An OMG® Automated Technical Debt Measure Publication

Automated Technical Debt Measure

Version 1.0

OMG Document Number: formal/2018-09-01

Release Date: September 2018

Normative Reference: https://www.omg.org/spec/ATDM/

Normative Machine Consumable Files:

https://www.omg.org/spec/ATDM/20170303/AutomatedTechnicalDebtMeasure.xmi

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP 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 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 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 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, 109 Highland Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

CORBA®, CORBA logos®, FIBO®, Financial Industry Business Ontology®, FINANCIAL INSTRUMENT GLOBAL IDENTIFIER®, IIOP®, IMM®, Model Driven Architecture®, MDA®, Object Management Group®, OMG®, OMG Logo®, SoaML®, SOAML®, SysML®, UAF®, Unified Modeling Language®, UML®, UML Cube Logo®, VSIPL®, and XMI® are registered trademarks of the Object Management Group, Inc.

For a complete list of trademarks, see: http://www.omg.org/legal/tm_list.htm. 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 OMG Specifications, Report an Issue.

Table of Contents

List of Figures

List of Tables

Preface

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 Metamodel™); 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 Specifications are available from the OMG website at:

http://www.omg.org/spec

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

OMG Headquarters
109 Highland Avenue
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 standards. Please consult http://www.iso.org

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification via the report form at the OMG main page, under OMG Specifications, Report an Issue.

This page intentionally left blank.

1 Scope

1.1 Purpose

The purpose of this specification is to establish a standard for automating a measure of Technical Debt that can be computed by source code analysis technologies which have implemented the CISQ Quality Characteristic measures. Within this defined focus, Technical Debt is calculated as an estimate of the effort to fix violations of good architectural and coding practices that must be remediated because of their risk and cost to the business. The foundation for specifying this measure has been provided in the CISQ Quality Characteristic measures approved as OMG standards, namely the Automated Source Code Reliability/Security/Performance Efficiency/Maintainability Measures. Using these OMG standards to provide the content for a measure of Technical Debt allows it to be based on published standards.

Adoption of the Technical Debt metaphor is growing as a means of communicating between IT executives and their technical staffs about quality issues and costs. Commercial IT executives have embraced the concept of Technical Debt for its value in predicting such factors as the costs of future corrective maintenance and the difficulty of enhancing or scaling an application. Currently, several static analysis vendors have added a measure of Technical Debt to their features, but none of these measures are based on an approved international standard.

1.2 The Technical Debt Metaphor

The Technical Debt metaphor was introduced by Ward Cunningham to describe how sub-optimal design decisions, often made to meet schedules, accumulated a debt that had to be repaid through corrective maintenance during future releases. CISQ participated in a 2016 workshop in Dagstuhl, Germany along with 40 members of the Technical Debt research community to create a framework for defining the metaphor and guiding research (Curtis, 2016). Two conclusions were reached at the end of the week:

1) There is no universally agreed definition of Technical Debt.
2) Industry and the research community have different goals in defining and measuring Technical Debt.

Regarding the second point, many in the research community restrict the domain of Technical Debt to sub-optimal design decisions that primarily affect maintainability issues such as changeability and scalability. Consistent with Cunningham's original formulation of the concept, they do not consider missing features, functional defects, or most structural flaws related to reliability, security, or performance efficiency to be part of the Technical Debt domain. The participants in the Dagstuhl workshop were unable to construct a crisp definition delimiting the domain of weaknesses to be included in Technical Debt.

In contrast, industry wants a measure that predicts the future costs of corrective maintenance and other software quality-related outcomes. Since the Consortium for IT Software Quality (CISQ) is an industry consortium, it has developed a specification for Technical Debt that is designed to predict corrective maintenance costs and related factors to guide IT decisions and resource allocations. The CISQ measure of Technical Debt builds on the existing four OMG standards CISQ has developed for measuring the structural quality of software. The violations of choosing 'debt' as a metaphor engages a set of financial concepts that help executives think about software quality in business terms. The components that comprise Technical Debt provide a foundation for the economics of software quality. The metaphor can be partitioned into the following elements:

- ***Technical Debt*** – Future costs attributable to known structural weaknesses in production code that must be fixed. Technical Debt includes both the debt's principal and interest. A weakness in production code is only included in Technical Debt calculations if those responsible for the application believe it is a 'must-fix' problem, therefore incurring corrective maintenance costs in a future release. Technical Debt is a primary component of the cost of application ownership.

- ***Principal*** – The cost of remediating must-fix problems in production code. At a minimum, the principal is calculated from the number of hours required to remediate these problems, multiplied by the fully burdened hourly cost of those involved in designing, implementing, and unit testing these fixes.
- ***Interest*** – Continuing costs, primarily in IT, attributable to must-fix problems so long as they remain in production code. These ongoing costs can result from the excessive effort to modify unnecessarily complex code, greater resource usage by inefficient code, etc.
- ***Business Risk*** – Potential costs to the business if must-fix problems in production code cause damaging operational events such as outages, data corruption, performance degradation, and security breaches.
- ***Liability*** – Costs to the business resulting from operational problems caused by flaws in production code. These flaws include both must-fix problems included in the calculation of Technical Debt as well as problems not listed as must-fix because their risk was underestimated.
- ***Opportunity Cost*** – Benefits such as revenue from new features that could have been achieved had resources been committed to developing new capability rather than being assigned to retire Technical Debt. Opportunity costs represent the tradeoff that application managers and executives must weigh when deciding how much effort to devote to retiring Technical Debt.

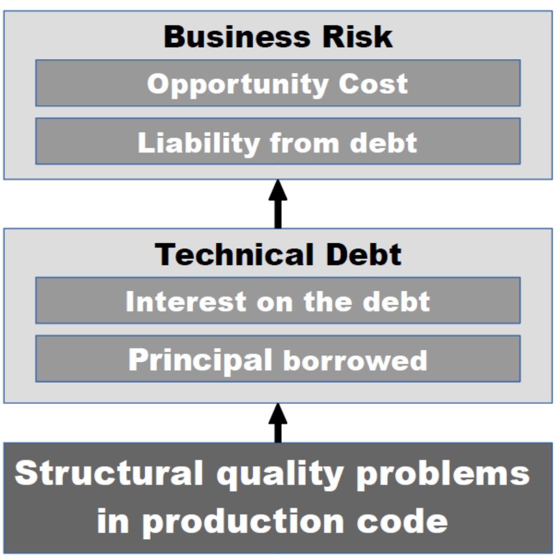

Figure 1.1 - The Technical Debt Metaphor

Relationships among components of the Technical Debt metaphor are displayed in Figure 1.1. The cost to fix structural quality problems constitutes the principal of the debt, while the inefficiencies they cause such as greater maintenance effort or excessive computing resources represent interest costs on the debt. The structural problems underlying Technical Debt also create business risks such as outages and security breaches, and the negative events they can cause result in liabilities such as lost revenue from online sales or costly clean-up from a security breach. The effort committed to remediating Technical Debt instead of developing new business functionality represents opportunity costs related to lost benefits that might otherwise have been achieved.

1.3 MeasuringTechnical Debt

This specification is narrowly focused on defining a measure of principal of a Technical Debt that can be computed from the CISQ Quality Characteristic measures. Other components of the Technical Debt metaphor may become the focus of future OMG specifications. There are five steps in calculating this measure that form the normative component of the specification for Technical Debt:

1. Detect occurrences of patterns specified as weaknesses by four OMG approved specifications: the Automated Source Code Reliability/Security/Performance Efficienty/Maintainability Measures; that is, detect the 86 violations of good architectural and coding practices that constitute these measures.
2. Assign an estimate of the amount of time to remediate each occurrence of a weakness based on a survey of software professionals; the estimate is a constant for each occurrence.
3. Collect qualification information about the occurrences of each weakness.
4. Compute an adjustment factor as a function of qualification information about each of the occurrences to negatively or positively impact the effort estimate.
5. Sum the total amount of time across all the occurrences for all 86 violations. The normative specification does not include variations in labor costs, skill levels, or currencies (dollars, euros, rupees, etc.) as these are adjustments that must be made based on local conditions.

The specification will also include a set of non-normative usage scenarios showing how qualification information from step 3 can be used to manage Technical Debt measures as well as customize the Technical Debt measure to local conditions within an organization. These factors include issues related to system testing and other processes that can vary across organizations.

1.4 Technical Debt as an Estimate

Technical Debt measures are most frequently used to estimate future corrective maintenance costs as input to decisions such as budgeting maintenance, allocating developer effort, or replacing an application. Corrective maintenance includes all the activities involved in analyzing a weakness, designing and implementing a correction, testing it, and any deployment activities that can be traced directly to the corrected weakness. The measure defined in this specification is a correlated rather than absolute measure of Technical Debt. That is, it is a predictor of the amount of corrective maintenance effort needed for an application. Each organization must develop its own equation linking Technical Debt with its costs and other outcomes. There are three primary issues that affect the usefulness of this measure.

First, the violations incorporated in the four Automated Source Code Reliability/Security/ Performance Efficiency/Maintainability Measures specifications were selected because they were considered weaknesses of sufficient severity that must be remediated because of their risk to costs and operational performance. However, an organization may choose to remediate only some of these violations, not incurring the debt associated with other violations. In this case the Technical Debt measure will over-estimate corrective maintenance costs. Conversely, an organization can choose to remediate more violations of good practice than are included in the CISQ measures, in which case Technical Debt underestimates corrective maintenance costs. In either case, Technical Debt provides a common benchmark for comparing the structural quality of different applications that can be adjusted to better represent local quality assurance strategies.

