value for the percentage of commented out instructions is 2%.
ASCMM-MNT-7: Inter-Module Dependency Cycles	Software that does not follow the principles of modularity causes excessive propagation of modification impacts	Avoid circular dependencies between modules	Number of instances where a module has references that cycle back to itself via the module callable or data relations cycle (for example, with JAVA this pattern means cycles between packages)
ASCMM-MNT-8: Source Element Excessive Size	Software that does not follow the principles of modularity causes excessive propagation of modification impacts	Avoid over-sizing of software elements	Number of instances where a file has too many lines of code based on a threshold value. The default threshold value for number of lines of code is 1000
ASCMM-MNT-9: Horizontal Layer Excessive Number	Software that does not follow the principles of layered architectures (such as strict partitioning and strict	Avoid the existence of too many or too few horizontal layers	Number of instances where a model of the architectural layers of an application contains too may or too few

	call hierarchy) decreases comprehensibility as well as simplicity to evolve the code		horizontal layers (excluding the vertical utility layers) based on comparison to a threshold value. The default value for the minimal number of horizontal layers is 4, and the default value for maximal number of horizontal layers is 8.
ASCMM-MNT-10: Named Callable and Method Control Element Multi-Layer Span	Software that does not follow the principles of layered architectures (such as strict partitioning and strict call hierarchy) decreases comprehensibility as well as simplicity to evolve the code	Avoid unclear allocation of software elements to a single architectural layer	Number of instances where a callable or method control element is part of two architectural layers
ASCMM-MNT-11: Callable and Method Control Element Excessive Cyclomatic Complexity Value	Software that does not follow the principles of structured programming degrades comprehensibility	Avoid overly complex control flow	Number of instances where a named callable control element or method control element has a control flow with a Cyclomatic Complexity number that exceeds a threshold value. Default threshold value for Cyclomatic Complexity is 20
ASCMM-MNT-12: Named Callable and Method Control Element with Layer-skipping Call	Software that does not follow the principles of layered architectures (such as strict partitioning and strict call hierarchy) decreases comprehensibility as well as simplicity to evolve the code	Avoid breaches of layered architecture principles due to layer-skipping references	Number of instances where a named callable or method control element from a higher horizontal layer directly calls a named callable or method control element in a lower horizontal layer that is not adjacent to the upper layer making

			the call, as defined in a model of the application's architectural layers (this excludes the vertical utility layers that can be referenced from any horizontal layer)
ASCMM-MNT-13: Callable and Method Control Element Excessive Number of Parameters	Software that does not cap the number of parameters degrades comprehensibility	Avoid over-parameterization	Number of instances where a named callable control element or method control element has a number of parameters in its signature that exceeds a threshold value. Default threshold value for the number of parameters is 7.
ASCMM-MNT-14: Callable and Method Control Element Excessive Number of Control Elements involving Data Element from Data Manager or File Resource	Software that does not cap the number of data operations degrades comprehensibility by requiring the understanding of too many external data structures	Avoid the existence of control elements with too many data operations	Number of instances where a named callable control element or method control element has a number of operations involving a data manager or a file resource that exceeds a threshold value. Default threshold value for the number of data operations is 7
ASCMM-MNT-15: Public Member Element	Software that does not follow the principles of data encapsulation incurs the risk of data corruption	Avoid openly accessible data elements	Number of instances where a storable data element or member data element is declared as public through a Create action
ASCMM-MNT-16: Method Control Element Usage of	Software that does not follow the principles of data encapsulation	Avoid direct access to data elements of another entity	Number of instances where a method control element from a class element accesses

Member Element from other Class Element	incurs the risk of data corruption		a member element from another class element
ASCMM-MNT-17: Class Element Excessive Inheritance Level	Software that does not follow the principles of reuse requires more maintenance effort in order to propagate changes to all instances of duplicated code	Avoid overly complex object-oriented inheritance capabilities when dealing with the number of levels of inheritance of classes	Number of instances where the inheritance level of a class element (that is, the number of parent class units) exceeds a threshold value. The default threshold value for number of inheritance levels is 7
ASCMM-MNT-18: Class Element Excessive Number of Children	Software that does not follow the principles of reuse requires more maintenance effort in order to propagate changes to all instances of duplicated code	Avoid overly complex object-oriented inheritance capabilities when dealing with the number of direct children of classes	Number of instances where the number of children of a class element (that is, its number of child classes) exceeds a threshold value. The default threshold value for number of children of a class element is 10
ASCMM-MNT-19: Named Callable and Method Control Element Excessive Similarity	Software that does not follow the principles of reuse requires more maintenance effort in order to propagate changes to all instances of duplicated code	Avoid software element redundancy	Number of instances where a named callable control element or method control element contains multiple computational objects that are identical to computational objects in another named callable or method control element in the application (the application determines the scope of the search for the second code item)
ASCMM-MNT-20: Unreachable Named Callable or Method Control Element	Software that does not follow the principles of reuse requires more maintenance effort in	Avoid inactive code blocks that can mistakenly be considered as active	Number of instances where a named callable control element or method

	order to propagate changes to all instances of duplicated code	and that can hide the active code in noise	control element is unreferenced by any other code item in the application (the application determines the scope of the search for code items that could call a callable or method control element)

7. IPMSS Representation of the Quality Measure Elements (Normative)

7.1 Introduction

This chapter displays in a human readable format the content of the machine readable XMI format file attached to the current specification.
The content of the machine readable XMI format file is the representations of the CISQ Quality Measure Elements

- according to the Implementation Patterns Metamodel for Software Systems (IPMSS)
- and relating to the Knowledge Discovery Meta-Model (KDM) within their description as frequently as possible, so as to be as generic as possible yet as accurate as possible.