Second, there are no existing industry-wide repositories of effort data related to remediating violations of good architectural and coding practices. Consequently, the remediation times used in this specification are based on surveys of experienced developers. A survey of requested developers to estimate their time-to-fix for the 86 weaknesses included in the 4 CISQ Quality Characteristic measures (CISQ, 2017). The times were to include analysis of the weakness through unit test. Most respondents were primarily developing in Java, .NET, or C# and the distribution of their times were roughly similar. Default times for each weakness were developed from the modal tendency of these distributions with some adjustments based their estimate of having to remediate more than one component or file.

Variations in time estimates and sampling factors could impact the default remediation times drawn from these data. Consequently, the specification allows for these default times to be overridden with local estimates where appropriate. As more data become available, these default constants can be updated if necessary in a future revision of this specification. The remediation times for each violation are adjusted using the qualification information discussed in later clauses. Similarly, these adjustment factors can be updated in future revisions as data become available regarding their value in improving estimates of remediation time.

Third, Technical Debt measures weaknesses in the structural quality of an application. It does not measure functional defects which must be remediated. Therefore, this measure does not assess all factors contributing to corrective maintenance costs. However, since practices related to detecting the non-functional, structural weaknesses in software have lagged those focused on functional defects, future maintenance effort is most often focused on structural weaknesses. Consequently, Technical Debt provides an estimate of these costs that can be adjusted to account for local experience in remediating functional defects that escape testing and must be fixed in future releases.

In view of these considerations, Technical Debt provides an estimate based on OMG standards that can be used to predict future risk and cost outcomes for an application. It can be used as a benchmark for comparing applications and it can be adjusted to local quality assurance practices and strategies.

2 Conformance

2.1 Overview

Implementations of this specification shall be able to demonstrate all five of the following attributes to claim conformance—automated, complete, objective, transparent, and verifiable:

- ***Automated*** - The calculation of this measure shall be fully automated. A conformant technology shall be able to consume and process machine readable outputs reporting weaknesses detected from analysis of the 4 CISQ Quality Characteristic measures and elements from analysis of the Automated Enhancement Points measure. Analyses to develop these inputs require 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.
- ***Complete*** - A conformant technology shall be able to calculate the Technical Debt measure as specified in this document. Consequently, the technology used to compute this measure shall be able to receive and process outputs produced by technologies that comply with the following OMG specifications:
    - Automated Source Code Reliability Measure
    - Automated Source Code Security Measure
    - Automated Source Code Performance Efficiency Measure
    - Automated Source Code Maintainability Measure
    - Automated Enhancement Points

- ***Objective*** - After the source code has been prepared for analysis using the information provided as inputs, the analysis, calculation, and presentation of results shall not require further human intervention. The analysis and calculation shall be able to repeatedly produce the same results and outputs on the same body of software.
- ***Transparent*** - Implementations that conform to this specification shall clearly list all tools that supplied inputs to this measure, as well as the source code, non-source code artifacts, and other information used to prepare the source code for analysis by these other tools.
- ***Verifiable*** - A conformant implementation shall state the assumptions and heuristics it uses in computing this measure in sufficient detail that the calculations can be independently verified by third parties. Clause 7.8 describes the measures and information required in the generated output. In addition, all inputs used are required to be clearly described and itemized so that they can be audited by a third party.

3 References

3.1 Normative References

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:

- Knowledge Discovery Metamodel, version 1.3 (KDM), formal/2011-08-04
- Structured Metrics Metamodel, version 1.1 (SMM), formal/2015-10-03
- Meta Object Facility, version 2.5 (MOF), formal/2015-06-05
- XML Metadata Interchange, version 2.5.1 (XMI), formal/2015-06-07
- Object Constraint Language, version 2.4 ()CL), formal/2014-02-03
- Automated Source Code Reliability Measure, version 1.0 (ASCRM), formal/2016-01-03
- Automated Source Code Security Measure, version 1.0 (ASCSM), formal/2016-01-04
- Automated Source Code Performance Efficienty Measure, version 1.0 (ASCPEM), formal/2016-01-02
- Automated Source Code Maintainability Measure, version 1.0 (ASCMM), formal/2016-01-01
- Automated Enhancement Points, version 1.0 (AEP), ptc/2016-06-03
- Structured Patterns Metamodel Specification 1.0 (SPMS), formal/2015-10-01
- ISO/IEC 25010 Systems and software engineering – System and software product Quality Requirements and Evaluation (SquaRE) – System and software quality models

3.2 Non-normative References

List of non-normative references:

- Paris Avgeriou, Philippe Kruchten, Robert L. Nord, Ipek Ozkaya, Carolyn Seaman (2016). Reducing friction in software development. *IEEE Software*, 33 (1), 66-73.
- Consortium for IT Software Quality (2017). *CISQ Compliance Assessment*. Needham, MA: Object Management Group.
- Consortium for IT Software Quality (2017). *CISQ Time-to-Fix Survey*. Needham, MA: Object Management Group.
- Ward Cunningham, “The WyCash Portfolio Management System”, OOPSLA ’92 Experience Report
- Curtis, B. (2016). Measuring and communicating the technical debt metaphor in industry. *Managing Technical Debt in Software Engineering*. Dagstuhl, Germany: Dagstuhl Publishing, 121-122.

- B. Curtis, J. Sappidi, & A. Szynkarski, (2012). Estimating the principal of an application's technical debt. *IEEE Software, 29* (6), 34-42.
- P. Kruchten, R. L. Nord, I. Ozkaya (2012). Technical Debt: From Metaphor to Theory and Practice. *IEEE Software*, 29 (6), 30-33.
- Software Engineering Body of Knowledge, V3.0 (SWEBOK). http://www.computer.org/web/swebok/v3

4 Terms and Definitions

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

Adjusted Remediation Effort

The number of minutes needed to remediate a specific source code pattern that has been adjusted by qualification measures.

Application Model

The Application Model is composed of the computational objects in the source code and their relationships, some of which can contain processing rules and logic. (KDM)

Automated Technical Debt

Automated Technical Debt sums the Remediation Efforts of all detected Technical Debt Items that are defined as Occurrences of Patterns representing weaknesses enumerated in the Automated Source Code Reliability/Security/Performance Efficiency/Maintainability Measure specifications.

Automated Maintainability Remediation Effort

Automated Maintainability Remediation Effort sums the Remediation Efforts of all detected Technical Debt Items that are Occurrences of Patterns representing weaknesses in the Automated Source Code Maintainability Measure specification.

Automated Performance Efficiency Remediation Effort

Automated Performance Efficiency Remediation Effort sums the Remediation Efforts of all detected Technical Debt Items that are Occurrences of Patterns representing weaknesses in the Automated Source Code Performance Efficiency Measure specification.

Automated Reliability Remediation Effort

Automated Reliability Remediation Effort sums the Remediation Efforts of all detected Technical Debt Items that are Occurrences of Patterns representing weaknesses in the Automated Source Code Reliability Measure specification.

Automated Security Remediation Effort

Automated Security Remediation Effort sums the Remediation Efforts of all detected Technical Debt Items that are Occurrences of Patterns representing weaknesses in the Automated Source Code Security Measure specification.

CISQ Quality Characteristic Measures

The 4 CISQ Quality Characteristic measures are Automated Source Code Reliability/Security/Performance Efficiency/Maintainability Measures. These measures have been approved as OMG standards. The scope of each CISQ Quality Characteristic measure conforms to its definition in ISO/IEC 25010. (ASCMM, ASCRM, ASCPEM, ASCSM).

Complexity [or Effort Complexity]

The Complexity – or Effort Complexity – of the code elements implementing an Occurrence is qualification information that is measured according to the Effort Complexity definition from the Automated Enhancement Points (AEP) specification. (AEP)

Concentration

Concentration is qualification information that measures the number of Occurrences within any Code Element in the software.

Contextual Technical Debt

Contextual Technical Debt is a measure of Technical Debt that only measures Technical Debt Items that are a selected subset of the Patterns included in Technical Debt, and/or that use a Remediation Effort configuration different from the one specified in the current document, and/or incorporating an adjustment factor as presented in 7.3.4, and/or incorporating modifying factors such as the ones presented in the informative Clause 6.

Corrective Maintanance

Corrective maintenance includes all the activities involved in analyzing a weakness, designing and implementing a correction, testing it, and any deployment activities that can directly be traced to the corrected weakness.

Evolution Status

The Evolution Status of an Occurrence and of code elements implementing an Occurrence is qualification information which indicates if the Occurrence or the code elements implementing an Occurrence have been added, updated, or deleted between measured revisions of the software.

Exposure

The Exposure of an Occurrence is qualification information that measures the level of connectedness of the Occurrence with the rest of the software, both directly and indirectly through call paths.

Occurrence [or Pattern Occurrence]

An occurrence (or Pattern Occurrence) designates a single instance of a Source Code Pattern (or Pattern) representing a weakness that has been implemented in the measured software. (ASCMM, ASCRM, ASCPEM, ASCSM)

Occurrence Gap Size

In the context of patterns which rely on roles that model values and threshold values that are not to be exceeded, the gap between these values must be closed to remediate this weakness; the Occurrence Gap Size is the extent of the gap, measured as the difference between the values and the thresholds.