IPMSS

More specifically, the machine readable XMI format file attached to the current specification uses the IPMSS Definitions Classes:

- PatternDefinition (ipmss:PatternDefinition): the pattern specification. In the context of this document, each CISQ Quality Measure Element is basically the count of occurrences of the described patterns.
- Role (ipmss:Role): “A pattern is informally defined as a set of relationships between a set of entities. Roles describe the set of entities within a pattern, between which those relationships will be described. As such the Role is a required association in a PatternDefinition. […]. Semantically, a Role is a 'slot' that is required to be fulfilled for an instance of its parent PatternDefinition to exist.”
- PatternSection (ipmss:PatternSection): “A PatternSection is a free-form prose textual description of a portion of a PatternDefinition.” In the context of this document, there are 6 different PatternSections in use:
    - “Descriptor” to provide pattern signature, a visible interface of the pattern,
    - “Measure Element” to provide a human readable explanation of the measure,
    - “Description” to provide a human readable explanation of the pattern that is sought after, identifying “Roles” and KDM modeling information,
    - “Objective” to provide a human readable explanation of the intent to get rid of the occurrences of the pattern that is sought after,
    - “Consequence” to provide a human readable explanation of the issue the detection of the pattern is designed to solve,
    - “Input” to provide a human readable of the parameters that are needed to fine-tune the behavior of the pattern detection (e.g.: the target application architectural blueprint to comply with)

- "Comment" to provide some additional information (until now, used to inform about situations where the same measure element is useful for another one of the categories)

As well as some of the IPMSS Relationships Classes:

- MemberOf (ipmss:MemberOf): "An InterpatternRelationship specialized to indicate inclusion in a Category"
- Category (ipmss:Category): "A Category is a simple grouping element for gathering related PatternDefinitions into clusters." In the context of this document, the IPMSS Categories are used to represent the 4 CISQ Quality Characteristics:
  - "CISQ Reliability",
  - "CISQ Security",
  - "CISQ Performance Efficiency",
  - And "CISQ Maintainability".

KDM

More specifically, the machine readable XMI format file attached to the current specification uses KDM entities in the "Description" section of the pattern definitions.
Descriptions try to remain as generic yet accurate as possible so that the pattern can be applicable and applied to as many situations as possible: different technologies, different programming languages …
This means:

1. The descriptions include information such as (code:MethodUnit), (action:Reads), (platform:ManagesResource), … to identify the KDM entities the pattern definition involves
2. The descriptions only detail the salient aspects of the pattern as the specifics can be technology- or language-dependant

Although a fair knowledge of the KDM is highly recommended to read this chapter, here follows a "KDM primer" to help getting started via a translation table between layman wording (simple although not fully accurate) and KDM wording used in the current specification.

Layman wording	**KDM wording**
function, method, procedure, stored procedure, sub, routine, etc.	named callable control element (code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored') or method control element (code:MethodUnit)
variable, field, member, etc.	storable data element (code:StorableUnit) or member data element (code:MemberUnit)
class	class element (code:StorableUnit with code:DataType code:ClassUnit)
interface	interface element (code:StorableUnit of code:DataType code:InterfaceUnit)
method	method element (code:MethodUnit)
field, member	member element (code:MemberUnit)

SQL stored procedures	stored callable control elements (code:CallableUnit with code:CallableKind 'stored') in a data manager resource (platform:DataManager)
return code value	value (code:Value) of the return parameter (code:ParameterUnit of code:ParameterKind 'return')
exception	exception parameter (code:ParameterUnit with code:ParameterKind 'exception')
user input data flow	an external value is entered is entered into the application through the 'ReadsUI' user interface ReadsUI action (ui:ReadsUI), transformed throughout the application along the 'TransformationSequence' sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), and ultimately used as
execution path	execution path (action:BlockUnit composed of action:ActionElements with action:CallableRelations to code:ControlElements)
libraries, etc.	deployed component (platform:DeployedComponent)
RDBMS	data manager resource (platform:DataManager)
loop body	loop body block (action:BlockUnit starting as the action:TrueFlow of the loop action:GuardedFlow and ending with an action:Flow back to the loop action:GuardedFlow)
loop condition	loop condition (action:BlockUnit used in the action:GuardedFlow)
singleton	class element (code:StorableUnit with code:DataType code:ClassUnit) that can be used only once in the 'to' assoction of a Create action (action:Creates)
checked	used by a check control element (code:ControlElement containing action:ActionElement with a kind from micro KDM list of comparison actions)

Reading guide

Each numbered sub-clause of this chapter

- Sub-clause 7.2 represents the IPMSS Category covered by the current specification
- Starting with number 7.3, each sub-clause represents a new IPMSS PatternDefinition member of this IPMSS Category

IPMSS PatternDefinition sub-sections are:

- Pattern category: the “ipmss:Category” category the pattern is related to through a “ipmss:MemberOf” relationship.
- Pattern sections: the list of "ipmss:PatternSection" sections from the pattern:
    - “Descriptor”,
    - “Description”,
    - “Objective”,
    - “Consequence”,
    - and, when applicable,
        - “Input”,
        - “Comment”.
- Pattern roles: the list of “ipmss:Role” roles used in the “Descriptor”, and “Description” sections above.

In the following sub-clauses,

- Data between square brackets (e.g.: [key CISQ Maintainability]) identifies “xmi:id” that are unique and used to reference entities. They are machine-generated to ensure unicity.
- Data between paranthesis (e.g.: (code:MethodUnit)) identifies KDM modeling information.
- Data between angle brackets (e.g.: <ControlElement>) identifies IPMSS Roles in Description and Input sections.