Pattern [or Source Code Pattern]

A Pattern (or Source Code Pattern) designates a set of elements and their relationships that can be detected through automated matching of the pattern description with structures in the source code. In the Automated Source Code Maintainability/Reliability/Performance Efficiency/Security Measure specifications, patterns provide analyzable descriptions by which a weakness related to one of the four CISQ Quality Characteristics specifications can be detected in the source code. (SPMS, ASCMM, ASCRM, ASCPEM, ASCSM)

Pattern Role

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 Pattern Definition. (SPMS)

Qualification Information

Qualification information describes attributes of the software context affecting an occurrence that can cause variation in the time required to remediate the specific occurrence. The qualification factors include complexity, concentration, evolution status, exposure, and technological diversity.

Qualification Measures

Qualification measures quantify the qualification information so they can be applied as adjustments in calculating the Automated Technical Debt Measure.

Remediation Effort

Remediation Effort designates the time required to remove an occurrence – or a set of occurrences – of a Technical Debt Item from the software. It covers the coding activity as well as unit/non-regression testing activities.

Software Cost

Software Cost is the financial burden of developing or maintaining the software. As used in this specification it is the money spent on corrective maintenance.

Software Value

Software Value is the business benefit derived by the ultimate consumers of the software.

Software Quality

Software Quality is the degree to which the software meets customer or user needs or expectations, and is free of defects that could cause the software to fail to meet these needs or expectations in the future. (ISO 25010)

Technical Debt Item

A Technical Debt Item is an atomic constitutive element of Technical Debt, that is, an instance of a weakness incorporated into one of the four CISQ Quality Characteristic measures. A Technical Debt Item is identified by detection of its characteristic Source Code Pattern.

Technological Diversity

The Technological Diversity of an Occurrence is qualification information that measures the number of distinct programming languages in which the code elements included in a single occurrence of a source code pattern are written.

Unadjusted Remediation Effort

The number of minutes needed to remediate a specific source code pattern before being adjusted by qualification measures. Default Unadjusted Remediation Efforts have been assigned to each source code pattern in the CISQ Quality Characteristics. However, these default values can be changed to better fit the local context and conditions prior to calculating ATDM.

Weakness [or Violation]

A weakness [or violation] designates a non-conformity to good architectural and coding practices defined in the CISQ Quality Characteristic specifications that must be remediated. (ASCMM, ASCRM, ASCPEM, ASCSM).

5 Symbols

AEP	Automated Enhancement Points
AMREM	Automated Maintainability Remediation Effort Measure
APEREM	Automated Performance Efficiency Remediation Effort Measure
ARREM	Automated Reliability Remediation Effort Measure
ASREM	Automated Security Remediation Effort Measure
ASCMM	Automated Source Code Maintainability Measure
ASCPEM	Automated Source Code Performance Efficiency Measure
ASCRM	Automated Source Code Reliability Measure
ATDM	Automated Technical Debt Measure
CISQ	Consortium for IT Software Quality
CTDM	Contextual Technical Debt Measure
SPMS	Structured Patterns Metamodel Specification
TD	Technical Debt

This page intentionally left blank

6 Foundational Information (Informative)

6.1 CISQ Quality Characteristic Measures

The Automated Technical Debt Measure (ATDM) is calculated from occurrences of the 86 weaknesses that are included in the 4 CISQ Quality Characteristic measures. Detecting and counting these weaknesses is the starting point for calculating ATDM. The CISQ Quality Characteristic measures consist of the following approved specifications of the OMG:

- ***Automated Source Code Reliability Measure (ASCRM)*** —violations of good architectural and coding practice that can cause outages, delayed recovery, data corruption, and unpredictable operational behavior.
- ***Automated Source Code Security Measure (ASCSM)*** —violations of good architectural and coding practice in an application that allow unauthorized intrusion into the application's source code, data store, operations, or connections.
- ***Automated Source Code Performance Efficiency Measure (ASCPEM)*** —violations of good architectural and coding practice that can result in slow response, degraded performance, or excessive use of computational resources.
- ***Automated Source Code Maintainability Measure (ASCMM)*** —violations of good architectural and coding practice that make an application's source code difficult to understand or modify.

The following sub clauses provide additional background information about the scope and content of Automated Source Code /Reliability/Security/Performance Efficiency/Maintainability Measure specifications regarding:

- The nature of development artifacts involved.
- The identification of occurrences of source code patterns from the *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM* specifications, including the modeling of the effort associated with remediating an actual Technical Debt Item.
- The qualification of each occurrence, that is, additional information associated with the occurrence to aid in prioritizing its remediation and other decisions or estimates.

6.1.1 Development Artifacts

Development artifacts composing a Technical Debt can be found in various locations:

- **Source Code**, including implemented Software Structure and Architecture
- **Build Scripts**
- **Test Scripts**
- **Documentation**
- **Technology**
- **Design**, including Architecture Decisions

6.1.1.1 Source Code

Source Code Development artifacts include all the elements and inter-element relationships that exist in the source code and the application model produced from it. The application model allows automated tools to analyze the software structure and architecture as implemented in the source code, rather than how the structure and architecture were designed or documented. Source Code Development artifacts are represented by the following elements from the Knowledge Discovery Meta-model (*KDM*):

- **Source Package** - representing physical artifacts
- **Code package** - representing low-level building blocks of the software
- **Action package** - representing low-level relationships and statements

- **Platform package** - representing run-time resources
- **UI package** - representing user-interface aspects of the software
- **Event package** - representing event-driven aspects of the software
- **Data package** - representing persistent data aspects of the software
- **Structure package** - representing architectural components of the software

6.1.1.2 Build Scripts

Build Scripts Development artifacts include all the elements produced by development teams to build the software. Build Scripts Development artifacts are represented by the following elements from the Knowledge Discovery Metamodel (*KDM*):

- **Build package** - representing artifacts related to the build process
- **Source and Code packages** - used as build resources

6.1.1.3 Test Scripts

Test Scripts Development artifacts include all the elements produced by development teams to verify the correct functioning of the software. Test Scripts Development artifacts are represented by the same *KDM* packages as Source Code Development artifacts, and only differ in nature by the intent behind their production.

6.1.1.4 Documentation

Documentation Development artifacts include all the elements produced by development teams to help understand how the software was developed. They do not include documentation artifacts that are found in the source code, and that are already covered by Source Code Development artifacts.

6.1.1.5 Technology

Technology Development artifacts are the programming languages used in developing the software, as well as third party supplied components that are required to develop and execute the software. In other words, they include all elements used in the software that are not under the control of the development organization, but can negatively impact the software or its development process. For example, the Technical Debt created by the discontinuation of the technologies used in developing the software.

6.1.1.6 Design

Design Development artifacts are all the decisions, including architectural decisions made and documented prior to developing the code. Design Development artifacts do not include the software design and architectural elements that are determined by analyzing the source code.

6.1.2 Source Code Patterns Representing Weaknesses

The Automated Source Code Maintainability/Reliability/Performance Efficiency/Security Measure specifications each defines a list of source code patterns that are considered severe enough violations of good architectural and coding practice that they must be remediated in a near-term release. These source code patterns are conformant to pattern formats specified in the Structured Patterns Metamodel Specification (SPMS). These source code patterns constitute Technical Debt Items, and are listed by their respective CISQ Quality Characteristic measure.

6.1.2.1 Automated Source Code Security Measure (ASCSM) Source Code Patterns

Table 6.1 lists the patterns defined in the Automated Source Code Security Measure specifications version 1.0. They are listed along with their Common Weakness Enumeration identifier.

Table 6.1 - List of ASCSM 1.0 Patterns

ASCSM Pattern Name
ASCSM-CWE-120 Buffer Copy without Checking Size of Input
ASCSM-CWE-129 Array Index Improper Input Neutralization
ASCSM-CWE-134 Format String Improper Input Neutralization
ASCSM-CWE-22 Path Traversal Improper Input Neutralization
ASCSM-CWE-252-resource Unchecked Return Parameter Value of named Callable and Method Control Element with Read, Write, and Manage Access to Platform Resource
ASCSM-CWE-327 Broken or Risky Cryptographic Algorithm Usage
ASCSM-CWE-396 Declaration of Catch for Generic Exception
ASCSM-CWE-397 Declaration of Throws for Generic Exception
ASCSM-CWE-434 File Upload Improper Input Neutralization
ASCSM-CWE-456 Storable and Member Data Element Missing Initialization
ASCSM-CWE-606 Unchecked Input for Loop Condition
ASCSM-CWE-667 Shared Resource Improper Locking
ASCSM-CWE-672 Expired or Released Resource Usage
ASCSM-CWE-681 Numeric Types Incorrect Conversion
ASCSM-CWE-99 Improper Control of Resource Identifiers ('Resource Injection')
ASCSM-CWE-772 Missing Release of Resource after Effective Lifetime
ASCSM-CWE-78 OS Command Injection Improper Input Neutralization
ASCSM-CWE-789 Uncontrolled Memory Allocation
ASCSM-CWE-79 Cross-site Scripting Improper Input Neutralization
ASCSM-CWE-798 Hard-Coded Credentials Usage for Remote Authentication
ASCSM-CWE-835 Loop with Unreachable Exit Condition ('Infinite Loop')
ASCSM-CWE-89 SQL Injection Improper Input Neutralization

6.1.2.2 Automated Source Code Reliability Measure (ASCRM) Source Code Patterns

Table 6.2 lists the patterns defined in the Automated Source Code Reliability Measure specifications version 1.0. Common Weakness Enumeration identifiers are listed for those weaknesses to which an identifier has been assigned.

Table 6.2: List of ASCRM 1.0 patterns

ASCRM pattern name
ASCRM-CWE-120 Buffer Copy without Checking Size of Input
ASCRM-CWE-252-data Unchecked Return Parameter Value of named Callable and Method Control Element with Read, Write, and Manage Access to Data Resource
ASCRM-CWE-252-resource Unchecked Return Parameter Value of named Callable and Method Control Element with Read, Write, and Manage Access to Platform Resource
ASCRM-CWE-396 Declaration of Catch for Generic Exception
ASCRM-CWE-397 Declaration of Throws for Generic Exception
ASCRM-CWE-674 Uncontrolled Recursion
ASCRM-CWE-456 Storable and Member Data Element Missing Initialization
ASCRM-CWE-704 Incorrect Type Conversion or Cast
ASCRM-CWE-772 Missing Release of Resource after Effective Lifetime
ASCRM-CWE-788 Memory Location Access After End of Buffer
ASCRM-RLB-1 Empty Exception Block
ASCRM-RLB-2 Serializable Storable Data Element without Serialization Control Element
ASCRM-RLB-3 Serializable Storable Data Element with non-Serializable Item Elements
ASCRM-RLB-4 Persistent Storable Data Element without Proper Comparison Control Element
ASCRM-RLB-5 Runtime Resource Management Control Element in a Component Built to Run on Application Servers
ASCRM-RLB-6 Storable or Member Data Element containing Pointer Item Element without Proper Copy Control Element
ASCRM-RLB-7 Class Instance Self Destruction Control Element
ASCRM-RLB-8 Named Callable and Method Control Elements with Variadic Parameter Element
ASCRM-RLB-9 Float Type Storable and Member Data Element Comparison with Equality Operator
ASCRM-RLB-10 Data Access Control Element from Outside Designated Data Manager Component
ASCRM-RLB-11 Named Callable and Method Control Element in Multi-Thread Context with non-Final Static Storable or Member Element
ASCRM-RLB-12 Singleton Class Instance Creation without Proper Lock Element Management
ASCRM-RLB-13 Inter-Module Dependency Cycles

ASCRM-RLB-14 Parent Class Element with References to Child Class Element
ASCRM-RLB-15 Class Element with Virtual Method Element without Virtual Destructor
ASCRM-RLB-16 Parent Class Element without Virtual Destructor Method Element
ASCRM-RLB-17 Child Class Element without Virtual Destructor unlike its Parent Class Element
ASCRM-RLB-18 Storable and Member Data Element Initialization with Hard-Coded Network Resource Configuration Data
ASCRM-RLB-19 Synchronous Call Time-Out Absence

6.1.2.3 Automated Source Code Performance Efficiency Measure (ASCPEM) Patterns

Table 6.3 lists the patterns defined in the Automated Source Code Performance Efficiency Measure specifications version 1.0.

Table 6.3: List of ASCPEM 1.0 patterns

ASCPEM pattern name
ASCPEM-PRF-1 Static Block Element containing Class Instance Creation Control Element
ASCPEM-PRF-2 Immutable Storable and Member Data Element Creation
ASCPEM-PRF-3 Static Member Data Element outside of a Singleton Class Element
ASCPEM-PRF-4 Data Resource Read and Write Access Excessive Complexity
ASCPEM-PRF-5 Data Resource Read Access Unsupported by Index Element
ASCPEM-PRF-6 Large Data Resource ColumnSet Excessive Number of Index Elements
ASCPEM-PRF-7 Large Data Resource ColumnSet with Index Element of Excessive Size
ASCPEM-PRF-8 Control Elements Requiring Significant Resource Element within Control Flow Loop Block
ASCPEM-PRF-9 Non-Stored SQL Callable Control Element with Excessive Number of Data Resource Access
ASCPEM-PRF-10 Non-SQL Named Callable and Method Control Element with Excessive Number of Data Resource Access
ASCPEM-PRF-11 Data Access Control Element from Outside Designated Data Manager Component
ASCPEM-PRF-12 Storable and Member Data Element Excessive Number of Aggregated Storable and Member Data Elements
ASCPEM-PRF-13 Data Resource Access not using Connection Pooling capability
ASCPEM-PRF-14 Storable and Member Data Element Memory Allocation Missing De-Allocation Control Element
ASCPEM-PRF-15 Storable and Member Data Element Reference Missing De-Referencing Control Element

6.1.2.4 Automated Source Code Maintainability Measure (ASCMM) Patterns

Table 6.4 lists the patterns defined in the Automated Source Code Maintainability Measure specifications version 1.0.

Table 6.4: List of ASCMM 1.0 patterns

ASCMM pattern name
ASCMM-MNT-1 Control Flow Transfer Control Element outside Switch Block
ASCMM-MNT-2 Class Element Excessive Inheritance of Class Elements with Concrete Implementation
ASCMM-MNT-3 Storable and Member Data Element Initialization with Hard-Coded Literals
ASCMM-MNT-4 Callable and Method Control Element Number of Outward Calls
ASCMM-MNT-5 Loop Value Update within the Loop
ASCMM-MNT-6 Commented-out Code Element Excessive Volume
ASCMM-MNT-7 Inter-Module Dependency Cycles
ASCMM-MNT-8 Source Element Excessive Size
ASCMM-MNT-10 Named Callable and Method Control Element Multi-Layer Span
ASCMM-MNT-11 Callable and Method Control Element Excessive Cyclomatic Complexity Value
ASCMM-MNT-12 Named Callable and Method Control Element with Layer-skipping Call
ASCMM-MNT-13 Callable and Method Control Element Excessive Number of Parameters
ASCMM-MNT-14 Callable and Method Control Element Excessive Number of Control Elements involving Data Element from Data Manager or File Resource
ASCMM-MNT-15 Public Member Element
ASCMM-MNT-16 Method Control Element Usage of Member Element from other Class Element
ASCMM-MNT-17 Class Element Excessive Inheritance Level
ASCMM-MNT-18 Class Element Excessive Number of Children
ASCMM-MNT-19 Named Callable and Method Control Element Excessive Similarity
ASCMM-MNT-20 Unreachable Named Callable or Method Control Element

6.1.2.5 Source Code Pattern Roles

Each source code pattern definition contains a specification of Roles (**SPMS:Definitions::Roles**). According to the Structured Patterns Metamodel Specification (SPMS), “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.”

In the current document, measurements of pattern occurrences rely on these Roles in the following ways:

- Some patterns rely on roles that model values and threshold values. For example, in the ASCPEM-PRF-10 pattern, one occurrence exists when the number of data queries (ASCPEM-PRF-10-roles-numberOfDataQueries) exceeds the number of data queries threshold value (ASCPEM-PRF-10-roles-numberOfDataQueriesThresholdValue). Therefore, to remediate this weakness the gap between these values must be closed. In these cases (enumerated in normative 7.3.3.7), the remediation effort is modeled by the multiplication of a constant by the extent of the gap via the adjustement factor.
- Qualification information collection relies on the implementation of these Roles.

6.1.2.6 Source Code Pattern Comments

Some pattern definitions contain in the Comment pattern section the following term: (SPMS:Definitions::PatternSection). In the CISQ Quality Characteristic measure specifications these comments indicate shared patterns between these specifications. For example, ASCSM-CWE-120-comment and ASCRM-CWE-120-comment state that "Measure element contributes to Security and Reliability." Information in such comments are used to avoid duplicate counting of remediation effort for an occurrence of CWE-120 when computing the overall Technical Debt score.

6.1.2.7 Adherence to ASCMM, ASCRM, ASCSM, and ASCPEM Specifications

The current specification document refers to the ASCMM, ASCRM, ASCSM, and ASCPEM specifications via OCL operations relying on SPMS specifications:

- Occurrences are identified by; <pattern>.A_instanceOf_PatternInstance::PatternInstance().
  For example, with *ASCMM-MNT-1*: ASCMM:ASCMMLibrary::ASCMM-MNT 1.A_instanceOf_PatternInstance::PatternInstance()
- Languages of code elements implementing the occurrence are identified by; <pattern>.A_instanceOf_PatternInstance::PatternInstance().fulfillments().fulfilledBy().source(). language().
  For example, with *ASCMM-MNT-1*: ASCMM:ASCMMLibrary::ASCMM-MNT-1.A_instanceOf_PatternInstance::PatternInstance().fulfillments().fulfilledBy().source().language()
- Code elements implementing the occurrence roles are identified by; <role>.A_boundTo_Binding::Binding().fulfilledBy().
  For example, with *ASCMM-MNT-1-roles-controlFlowJumStatement*: ASCMM:ASCMMLibrary::ASCMM-MNT-1-roles-controlFlowJumpStatement.A_boundTo_Binding::Binding().fulfilledBy()

6.2 Qualification Measures

Qualification measures describe attributes of the software context affecting an occurrence that can cause variation in the time required to remediate the specific occurrence. The contextual attributes quantified in qualification measures include complexity, concentration, evolution status, exposure, and technological diversity. In this specification, qualification measures related to pattern occurrences are used as follows:

- They are measures available for use in analyzing, interpreting, and using Technical-Debt values in making decisions, benchmarking, modeling, and other uses to which Technical Debt values may be put. For instance, when prioritizing the remediation of an occurrence of a source code pattern, the context surrounding the occurrence influences the assessment of:
  - the operational risk associated with not removing the occurrence,
  - the destabilization risk associated with removing the occurrence,

- the opportunity to reduce costs by removing many occurrences at the same time, or freshly created occurrences, and
  - the organizational risk associated with the synchronization of different teams to handle complex occurrences involving different technologies.
- They are measures available for use in computing an adjustment factor for the remediation effort of each occurrence that account for attributes of the software context in which the occurrence resides. For instance, when remediating an occurrence resides. For instance, when remediating an occurrence of a source code pattern, the required effort is impacted by the complexity of the code elements implementing the ocurrence, their connectedness to other code elements in the software, the number of languages in the occurrence's implementation, etc.