7.2 Category definition of CISQ Maintainability

[key ASCMM_Maintainability] CISQ Maintainability

7.3 Pattern definition of ASCMM-MNT-1: Control Flow Transfer Control Element outside Switch Block

Pattern Category

[key ASCMM-MNT-1-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-1-objective]
Avoid the unconditional transfer of control flow outside of switch structures

Consequence

[key ASCMM-MNT-1-consequence]
Software that does not follow the principles of structured programming degrades comprehensibility

Measure Element

[key ASCMM-MNT-1-measure-element]
Number of instances where an unconditional transfer of control is located outside the branching based on the value of a storable element

Description

[key ASCMM-MNT-1-description]
This pattern identifies situations where <ControlFlowJumpStatement> control flow (action:ActionElement with micro KDM kind such as 'Goto') unconditional transfer of control is located outside the <SwitchBranching> branching based on the value of a storable element (action:ActionElement with micro KDM kind 'Switch').

Descriptor

[key ASCMM-MNT-1-descriptor]
ASCMM-MNT-1(ControlFlowJumpStatement: controlFlowJumpStatement,SwitchBranching: switchBranching)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-1-roles-controlFlowJumpStatement] ControlFlowJumpStatement
[key ASCMM-MNT-1-roles-switchBranching] SwitchBranching

7.4 Pattern definition of ASCMM-MNT-2: Class Element Excessive Inheritance of Class Elements with Concrete Implementation

Pattern Category

[key ASCMM-MNT-2-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-2-objective]
Avoid the multiple inheritance of classes with concrete implementations

Consequence

[key ASCMM-MNT-2-consequence]
Software that does not follow the principles of reuse requires more maintenance effort in order to propagate changes to all instances of duplicated code

Measure Element

[key ASCMM-MNT-2-measure-element]
Number of instances where the number of inheritances of concrete classes of a class element is considered too large, based on exceeding a threshold value. Default value for the number of concrete class Inheritances is 1.

Description

[key ASCMM-MNT-2-description]
This pattern idendifies situations where the number of inheritence (code:Extends relation) of concrete classes (code:StorableUnit of code:DataType code:ClassUnit having code:MethodUnit with code:MethodKind different from 'abstract') of the <Class> class element (code:StorableUnit of code:DataType code:ClassUnit) is considered as too large, based on its <NumberOfConcreteClasseInheritences> number of inheritence of concrete classes which exceeds the <NumberOfConcreteClasseInheritencesThresholdValue> threshold value.
Default value for <NumberOfConcreteClasseInheritencesThresholdValue> is 1.

Descriptor

[key ASCMM-MNT-2-descriptor]
ASCMM-MNT-2(Class: class,NumberOfConcreteClasseInheritences: numberOfConcreteClasseInheritences, NumberOfConcreteClasseInheritencesThresholdValue: numberOfConcreteClasseInheritencesThresholdValue)

Variable input

[key ASCMM-MNT-2-input]
<NumberOfConcreteClasseInheritencesThresholdValue> maximum value

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-2-roles-class] Class
[key ASCMM-MNT-2-roles-numberOfConcreteClasseInheritences] NumberOfConcreteClasseInheritences
[key ASCMM-MNT-2-roles-numberOfConcreteClasseInheritencesThresholdValue] NumberOfConcreteClasseInheritencesThresholdValue

7.5 Pattern definition of ASCMM-MNT-3: Storable and Member Data Element Initialization with Hard-Coded Literals

Pattern Category

[key ASCMM-MNT-3-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-3-objective]
Avoid hard-coded non-trivial values in the code

Consequence

[key ASCMM-MNT-3-consequence]
Software featuring hard-coded pieces of information within its own code reduces adaptability

Measure Element

[key ASCMM-MNT-3-measure-element]
Number of instances where a literal value element is used to intialize a storable data element or member data element via a 'Write' action; exceptions are simple integers and static constant storable or member data elements

Description

[key ASCMM-MNT-3-description]
This pattern identifies situations where the <ValueElement> literal value element (code:Value) is used to intialize the storable data element (code:StorableUnit) or member data element (code:MemberUnit) via the <InitialisationStatement> Write action (action:Writes); exceptions are simple integers and static of constant storable or member data elements.

Descriptor

[key ASCMM-MNT-3-descriptor]
ASCMM-MNT-3(ValueElement: valueElement,InitialisationStatement: initialisationStatement)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-3-roles-valueElement] ValueElement
[key ASCMM-MNT-3-roles-initialisationStatement] InitialisationStatement

7.6 Pattern definition of ASCMM-MNT-4: Callable and Method Control Element Number of Outward Calls

Pattern Category

[key ASCMM-MNT-4-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-4-objective]
Avoid overly complex outward dependencies

Consequence

[key ASCMM-MNT-4-consequence]
Software that does not follow the principles of modularity causes excessive propagation of modification impacts

Measure Element

[key ASCMM-MNT-4-measure-element]
Number of instances where a named callable control element or method control element has a Fan-Out value that is too large, that is, its number of references to other objects within the application exceeds a threshold value (the application determines the scope of the search for the referenced objects).
Default threshold value for the number of references to other objects within the application is 5

Description

[key ASCMM-MNT-4-description]
This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored') or method control element (code:MethodUnit) has a Fan-Out value (number of code:Calls relations where it is used in the 'from' association) that is too large, based on its <NumberOfOutwardReferences> number of references to other objects within the <Application> application which exceeds the <NumberOfOutwardReferencesThresholdValue> threshold value; the <Application> application determines the scope of the search for the referenced objects.
Default value for <NumberOfOutwardReferencesThresholdValue> threshold value is 5.