Therefore, along with the identifying occurrences of source code patterns, the measurement of the Technical Debt will include for each occurrence the following measures:

- **Complexity** - of code elements, measured by the Effort Complexity, as defined in the Automated Enhancement Points (AEP) specification.
- **Exposure** - of code elements propagating effects of the occurrence to the rest of the software. Based on the extent of propagation, remediating the occurrence could involve direct references to code elements (measured as the code elements' number of distinct direct callers), or indirect references (measured as the number of distinct call paths leading to the code elements).
- **Technological diversity** - the number of the languages in which elements in the source code pattern of a specific occurrence are instantiated.
- **Concentration** - total number of occurrences of any source code patterns within a single code element (e.g., class, module, component, subroutine, etc.).
- **Evolution status** - changes and evolution both of code elements in the occurrence and of code elements constituting the immediate software environment within which the occurrence is embedded.

In the context of patterns which rely on roles that model values and threshold values that are not to be exceeded, the gap size for each pattern occurrence shall be collected and measured as the difference between the values and the threshold values.

These measures are included in the specification for Technical Debt to provide standard measures for use in interpreting Technical Debt information. Although organizations may develop their own interpretive measures, the use of these interpretive measures relieves an organization from having to develop its own proprietary adjustment formulas and provides standards for benchmarking adjusted values of Technical Debt. Expected benefits from using qualification measures include the following:

- **Complexity** - ability to discriminate between situations where the remediation of Technical Debt Items can lead to additional costs due to the over-complexity of the fix.
- **Exposure** - ability to discriminate between situations where the remediation of Technical Debt Items can lead to additional costs due to the nature and location of the fix. To serve as a risk warning indicator when assessing or monitoring the Technical Debt. To provide a priority setting guide (e.g., prioritizing Technical Debt Items with high exposure for remediation at the beginning of a release to provide time to ensure detection of side-effects, while scheduling Technical Debt Items with low exposure at the end of a release to minimize risk of destabilizing the software).
- **Technological diversity** - ability to identify situations where, because of the need to involve and coordinate multiple individuals or teams with different knowledge and skills, remediation effort could increase dramatically.
- **Concentration** - ability to identify concentrations of Technical Debt Items in the same classes, components, etc. where remediation effort can be optimized (e.g., re-engineering code elements that are rife with Technical Debt Items wherein effort spend understanding, testing, etc. can be shared across Technical Debt Items).

- **Evolution Status** - ability to identify changes and evolution in the code elements in which Technical Debt Items are embedded that allow some optimization for remediating one or more occurrences (e.g., target items in code elements that are being evolved, to share and reduce the total effort to understand them and test them).

6.3 Contextual Technical Debt Measure (CTDM)

Some organizations may want to customize how the Automated Technical Debt Measure (ATDM) calculation to reflect local conditions or practices. Such customizations may exclude some source code patterns from the calculation or adjust the default values for remediation effort. These adjustments can be made for either the entire organization or for individual applications. Customized calculations shall be designated as a Contextual Technical Debt Measures (CTDM) to distinguish them from the standard calculation (ATDM) which can be used for benchmarking with other organizations or datasets.

This page intentionally left blank.

7 Automated Technical Debt Measure Specification (normative)

7.1 Computing Process Overview

7.1.1 Automated Technical Debt Measure (ATDM)

The Automated Technical Debt Measures (ATDM) shall be calculated through the following process:

1. Collect source code for one or two revisions of the software.
2. Generate the application model for available revision(s), taking care of the evolveTo/evolveFrom relationships between code elements when there are two revisions.
3. Detect occurrences of the Source Code Patterns enumerated in *ASCRM*, *ASCSM*, *ASCPEM*, and *ASCMM*.
4. Compute the unadjusted remediation effort for each occurrence, as:
    a. A pattern-dependent constant, when the pattern only relies on the existence of code elements and relationships.
    b. A pattern-dependent constant multiplied by the difference between measured value(s) and required threshold value(s), when the pattern relies on value(s) exceeding threshold(s).
5. Collect qualification information for each occurrence, i.e., technological diversity, complexity, concentration, exposure, and evolution status (only when two revisions of the software were processed in steps 1, 2, and 3).
    a. Technological diversity is the count of programming languages in use in the implementation code elements of an occurrence.
    b. Complexity is the Effort Complexity from the Automated Enhancement Points (AEP) specification.
    c. Exposure is the call graph branching factor.
    d. Concentration is the number of source code pattern occurrences the implementation code elements are involved in.
    e. Evolution status requires determining when an occurrence of the code elements constituting the immediate software environment within which the occurrence is embedded have been added, removed, or updated between the measured revisions of the software.
    f. Occurrence gap size, when the pattern relies on roles that model values and threshold values that are not to be exceeded.
6. Compute an adjustment factor for each occurrence, based on qualification measures from step 5.
    a. Technological diversity is used as is.
    b. Complexity is computed as an average across the implementations of the pattern roles of complexity overhead, measured as a ratio of the complexity from step 5.c divided by the lowest complexity value the implementations could have had (i.e., complexity as defined and calculated in the Automated Enhancement Points specification).

c. Exposure is computed as an average across the implementations of pattern exposed roles of the exposure overhead, measured as a logarithmic transformation of the exposure value from step 5.c (i.e., exposure as defined and calculated in the Automated Enhancement Points specification).

d. Concentration is used as an average across the implementations of the pattern roles of the inverse of the concentration value from step 5.d.

e. Evolution status is not used in the adjustment factor.

7. Multiply the adjustment factor from step 6 to the unadjusted remediation effort from step 4 to get the remediation effort for each occurrence.

8. Sum the occurrence remediation efforts from step 7 for each pattern to calculate the pattern-specific remediation effort.

9. For each CISQ Quality Characteristic, sum the pattern remediation efforts from step 8 for source code patterns associated with that characteristic (*ASCMM*, *ASCRM*, *ASCPEM*, *ASMSM*) to compute the total remediation effort for that specific characteristic (i.e., *AMREM*, *ARREM*, *APEREM*, or *ASREM* respectively).

10. Sum the pattern remediation efforts from step 8 for source code patterns associated with all 4 CISQ Quality Characteristics (*ASCMM*, *ASCRM*, *ASCPEM*, *ASCSM*) to compute the Automated Technical Debt Measure (*ATDM*). (Note some patterns are "shared" between *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*; the associated remediation effort for such patterns will be counted only once.)

11. Sum occurrence remediation efforts from step 7 for all occurrences within a specified range of qualification measures to build distributions of the *ATDM* according to the requested range.

Figure 7.1 and Figure 7.2 visually summarize the computation formulae. They are provided for illustration and clarity purposes. However, they do not contain all the normative measure elements detailed in this clause.

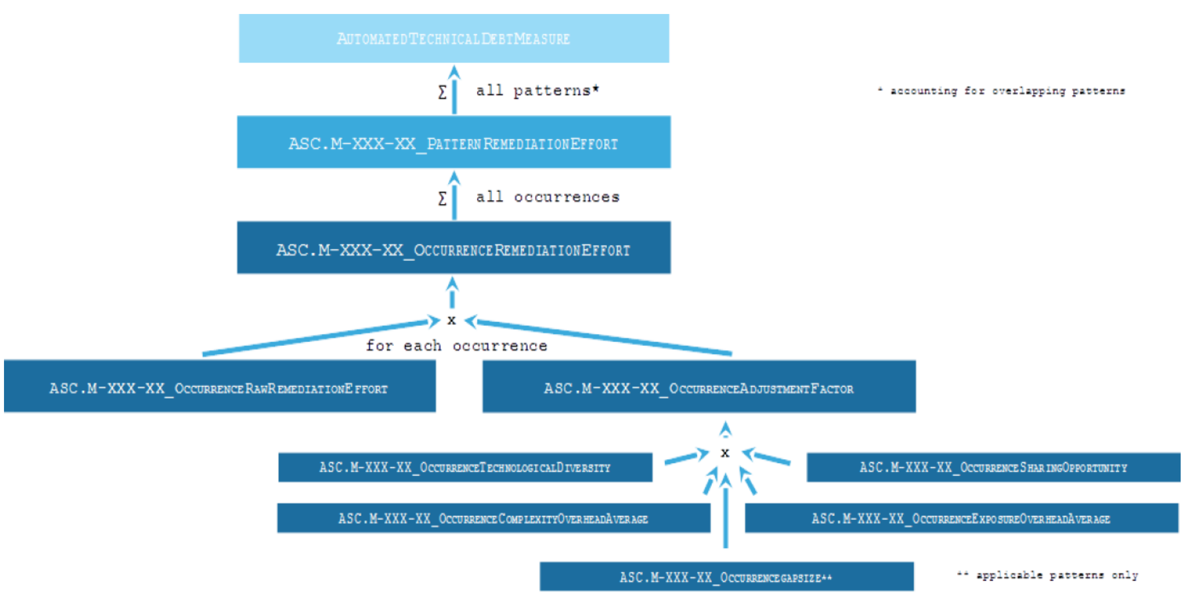

Figure 7.1 - Illustration of the ATDM computation formula

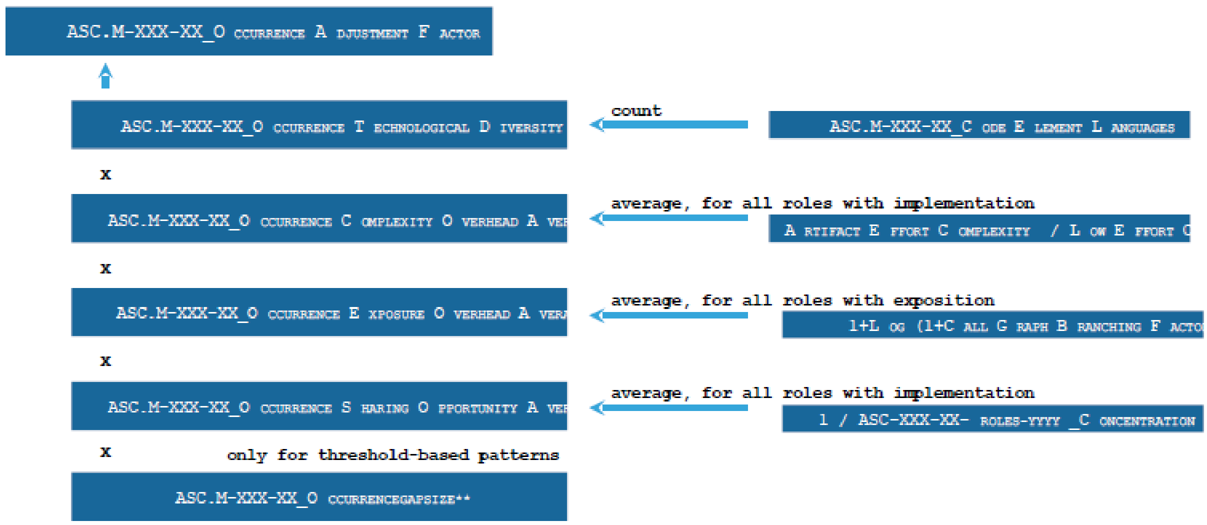

Figure 7.2 - Illustration of the Adjustment Factor computation formula

7.1.2 Contextual Technical Debt Measure (CTDM)

The process to follow to compute CTDM shall be identical to that for ATDM except for the following steps:

3. Detect occurrences of selected patterns

6. Compute a custom adjustment factor

9./10. Sum Pattern remediation effort for all selected patterns

7.2 Application Model

7.2.1 Overview

The calculation of the Automated Technical Debt Measure (ATDM) shall be performed:

- either on one revision of the software, which is called "ToRevision," or
- between two revisions of the software, which are called "FromRevision" and "ToRevision." "ToRevision" being the more recent of the two revisions.

Each available revision shall be analyzed to create an application model of the software. The application model shall be composed of:

- Computational objects in the source code and their relationships.
- Occurrences of patterns, including the binding information to the computational objects and relationships.

When both “FromRevision” and “ToRevision” revisions are available, the evolvedTo/evolvedFrom relationship shall be identified for all computational elements (i.e., to identify when code elements in “FromRevision” revision are also found in “ToRevision” revision, and shall be identified as either an evolved version of the computational object, or an unchanged version) as presented in the Structured Metrics Metamodel (SMM Clause 17.1).

7.2.2 Representation in SMM of the revision(s)

SMM enables the following modeling:

- One **smm:Observation** of collected revision(s) so that the base application model shall contain all required items.
- One **smm:ObservationScope** in this **smm:Observation** for each revision shall be used to identify items from each revision.

7.2.3 Measure Specifications

To handle the latest revision when two revisions are delivered, the analysis shall establish the following scope related entities:

- An **smm:ObservationScope**

```
<measureElement xmi:id="toRevisionMeasurementScope"
xmi:type="smm:ObservationScope" name="toRevisionMeasurementScope"
class="MOF::Element" shortDescription="Subset of the Application Model which contains
code elements from the initial revision. Code elements are related to code elements from the
final revision by evolvedTo/evolvedFrom relationships." />
```

- An **smm:OCLOperation** to easily identify a code element from the **smm:ObservationScope**

```
<measureElement xmi:type="smm:OCLOperation" xmi:id="isInLatestRevision"
name="isInLatestRevision" context="kdm:Core::Element"
body="(toRevisionMeasurementScope()-&amp;gt;includes(self))"/>
```

To handle the previous revision when two revisions are delivered, the analysis shall establish the following scope related entities:

- A second **smm:ObservationScope**

```
<measureElement xmi:id="fromRevisionMeasurementScope"
xmi:type="smm:ObservationScope" name="fromRevisionMeasurementScope"
class="MOF::Element" shortDescription="Subset of the Application Model which contains
code elements from the final revision. Code elements are related to code elements from the
initial revision by evolvedTo/evolvedFrom relationships." />
```

- A second **smm:OCLOperation** to easily identify a code element from the **smm:ObservationScope**

```
<measureElement xmi:type="smm:OCLOperation" xmi:id="isInPreviousRevision"
name="isInPreviousRevision" context="kdm:Core::Element"
body="(fromRevisionMeasurementScope()-&amp;gt;includes(self))"/>
```

7.3 Quantification of Remediation Effort at the Pattern Occurrence Level

This sub clause describes the steps that shall be used to compute the remediation effort measures of a given source code pattern occurrence (Technical Debt item) in a specific revision of the software.

For each pattern occurrence, in each revision, the effort (coding, unit/non-regression testing adaptation) to remediate it shall be computed as a calculation conforming to the following process:

1. identify occurrences
2. get "unadjusted" remediation effort configuration
3. collect qualification information
4. compute adjustment factor
5. compute "adjusted" remediation effort

7.3.1 Occurrence Identification

For each pattern, identify each individual occurrence thanks to an **smm:Scope** relying on an **smm:Operation** to use as a scope recognizer. These items are demonstrated with the *ASCRM-CWE-120* pattern as follows:

- An **smm:Scope**

```
<measureElement xmi:id="ASCRM-CWE-120_Occurrence"
xmi:type="smm:Scope"
name="ASCRM-CWE-120_Occurrence"
class="SPMS:Observations::PatternInstance"
recognizer="ASCRM-CWE-120_Occurrence_Recognizer" />
```

- defined by an OCL **smm:Operation**

```
<measureElement xmi:id="ASCRM-CWE-120_Occurrence_Recognizer"
xmi:type="smm:Operation"
name="ASCRM-CWE-120_Occurrence_Recognizer"
language="OCL"
body="ASCRM:ASCRMLibrary::ASCRM-CWE-
120.A_instanceOf_PatternInstance::PatternInstance()" />
```

Figure 7.3 illustrates the SMM modeling with ASCRM-CWE-120 pattern.

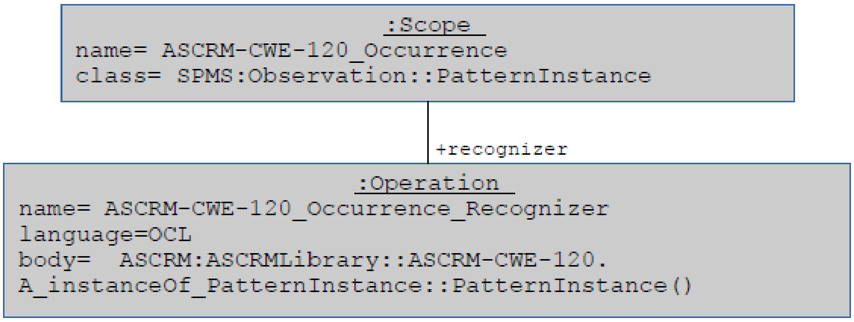

Figure 7.3 - ASCRM-CWE-120 occurrence identification with SMM Scope and Recognizer

Measure Specifications

An **smm:Scope** measure (named as the pattern key with an '_Occurrence' suffix) and its **smm:Operation** recognizer (named as the pattern key with an '_Occurrence_Recognizer' suffix) shall be defined for each source code pattern from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*, as illustrated with the *ASCRM-CWE-120* pattern above.

7.3.2 Unadjusted Remediation Effort Configuration

This paragraph describes the steps that shall be used to get the remediation effort measure of a given occurrence of a source code pattern (Technical Debt Item) in a given revision of the software, unadjusted by qualification information about the occurrence.

For each occurrence in each revision, the effort (coding, unit/non-regression testing adaptation) to remediate the occurrence shall be determined as follows.

The unadjusted remediation effort shall be the remediation effort assigned to the source code pattern. The occurrence remediation effort shall be modeled as an **smm:DirectMeasure** using an **smm:Operation** relying on a formula which uses a parameter to handle the remediation effort amount.