Descriptor

[key ASCMM-MNT-4-descriptor]
ASCMM-MNT-4(ControlElement: controlElement,NumberOfOutwardReferences: numberOfOutwardReferences, NumberOfOutwardReferencesThresholdValue: numberOfOutwardReferencesThresholdValue, Application: application)

Variable input

[key ASCMM-MNT-4-input]
<NumberOfOutwardReferencesThresholdValue> maximum value

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-4-roles-controlElement] ControlElement
[key ASCMM-MNT-4-roles-numberOfOutwardReferences] NumberOfOutwardReferences

[key ASCMM-MNT-4-roles-numberOfOutwardReferencesThresholdValue]
NumberOfOutwardReferencesThresholdValue
[key ASCMM-MNT-4-roles-application] Application

7.7 Pattern definition of ASCMM-MNT-5: Loop Value Update within the Loop

Pattern Category

[key ASCMM-MNT-5-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-5-objective]
Avoid overly complex behaviors of loop indices

Consequence

[key ASCMM-MNT-5-consequence]
Software that does not follow the principles of modularity causes excessive propagation of modification impacts

Measure Element

[key ASCMM-MNT-5-measure-element]
Number of instances where a value of a local storable data element used in the condition of the loop control flow is updated within the 'Write' action located in the loop body block

Description

[key ASCMM-MNT-5-description]
This pattern identifies situations where the value (code:Value) of the <LoopElement> local storable data element (code:StorableUnit with code:StorableKind 'local') used in the condition of the loop control flow of code is updated within the <UpdateStatement> Write action (action:Writes) located in the loop body block (action:BlockUnit starting as the action:TrueFlow of the loop action:GuardedFlow and ending with an action:Flow back to the loop action:GuardedFlow)

Descriptor

[key ASCMM-MNT-5-descriptor]
ASCMM-MNT-5(LoopElement: loopElement,UpdateStatement: updateStatement)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-5-roles-loopElement] LoopElement
[key ASCMM-MNT-5-roles-updateStatement] UpdateStatement

7.8 Pattern definition of ASCMM-MNT-6: Commented-out Code Element Excessive Volume

Pattern Category

[key ASCMM-MNT-6-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-6-objective]
Avoid code blocks found in comments

Consequence

[key ASCMM-MNT-6-consequence]
Software that does contain commented-out code that can mistakenly be considered as active code and that can hide a lack of comments causes excessive modification effort

Measure Element

[key ASCMM-MNT-6-measure-element]
Number of instances where a named callable control element or method control element contains too many commented-out code items compared to a threshold that is based on the percentage of instructions in the callable control element or method control element that are in comments. Default threshold value for the percentage of commented-out out instructions is 2%.

Description

[key ASCMM-MNT-6-description]
This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored') or method control element (code:MethodUnit) contains too much commented-out code items (code:CodeItem), based on the <PercentageOfCommentedOutInstructions> percentage of instructions in the <ControlElement> callable or method control element that are in comments (code:CommentUnit) which exceeds the <PercentageOfCommentedOutInstructionsThresholdValue> threshold value.
Default value for <PercentageOfCommentedOutInstructionsThresholdValue> threshold value is 2%.

Descriptor

[key ASCMM-MNT-6-descriptor]
ASCMM-MNT-6(ControlElement: controlElement,PercentageOfCommentedOutInstructions: percentageOfCommentedOutInstructions, PercentageOfCommentedOutInstructionsThresholdValue: percentageOfCommentedOutInstructionsThresholdValue)

Variable input

[key ASCMM-MNT-6-input]
<PercentageOfCommentedOutInstructionsThresholdValue> maximum value

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-6-roles-controlElement] ControlElement
[key ASCMM-MNT-6-roles-percentageOfCommentedOutInstructions]
PercentageOfCommentedOutInstructions
[key ASCMM-MNT-6-roles-percentageOfCommentedOutInstructionsThresholdValue]
PercentageOfCommentedOutInstructionsThresholdValue

7.9 Pattern definition of ASCMM-MNT-7: Inter-Module Dependency Cycles

Pattern Category

[key ASCMM-MNT-7-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-7-objective]
Avoid circular dependencies between modules

Consequence

[key ASCMM-MNT-7-consequence]
Software that does not follow the principles of modularity causes excessive propagation of modification impacts

Measure Element

[key ASCMM-MNT-7-measure-element]
Number of instances where a module has references that cycle back to itself via the module callable or data relations cycle (for example, with JAVA this pattern means cycles between packages)

Description

[key ASCMM-MNT-7-description]
This pattern identifies situations where the <Module> module (code:Module) has references that cycle back to itself via the <ModuleDependencyCycle> module callable or data relations cycle (action:BlockUnit composed of action:CallableActions or action:DataActions).
As an example, with JAVA, this pattern means cycles between packages (code:Package).

Descriptor

[key ASCMM-MNT-7-descriptor]

ASCMM-MNT-7(Module: module,ModuleDependencyCycle: moduleDependencyCycle)

Variable input
(none applicable)

Comment
[key ASCMM-MNT-7-comment] Measure element contributes to Maintainability and Reliability

List of Roles
[key ASCMM-MNT-7-roles-module] Module
[key ASCMM-MNT-7-roles-moduleDependencyCycle] ModuleDependencyCycle

7.10 Pattern definition of ASCMM-MNT-8: Source Element Excessive Size

Pattern Category
[key ASCMM-MNT-8-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective
[key ASCMM-MNT-8-objective]
Avoid over-sizing of software elements

Consequence
[key ASCMM-MNT-8-consequence]
Software that does not follow the principles of modularity causes excessive propagation of modification impacts

Measure Element
[key ASCMM-MNT-8-measure-element]
Number of instances where a file has too many lines of code based on a threshold value. The default threshold value for number of lines of code is 1000

Description
[key ASCMM-MNT-8-description]
This pattern identifies situations where the <File> file (source:SourceFile) has too many lines of code, based on its <NumberOfLinesOfCode> number of lines of code which exceeds the <NumberOfLinesOfCodeThresholdValue> threshold value.
The default value for <NumberOfLinesOfCodeThresholdValue> is 1000.

Descriptor
[key ASCMM-MNT-8-descriptor]
ASCMM-MNT-8(File: file,NumberOfLinesOfCode: numberOfLinesOfCode, NumberOfLinesOfCodeThresholdValue: numberOfLinesOfCodeThresholdValue)

Variable input

[key ASCMM-MNT-8-input]
<NumberOfLinesOfCodeThresholdValue> maximum value

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-8-roles-file] File
[key ASCMM-MNT-8-roles-numberOfLinesOfCode] NumberOfLinesOfCode
[key ASCMM-MNT-8-roles-numberOfLinesOfCodeThresholdValue]
NumberOfLinesOfCodeThresholdValue

7.11 Pattern definition of ASCMM-MNT-9: Horizontal Layer Excessive Number

Pattern Category

[key ASCMM-MNT-9-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-9-objective]
Avoid the existence of too many or too few horizontal layers

Consequence

[key ASCMM-MNT-9-consequence]
Software that does not follow the principles of layered architectures (such as strict partitioning and strict call hierarchy) decreases comprehensibility as well as simplicity to evolve the code

Measure Element

[key ASCMM-MNT-9-measure-element]
Number of instances where a model of the architectural layers of an application contains too may or too few horizontal layers (excluding the vertical utility layers) based on comparison to a threshold value. The default value for the minimal number of horizontal layers is 4, and the default value for maximal number of horizontal layers is 8.

Description

[key ASCMM-MNT-9-description]
This pattern identifies situations where the <ArchitectureModel> model of the architectural layers contains too may or too few horizontal layers (structure:Layer), based on its <NumberOfHorizontalLayers> number of horizontal layers (that is, excluding the vertical utility layers) that is smaller than the <NumberOfHorizontalLayerThresholdMinimalValue> threshold value or greater than the <NumberOfHorizontalLayerThresholdMaximalValue> threshold value.
The default value for <NumberOfHorizontalLayerThresholdMinimalValue> is 4.

The default value for <NumberOfHorizontalLayerThresholdMaximalValue> is 8.

Descriptor

[key ASCMM-MNT-9-descriptor]
ASCMM-MNT-9(NumberOfHorizontalLayers:
numberOfHorizontalLayers,NumberOfHorizontalLayerThresholdMinimalValue:
numberOfHorizontalLayerThresholdMinimalValue, NumberOfHorizontalLayerThresholdMaximalValue:
numberOfHorizontalLayerThresholdMaximalValue, ArchitectureModel: architectureModel)

Variable input

[key ASCMM-MNT-9-input]
<ArchitectureModel> defining the application's architectural blueprint.
<NumberOfHorizontalLayerThresholdMinimalValue> minimal value
<NumberOfHorizontalLayerThresholdMaximalValue> maximal value.

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-9-roles-numberOfHorizontalLayers] NumberOfHorizontalLayers
[key ASCMM-MNT-9-roles-numberOfHorizontalLayerThresholdMinimalValue]
NumberOfHorizontalLayerThresholdMinimalValue
[key ASCMM-MNT-9-roles-numberOfHorizontalLayerThresholdMaximalValue]
NumberOfHorizontalLayerThresholdMaximalValue
[key ASCMM-MNT-9-roles-architectureModel] ArchitectureModel

7.12 Pattern definition of ASCMM-MNT-10: Named Callable and Method Control Element Multi-Layer Span

Pattern Category

[key ASCMM-MNT-10-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-10-objective]
Avoid unclear allocation of software elements to a single architectural layer

Consequence

[key ASCMM-MNT-10-consequence]
Software that does not follow the principles of layered architectures (such as strict partitioning and strict call hierarchy) decreases comprehensibility as well as simplicity to evolve the code

Measure Element

[key ASCMM-MNT-10-measure-element]
Number of instances where a callable or method control element is part of two architectural layers

Description

[key ASCMM-MNT-10-description]
This pattern identifies siuations where the <ControlElement> callable or method control element (code:ControlElement) is part of both <Layer1> and <Layer2> architectural layers (structure:Layer)

Descriptor

[key ASCMM-MNT-10-descriptor]
ASCMM-MNT-10(ControlElement: controlElement,Layer1: layer1, Layer2: layer2)

Variable input

[key ASCMM-MNT-10-input]
<ArchitectureModel> defining the application's architectural blueprint.

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-10-roles-controlElement] ControlElement
[key ASCMM-MNT-10-roles-layer1] Layer1
[key ASCMM-MNT-10-roles-layer2] Layer2

7.13 Pattern definition of ASCMM-MNT-11: Callable and Method Control Element Excessive Cyclomatic Complexity Value

Pattern Category

[key ASCMM-MNT-11-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-11-objective]
Avoid overly complex control flow

Consequence

[key ASCMM-MNT-11-consequence]
Software that does not follow the principles of structured programming degrades comprehensibility

Measure Element

[key ASCMM-MNT-11-measure-element]

Number of instances where a named callable control element or method control element has a control flow with a Cyclomatic Complexity number that exceeds a threshold value. Default threshold value for Cyclomatic Complexity is 20

Description

[key ASCMM-MNT-11-description]
This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored') or method control element (code:MethodUnit) has a control flow (action:ControlFlow) with a <CyclomaticComplexityValue> Cyclomatic Complexity which is greater than the <CyclomaticComplexityThresholdValue> threshold value.
Default value for <CyclomaticComplexityThreshold> is 20.

Descriptor

[key ASCMM-MNT-11-descriptor]
ASCMM-MNT-11(ControlElement: controlElement,CyclomaticComplexity: cyclomaticComplexity, CyclomaticComplexityThresholdValue: cyclomaticComplexityThresholdValue)

Variable input

[key ASCMM-MNT-11-input]
<CyclomaticComplexityThresholdValue> maximum value of distinct path through the control element

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-11-roles-controlElement] ControlElement
[key ASCMM-MNT-11-roles-cyclomaticComplexity] CyclomaticComplexity
[key ASCMM-MNT-11-roles-cyclomaticComplexityThresholdValue] CyclomaticComplexityThresholdValue

7.14 Pattern definition of ASCMM-MNT-12: Named Callable and Method Control Element with Layer-skipping Call

Pattern Category

[key ASCMM-MNT-12-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-12-objective]
Avoid breaches of layered architecture principles due to layer-skipping references