These rules are demonstrated with the *ASCRM-CWE-120* pattern as follows:

- An **smm:DirectMeasure**

```
<measureElement xmi:type="smm:DirectMeasure"
xmi:id="ASCRM-CWE-120_OccurrenceUnadjustedRemediationEffort"
name="ASCRM-CWE-120_OccurrenceUnadjustedRemediationEffort"
unit="effort(minutes)"
trait="RemediationEffortEstimating"
scope="softwareMeasurementScope"
shortDescription="Effort to remove one occurrence of ASCRM-CWE-120 pattern"
operation="ASCRM-CWE-120_OccurrenceUnadjustedRemediationEffort_Value" />
```

- defined by an OCL **smm:Operation**

  <measureElement
  xmi:id="ASCRM-CWE-120_OccurrenceUnadjustedRemediationEffort_Value"
  xmi:type="smm:Operation"
  name="ASCRM-CWE-120_OccurrenceUnadjustedRemediationEffort_Value"
  language="OCL"
  body="Real { ASCRM-CWE-120_OccurrenceUnadjustedRemediationEffort_Value_OccurrenceRemovalEffortInMinutes = 20 }"
  trait="RemediationEffortEstimating"/>

Figure 7.4 illustrates the SMM modeling with *ASCRM-CWE-120* pattern.

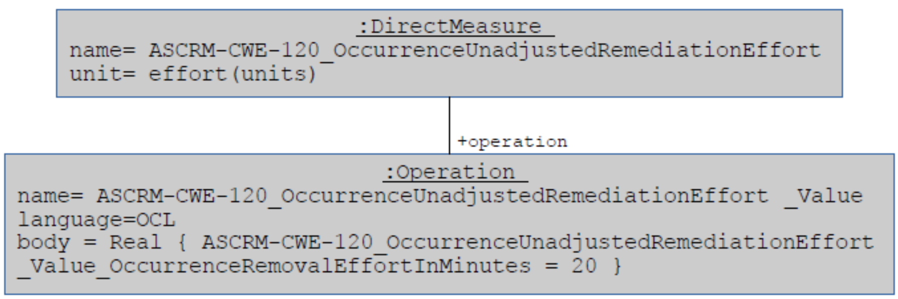

Figure 7.4 - ASCRM-CWE-120 remediation effort configuration access with SMM DirectMeasure and Operation

Measure specifications

An **smm:DirectMeasure** measure (named as the pattern key with a '_OccurrenceUnadjustedRemediationEffort' suffix) and its **smm:Operation** (named as the pattern key with a '_OccurrenceUnadjustedRemediationEffort_Value' suffix) shall be defined for each source code pattern from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*, as illustrated with the *ASCRM-CWE-120* pattern above.

The default values are listed in Clause 7.7.

7.3.3 Qualification of Pattern Occurrences

This sub clause describes the steps that shall be used to compute qualification measures that can be applied to each individual source code pattern occurrence.

These qualification measures are an integral part of the calculation of Technical Debt, via the adjustment factor detailed in 7.3.4. These measures can also be used in analyzing, interpreting, and using Technical-Debt values for making decisions, benchmarking, modeling, and other uses.

The measurement process shall include two sets of scopes:

- The code elements from the role implementations of each occurrence.
- The languages in which code elements were implemented, from the role implementations of each occurrence.

Then, the measurement process shall compute the following qualification measures:

- Technological diversity, using the language-related scopes.
- Complexity, Exposure, Concentration, and Evolution statuses, using the code-elements-related scopes.

Last, when applicable, the measurement process shall compute the occurrence gap size.

7.3.3.1 Occurrence implementation code elements

An **smm:Scope** (named as the role name with a '_CodeElements' suffix), and its recognizer **smm:Operation** (named as the role name with a '_CodeElements_Recognizer' suffix) shall be defined for each applicable Role (listed below) in a source code pattern from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*, as follows.

For example, with *ASCRM-CWE-120-roles-targetTransformationSequence*:

- an **smm:Scope**

```
<measureElement
xmi:id="ASCRM-CWE-120-roles-targetTransformationSequence_CodeElements"
name="ASCRM-CWE-120-roles-targetTransformationSequence_CodeElements"
xmi:type="smm:Scope"
class="kdm:Code::AbstractCodeElement"
operation="ASCRM-CWE-120-roles-
targetTransformationSequence_CodeElements_Recognizer" />
```

- relying on an **smm:Operation**

```
<measureElement
xmi:id="ASCRM-CWE-120-roles-targetTransformationSequence_CodeElements_Recognizer"
name="ASCRM-CWE-120-roles-targetTransformationSequence_CodeElements_Recognizer"

xmi:type="smm:Operation"
language="OCL" body="ASCRM:ASCRMLibrary::ASCRM-CWE-120-roles-
targetTransformationSequence.A_boundTo_Binding::Binding().fulfilledBy()"/>
```

Figure 7.5 illustrates the SMM modeling with ASCRM-CWE-120 roles targetTransformationSequence role.

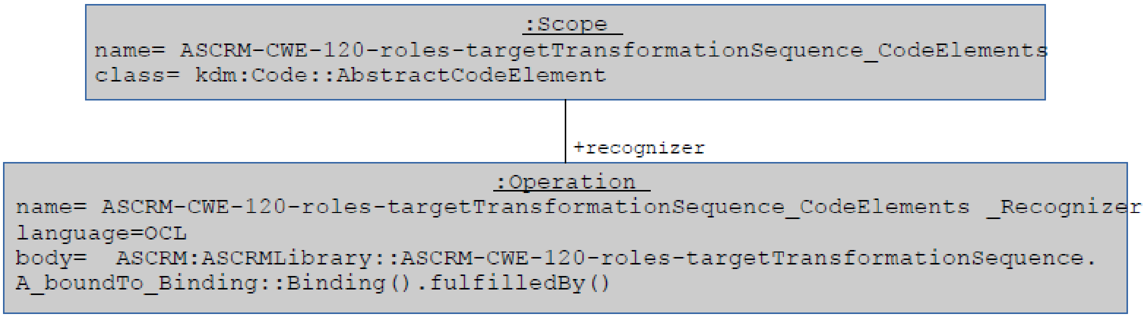

Figure 7.5 - ASCRM-CWE-120-roles-targetTransformationSequence role implementation identification with SMM Scope and Recognizer

Measure specifications

An **smm:Scope** measure (named as the role key with a '_CodeElements' suffix) and its **smm:Operation** recognizer (named as the pattern key with a '_CodeElements_Recognizer' suffix) shall be defined for each applicable role from source code pattern from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*, as illustrated with the *ASCRM-CWE-120-roles-targetTransformationSequence* pattern above.

Applicable roles are:

- *ASCMM*
    - ASCMM-MNT-1-roles-controlFlowJumpStatement
    - ASCMM-MNT-1-roles-switchBranching
    - ASCMM-MNT-2-roles-class
    - ASCMM-MNT-3-roles-valueElement
    - ASCMM-MNT-3-roles-initialisationStatement
    - ASCMM-MNT-4-roles-controlElement
    - ASCMM-MNT-5-roles-loopElement
    - ASCMM-MNT-5-roles-updateStatement
    - ASCMM-MNT-6-roles-controlElement
    - ASCMM-MNT-7-roles-module
    - ASCMM-MNT-7-roles-moduleDependencyCycle
    - ASCMM-MNT-8-roles-file
    - ASCMM-MNT-10-roles-controlElement
    - ASCMM-MNT-11-roles-controlElement
    - ASCMM-MNT-12-roles-callerObject

- ASCMM-MNT-12-roles-calleeObject
- ASCMM-MNT-13-roles-controlElement
- ASCMM-MNT-14-roles-controlElement
- ASCMM-MNT-15-roles-publicDataElement
- ASCMM-MNT-15-roles-dataElementDeclarationStatement
- ASCMM-MNT-16-roles-class1
- ASCMM-MNT-16-roles-class2
- ASCMM-MNT-16-roles-field
- ASCMM-MNT-17-roles-class
- ASCMM-MNT-18-roles-class
- ASCMM-MNT-19-roles-controlElement1
- ASCMM-MNT-19-roles-controlElement2
- ASCMM-MNT-20-roles-controlElement

- *ASCRM*
  - ASCRM-CWE-397-roles-controlElement
  - ASCRM-CWE-397-roles-throwsAction
  - ASCRM-CWE-397-roles-thrownExceptionParameter
  - ASCRM-CWE-396-roles-controlElement
  - ASCRM-CWE-396-roles-catchElement
  - ASCRM-CWE-396-roles-caughtExceptionParameter
  - ASCRM-CWE-456-roles-dataElement
  - ASCRM-CWE-456-roles-declarationStatement
  - ASCRM-CWE-456-roles-evaluationStatement
  - ASCRM-CWE-704-roles-dataElement
  - ASCRM-CWE-704-roles-dataElementDeclarationStatement
  - ASCRM-CWE-704-roles-typeCastExpression
  - ASCRM-CWE-772-roles-platformResource
  - ASCRM-CWE-772-roles-ResourceAllocationStatement
  - ASCRM-CWE-772-roles-transformationSequence
  - ASCRM-CWE-120-roles-sourceBufferAllocationStatement
  - ASCRM-CWE-120-roles-targetBufferAllocationStatement
  - ASCRM-CWE-120-roles-sourceTransformationSequence