Consequence

[key ASCMM-MNT-12-consequence]
Software that does not follow the principles of layered architectures (such as strict partitioning and strict call hierarchy) decreases comprehensibility as well as simplicity to evolve the code

Measure Element

[key ASCMM-MNT-12-measure-element]
Number of instances where a named callable or method control element from a higher horizontal layer directly calls a named callable or method control element in a lower horizontal layer that is not adjacent to the upper layer making the call, as defined in a model of the application's architectural layers (this excludes the vertical utility layers that can be referenced from any horizontal layer)

Description

[key ASCMM-MNT-12-description]
This pattern identifies situations where the <CallerObject> named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored') from the <HigherLayer> higher horizontal layer (structure:Layer) directly calls (action:CallableRelations) the <CalleeObject> named callable or method control element from the <LowerLayer> lower horizontal layer (structure:Layer), while the <LowerLayer> layer is not the next lower layer to the <UpperLayer> layer, as defined in the <ArchitectureModel> model of the architectural layers; this excludes the vertical utility layers that can be referenced from any horizontal layers

Descriptor

[key ASCMM-MNT-12-descriptor]
ASCMM-MNT-12(CallerObject: callerObject,HigherLayer: higherLayer, CalleeObject: calleeObject, LowerLayer: lowerLayer, ArchitectureModel: architectureModel)

Variable input

[key ASCMM-MNT-12-input]
<ArchitectureModel> defining the application's architectural blueprint.

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-12-roles-callerObject] CallerObject
[key ASCMM-MNT-12-roles-higherLayer] HigherLayer
[key ASCMM-MNT-12-roles-calleeObject] CalleeObject
[key ASCMM-MNT-12-roles-lowerLayer] LowerLayer
[key ASCMM-MNT-12-roles-architectureModel] ArchitectureModel

7.15 Pattern definition of ASCMM-MNT-13: Callable and Method Control Element Excessive Number of Parameters

Pattern Category

[key ASCMM-MNT-13-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-13-objective]
Avoid over-parameterization

Consequence

[key ASCMM-MNT-13-consequence]
Software that does not cap the number of parameters degrades comprehensibility

Measure Element

[key ASCMM-MNT-13-measure-element]
Number of instances where a named callable control element or method control element has a number of parameters in its signature that exceeds a threshold value. Default threshold value for the number of parameters is 7.

Description

[key ASCMM-MNT-13-description]
This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored') or method control element (code:MethodUnit) has <ParameterNumber> parameters (code:ParameterUnit with ParameterKind 'byValue', 'byName', 'byReference', or 'variadic') in its signature (code:Signature) which is greater than the <ParameterNumberThreshold> threshold value.
Default value for the <ParameterNumberThreshold> is 7.

Descriptor

[key ASCMM-MNT-13-descriptor]
ASCMM-MNT-13(ControlElement: controlElement,ParameterNumber: parameterNumber, ParameterNumberThreshold: parameterNumberThreshold)

Variable input

[key ASCMM-MNT-13-input]
<ParameterNumberThreshold> maximum value of parameters in signature

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-13-roles-controlElement] ControlElement
[key ASCMM-MNT-13-roles-parameterNumber] ParameterNumber
[key ASCMM-MNT-13-roles-parameterNumberThreshold] ParameterNumberThreshold

7.16 Pattern definition of ASCMM-MNT-14: Callable and Method Control Element Excessive Number of Control Elements involving Data Element from Data Manager or File Resource

Pattern Category

[key ASCMM-MNT-14-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-14-objective]
Avoid the existence of control elements with too many data operations

Consequence

[key ASCMM-MNT-14-consequence]
Software that does not cap the number of data operations degrades comprehensibility by requiring the understanding of too many external data structures

Measure Element

[key ASCMM-MNT-14-measure-element]
Number of instances where a named callable control element or method control element has a number of operations involving a data manager or a file resource that exceeds a threshold value. Default threshold value for the number of data operations is 7

Description

[key ASCMM-MNT-14-description]
This pattern identifies situations where the <ControlElement> named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored') has too many control elements involving a data manager (platform:DataManager) or a file resource (platform:FileResource), based on its <NumberOfDataOperations> number of such control elements, which exceeds the <NumberOfDataOperationsThresholdValue> threshold value.
Default value for <NumberOfDataOperationsThresholdValue> is 7.

Descriptor

[key ASCMM-MNT-14-descriptor]
ASCMM-MNT-14(ControlElement: controlElement,NumberOfDataOperations: numberOfDataOperations, NumberOfDataOperationsThresholdValue: numberOfDataOperationsThresholdValue)

Variable input

[key ASCMM-MNT-14-input]
<NumberOfDataOperationsThresholdValue> maximum value of data operation in control element

Comment
(none applicable)

List of Roles
[key ASCMM-MNT-14-roles-controlElement] ControlElement
[key ASCMM-MNT-14-roles-numberOfDataOperations] NumberOfDataOperations
[key ASCMM-MNT-14-roles-numberOfDataOperationsThresholdValue] NumberOfDataOperationsThresholdValue

7.17 Pattern definition of ASCMM-MNT-15: Public Member Element

Pattern Category
[key ASCMM-MNT-15-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective
[key ASCMM-MNT-15-objective]
Avoid openly accessible data elements

Consequence
[key ASCMM-MNT-15-consequence]
Software that does not follow the principles of data encapsulation incurs the risk of data corruption

Measure Element
[key ASCMM-MNT-15-measure-element]
Number of instances where a storable data element or member data element is declared as public through a Create action

Description
[key ASCMM-MNT-15-description]
This pattern identifies situations where the <PublicDataElement> storable data element (code:StorableUnit) or member data element (code:MemberUnit) is declared as public (code:ExportKind 'public') through the <DataElementDeclarationStatement> Create action (action:Creates).

Descriptor
[key ASCMM-MNT-15-descriptor]
ASCMM-MNT-15(PublicDataElement: publicDataElement,DataElementDeclarationStatement: dataElementDeclarationStatement)

Variable input
(none applicable)

Comment
(none applicable)

List of Roles

[key ASCMM-MNT-15-roles-publicDataElement] PublicDataElement
[key ASCMM-MNT-15-roles-dataElementDeclarationStatement] DataElementDeclarationStatement

7.18 Pattern definition of ASCMM-MNT-16: Method Control Element Usage of Member Element from other Class Element

Pattern Category

[key ASCMM-MNT-16-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-16-objective]
Avoid direct access to data elements of another entity

Consequence

[key ASCMM-MNT-16-consequence]
Software that does not follow the principles of data encapsulation incurs the risk of data corruption

Measure Element

[key ASCMM-MNT-16-measure-element]
Number of instances where a method control element from a class element accesses a member element from another class element

Description

[key ASCMM-MNT-16-description]
This pattern identifies situations where the <Method> method control element (code:MethodUnit) from <Class1> class element (code:StorableUnit of code:DataType code:ClassUnit) accesses (action:DataRelations) the <Field> member element (code:MemberUnit) from <Class2> class element.

Descriptor

[key ASCMM-MNT-16-descriptor]
ASCMM-MNT-16(Class1: class1,Class2: class2, Field: field)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-16-roles-class1] Class1
[key ASCMM-MNT-16-roles-class2] Class2

[key ASCMM-MNT-16-roles-field] Field

7.19 Pattern definition of ASCMM-MNT-17: Class Element Excessive Inheritance Level

Pattern Category

[key ASCMM-MNT-17-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-17-objective]
Avoid overly complex object-oriented inheritance capabilities when dealing with the number of levels of inheritance of classes

Consequence

[key ASCMM-MNT-17-consequence]
Software that does not follow the principles of reuse requires more maintenance effort in order to propagate changes to all instances of duplicated code

Measure Element

[key ASCMM-MNT-17-measure-element]
Number of instances where the inheritance level of a class element (that is, the number of parent class units) exceeds a threshold value. The default threshold value for number of inheritance levels is 7

Description

[key ASCMM-MNT-17-description]
This pattern idendifies situations where the inheritence level (number of level of code:Extends relations) of the <Class> class element (code:StorableUnit with code:DataType code:ClassUnit) is considered as too large, based on its <NumberOfInheritenceLevels> number of parent class units which exceeds the <NumberOfInheritenceLevelsThresholdValue> threshold value.
The default value for <NumberOfInheritenceLevelsThresholdValue> is 7.

Descriptor

[key ASCMM-MNT-17-descriptor]
ASCMM-MNT-17(Class: class,NumberOfInheritenceLevels: numberOfInheritenceLevels, NumberOfInheritenceLevelsThresholdValue: numberOfInheritenceLevelsThresholdValue)

Variable input

[key ASCMM-MNT-17-input]
<NumberOfInheritenceLevelsThresholdValue> maximum value

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-17-roles-class] Class
[key ASCMM-MNT-17-roles-numberOfInheritenceLevels] NumberOfInheritenceLevels
[key ASCMM-MNT-17-roles-numberOfInheritenceLevelsThresholdValue] NumberOfInheritenceLevelsThresholdValue

7.20 Pattern definition of ASCMM-MNT-18: Class Element Excessive Number of Children

Pattern Category

[key ASCMM-MNT-18-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-18-objective]
Avoid overly complex object-oriented inheritance capabilities when dealing with the number of direct children of classes

Consequence

[key ASCMM-MNT-18-consequence]
Software that does not follow the principles of reuse requires more maintenance effort in order to propagate changes to all instances of duplicated code

Measure Element

[key ASCMM-MNT-18-measure-element]
Number of instances where the number of children of a class element (that is, its number of child classes) exceeds a threshold value. The default threshold value for number of children of a class element is 10

Description

[key ASCMM-MNT-18-description]
This pattern idendifies situations where the number of children (code:StorableUnit of code:DataType code:ClassUnit with direct code:Extends relation) of the <Class> class element (code:StorableUnit of code:DataType code:ClassUnit) is considered as too large, based on its <NumberOfChildren> number of child classes which exceeds the <NumberOfChildrenThresholdValue> threshold value.
The default value for <NumberOfChildrenThresholdValue> is 10.

Descriptor

[key ASCMM-MNT-18-descriptor]
ASCMM-MNT-18(Class: class,NumberOfChildren: numberOfChildren, NumberOfChildrenThresholdValue: numberOfChildrenThresholdValue)

Variable input

[key ASCMM-MNT-18-input]
<NumberOfChildrenThresholdValue> maximum value

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-18-roles-class] Class
[key ASCMM-MNT-18-roles-numberOfChildren] NumberOfChildren
[key ASCMM-MNT-18-roles-numberOfChildrenThresholdValue] NumberOfChildrenThresholdValue

7.21 Pattern definition of ASCMM-MNT-19: Named Callable and Method Control Element Excessive Similarity

Pattern Category

[key ASCMM-MNT-19-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-19-objective]
Avoid software element redundancy

Consequence

[key ASCMM-MNT-19-consequence]
Software that does not follow the principles of reuse requires more maintenance effort in order to propagate changes to all instances of duplicated code

Measure Element

[key ASCMM-MNT-19-measure-element]
Number of instances where a named callable control element or method control element contains multiple computational objects that are identical to computational objects in another named callable or method control element in the application (the application determines the scope of the search for the second code item)

Description

[key ASCMM-MNT-19-description]
This pattern identifies situations where the <ControlElement1> named callable control element (code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored') or method control element (code:MethodUnit) contains too many identical computational objects (code:ComputationalObject), based on the <NumberOfIdenticalTokens> number of identical computational objects with the <ControlElement2> named callable or method control element within the <Application> application; the <Application> application determines the scope of the search for the <ControlElement2> code item.

Descriptor

[key ASCMM-MNT-19-descriptor]
ASCMM-MNT-19(ControlElement1: controlElement1,ControlElement2: controlElement2, NumberOfIdenticalTokens: numberOfIdenticalTokens, Application: application)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-19-roles-controlElement1] ControlElement1
[key ASCMM-MNT-19-roles-controlElement2] ControlElement2
[key ASCMM-MNT-19-roles-numberOfIdenticalTokens] NumberOfIdenticalTokens
[key ASCMM-MNT-19-roles-application] Application

7.22 Pattern definition of ASCMM-MNT-20: Unreachable Named Callable or Method Control Element

Pattern Category

[key ASCMM-MNT-20-relatedPatts-maintainability] ASCMM_Maintainability

Pattern Sections

Objective

[key ASCMM-MNT-20-objective]
Avoid inactive code blocks that can mistakenly be considered as active and that can hide the active code in noise

Consequence

[key ASCMM-MNT-20-consequence]
Software that does not follow the principles of reuse requires more maintenance effort in order to propagate changes to all instances of duplicated code

Measure Element

[key ASCMM-MNT-20-measure-element]
Number of instances where a named callable control element or method control element is unreferenced by any other code item in the application (the application determines the scope of the search for code items that could call a callable or method control element)

Description

[key ASCMM-MNT-20-description]

This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored') or method control element (code:MethodUnit) is unreferenced (action:CallableRelations) by any other code item (code:CodeItem) in the <Application> application; the <Application> application determines the scope of the search for code items that could call the <ControlElement> callable or method control element.

Descriptor

[key ASCMM-MNT-20-descriptor]
ASCMM-MNT-20(ControlElement: controlElement,Application: application)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCMM-MNT-20-roles-controlElement] ControlElement
[key ASCMM-MNT-20-roles-application] Application

8. Calculation of the Automated Source Code Maintainability Measure and Functional Density (Normative)

8.1 Calculation of the Base Measure

A count of total violations of quality rules was selected as the best alternative for measurement. Software quality 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 critical violations of Maintainability quality rules cannot be isolated to a single component, but rather involve interactions among several components. Therefore, the Automated Source Code Maintainability Measure is computed as the sum of its 20 quality measure elements computed across the entire application.

The calculation of the Automated Source Code Maintainability Measure begins with determining the value of each of the 20 Maintainability measure elements. Each Maintainability measure element is measured as the total number of violations of its associated quality rule that are detected through automated analysis. Thus the value of each of the 20 Maintainability measure elements is represented as CISQ-MntME$_i$ where the range for i runs from 1 to 20.

$$\text{CISQ-MntME}_i = \sum \text{(all violations of type CISQ-MntME}_i \text{ detected through automated analysis)}$$

The value of the un-weighted and un-normalized Automated Source Code Maintainability Measure (CISQ-Mnt) is the sum of the values of the 20 Maintainability measure elements.

$$\text{CISQ-Mnt} = \sum_{i=1}^{20} \text{CISQ-MntME}_i$$

Higher values of CISQ-Mnt indicate a larger number of Maintainability-related defects in the application.

8.2 Functional Density of Maintainability Violations

In order to better compare Maintainability results among different applications, the Automated Source Code Maintainability Measure can be normalized by size to create a density measure. There are several size measures with which the density of Maintainability 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 applications. However, the OMG Automated Function Points measure offers an automatable size measure that, as an OMG Supported Specification, is standardized, adapted from the International Function Point User Group's (IFPUG) counting guidelines, and commercially supported. Although other size measures can be legitimately used to evaluate the density of Maintainability violations, the following density measure for Maintainability violations is derived from OMG supported specifications for Automated Function Points and the Automated Source Code Maintainability Measure. Thus, the functional density of Maintainability violations is a simple division expressed as follows.

$$\text{CISQ-Mnt-density} = \frac{\text{CISQ-Mnt}}{\text{AFP}}$$

9. Alternative Weighted Measures and Uses (Informative)

9.1 Additional Derived Measures

There are many additional weighting schemes that can be applied to the Automated Source Code Maintainability Measure or to the Maintainability measure elements that compose it. Table 3 presents several candidate weighted measures and their potential uses. However, these weighting schemes are not derived from any existing standards and are therefore not normative.

Table 3. Informative Weighting Schemes for Maintainability Measurement

Weighting scheme	Potential uses
Weight each maintainability measure by its severity	Measuring risk of maintenance problems such as software that is difficult to understand or change
Weight each Maintainability 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 Maintainability violations	Prioritizing modules or application components for corrective maintenance or replacement

10. References (Informative)

Consortium for IT Software Quality (2010). http://www.it-cisq.org

Curtis, B. (1980). Measurement and experimentation in software engineering. *Proceedings of the IEEE*, 68 (9), 1103-1119.

International Organization for Standards. ISO/IEC 25010 Systems and software engineering – System and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models

International Organization for Standards (2012). *ISO/IEC 25023 (in development) Systems and software engineering: Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of system and software product quality*.

International Organization for Standards (2012). *ISO/IEC TR 9126-3:2003, Software engineering — Product quality — Part 3: Internal metrics.*

McCabe, T.J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, *2* (4), 308–320.

Object Management Group (2014). Automated Function Points. formal 2014-01-03 http://www.omg.org/spec/AFP/, .

Appendix A: CISQ

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

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

Companies interested in joining CISQ held executive forums in Frankfurt, Germany; Arlington, VA; and Bangalore, India to set strategy and direction for the consortium. In these forums four quality characteristics were selected as the most important targets for automation—reliability, security, performance efficiency, and maintainability. These attributes cover four of the eight quality characteristics described in ISO/IEC 25010. Figure 1 displays the ISO/IEC 25010 software product quality model with the four software quality characteristics selected for automation by CISQ highlighted in orange. Each software quality characteristic is shown with the sub-characteristics that compose it.