- ASCRM-CWE-120-roles-targetTransformationSequence
- ASCRM-CWE-120-roles-moveBufferStatement
- ASCRM-RLB-1-roles-controlElement
- ASCRM-RLB-1-roles-exceptionHandlingBlock
- ASCRM-CWE-252-data-roles-controlElement
- ASCRM-CWE-252-data-roles-sQLStatement
- ASCRM-CWE-252-data-roles-executeSQLStatement
- ASCRM-RLB-2-roles-serializableStorableDataElement
- ASCRM-RLB-2-roles- controlElementList
- ASCRM-RLB-3-roles-serializableStorableDataElement
- ASCRM-RLB-3-roles-nonSerializableItem
- ASCRM-RLB-4-roles-persistantStorableDataElement
- ASCRM-RLB-5-roles-lowLevelResourceManagementAPIList
- ASCRM-RLB-6-roles-dataElement
- ASCRM-RLB-6-roles-childPointerDataElement
- ASCRM-RLB-7-roles-class
- ASCRM-RLB-7-roles-selfDestructionControlElement
- ASCRM-RLB-8-roles-controlElement
- ASCRM-RLB-8-roles-variableNumberOfParameterSyntax
- ASCRM-CWE-252-resource-roles-controlElement
- ASCRM-CWE-252-resource-roles-resourceAccessStatement
- ASCRM-RLB-9-roles-comparisonStatement
- ASCRM-CWE-788-roles-valueElement
- ASCRM-CWE-788-roles-buffer
- ASCRM-CWE-788-roles-bufferReferenceStatement
- ASCRM-CWE-788-roles-bufferAllocationStatement
- ASCRM-CWE-788-roles-transformationSequence
- ASCRM-RLB-10-roles-controlElement
- ASCRM-RLB-10-roles-dataAccessStatement
- ASCRM-RLB-11-roles-controlElement
- ASCRM-RLB-11-roles-nonFinalStaticField
- ASCRM-RLB-12-roles-singletonClass

- ASCRM-RLB-12-roles-instanciationStatement
- ASCRM-RLB-13-roles-module
- ASCRM-RLB-13-roles-moduleDependencyCycle
- ASCRM-RLB-14-roles-parentClass
- ASCRM-RLB-14-roles-childClass
- ASCRM-RLB-14-roles-referenceStatement
- ASCRM-RLB-14-roles-class
- ASCRM-RLB-15-roles-virtualMethod
- ASCRM-RLB-16-roles-parentClass
- ASCRM-RLB-16-roles-childClass
- ASCRM-RLB-17-roles-parentClass
- ASCRM-RLB-17-roles-childClass
- ASCRM-RLB-17-roles-parentVirtualDestructor
- ASCRM-RLB-18-roles-dataElement
- ASCRM-RLB-18-roles-initialisationStatement
- ASCRM-RLB-18-roles-networdResourceIdentificationValue
- ASCRM-RLB-19-roles-synchronousCallInstruction
- ASCRM-CWE-674-roles-controlElement
- ASCRM-CWE-674-roles-recursiveExecutionPath

- *ASCSM*
  - ASCSM-CWE-120-roles-sourceBufferAllocationStatement
  - ASCSM-CWE-120-roles-targetBufferAllocationStatement
  - ASCSM-CWE-120-roles-sourceTransformationSequence
  - ASCSM-CWE-120-roles-targetTransformationSequence
  - ASCSM-CWE-120-roles-moveBufferStatement
  - ASCSM-CWE-129-roles-userInput
  - ASCSM-CWE-129-roles-arrayAccessStatement
  - ASCSM-CWE-129-roles-array
  - ASCSM-CWE-129-roles-transformationSequence
  - ASCSM-CWE-134-roles-userInput
  - ASCSM-CWE-134-roles-formatStatement

- ASCSM-CWE-134-roles-transformationSequence
- ASCSM-CWE-22-roles-userInput
- ASCSM-CWE-22-roles-pathCreationStatement
- ASCSM-CWE-22-roles-transformationSequence
- ASCSM-CWE-252-resource-roles-controlElement
- ASCSM-CWE-252-resource-roles-resourceAccessStatement
- ASCSM-CWE-327-roles-cryptographicDeployedComponentInUse
- ASCSM-CWE-396-roles-controlElement
- ASCSM-CWE-396-roles-catchElement
- ASCSM-CWE-396-roles-caughtExceptionParameter
- ASCSM-CWE-397-roles-controlElement
- ASCSM-CWE-397-roles-throwsAction
- ASCSM-CWE-397-roles-thrownExceptionParameter
- ASCSM-CWE-434-roles-userInput
- ASCSM-CWE-434-roles-transformationSequence
- ASCSM-CWE-434-roles-fileUploadStatement
- ASCSM-CWE-456-roles-dataElement
- ASCSM-CWE-456-roles-declarationStatement
- ASCSM-CWE-456-roles-evaluationStatement
- ASCSM-CWE-606-roles-userInput
- ASCSM-CWE-606-roles-loopConditionStatement
- ASCSM-CWE-606-roles-transformationSequence
- ASCSM-CWE-667-roles-publicDataElement
- ASCSM-CWE-667-roles-dataElementDeclartionStatement
- ASCSM-CWE-667-roles-dataElementAccessStatement
- ASCSM-CWE-672-roles-platformResource
- ASCSM-CWE-672-roles-resourceReleaseStatement
- ASCSM-CWE-672-roles-transportSequence
- ASCSM-CWE-672-roles-resourceAccessStatement
- ASCSM-CWE-681-roles-dataElement
- ASCSM-CWE-681-roles-dataElementDeclarationStatement
- ASCSM-CWE-681-roles-numericalDataType

- ASCSM-CWE-681-roles-typeCastExpression
- ASCSM-CWE-681-roles-targetDataType
- ASCSM-CWE-99-roles-userInput
- ASCSM-CWE-99-roles-accessByNameStatement
- ASCSM-CWE-99-roles-transformationSequence
- ASCSM-CWE-772-roles-platformResource
- ASCSM-CWE-772-roles-resourceAllocationStatement
- ASCSM-CWE-772-roles-transformationSequence
- ASCSM-CWE-78-roles-userInput
- ASCSM-CWE-78-roles-executeRunTimeCommandStatement
- ASCSM-CWE-78-roles-transformationSequence
- ASCSM-CWE-789-roles-userInput
- ASCSM-CWE-789-roles-bufferAccessStatement
- ASCSM-CWE-789-roles-transformationSequence
- ASCSM-CWE-789-roles-bufferAllocationStatement
- ASCSM-CWE-79-roles-userInput
- ASCSM-CWE-79-roles-userDisplay
- ASCSM-CWE-79-roles-transformationSequence
- ASCSM-CWE-798-roles-initialisationStatement
- ASCSM-CWE-798-roles-authenticationStatement
- ASCSM-CWE-798-roles-transportSequence
- ASCSM-CWE-835-roles-controlElement
- ASCSM-CWE-835-roles-recursiveExecutionPath
- ASCSM-CWE-89-roles-userInput
- ASCSM-CWE-89-roles-sQLCompilationStatement
- ASCSM-CWE-89-roles-transformationSequence

- *ASCPEM*
  - ASCPEM-PRF-1-roles-staticBlock
  - ASCPEM-PRF-1-roles-initialisationStatement
  - ASCPEM-PRF-2-roles-controlElement
  - ASCPEM-PRF-2-roles-stringConcatenuationStatement

- ASCPEM-PRF-3-roles-staticField
- ASCPEM-PRF-3-roles-parentClass
- ASCPEM-PRF-4-roles-dataTable
- ASCPEM-PRF-4-roles-queryStatement
- ASCPEM-PRF-5-roles-selectSQLStatement
- ASCPEM-PRF-6-roles-dataTable
- ASCPEM-PRF-7-roles-dataTable
- ASCPEM-PRF-7-roles-index
- ASCPEM-PRF-8-roles-loopStatement
- ASCPEM-PRF-8-roles-expensiveOpertion
- ASCPEM-PRF-8-roles-executionPath
- ASCPEM-PRF-9-roles-controlElement
- ASCPEM-PRF-10-roles-controlElement
- ASCPEM-PRF-11-roles-controlElement
- ASCPEM-PRF-11-roles-sQLStatement
- ASCPEM-PRF-12-roles-aggregatingDataElement
- ASCPEM-PRF-13-roles-controlElement
- ASCPEM-PRF-13-roles-sQLConnectionInitializationStatement
- ASCPEM-PRF-14-roles-memoryAllocationStatement
- ASCPEM-PRF-14-roles-transformationSequence
- ASCPEM-PRF-15-roles-methodElement
- ASCPEM-PRF-15-roles-referenceStatement
- ASCPEM-PRF-15-roles-referencedObject

7.3.3.2 Occurrence implementation languages

The set of languages in which a single pattern occurrence has been implemented shall be determined through the following process:

1. For each occurrence, list implementation code elements, regardless of the role.
2. For each code element, list the source region(s).
3. For each source region, collect the language attribute value.

An **smm:Scope** (named as the pattern name with a '_CodeElementLanguages' suffix), and its recognizer **smm:Operation** (named as the pattern name with a '_CodeElementLanguages_Recognizer' suffix) shall be defined for each pattern.

For example, with *ASCRM-CWE-120*:

- an **smm:Scope**

  **<measureElement xmi:id="ASCRM-CWE-120_CodeElementLanguages"
  xmi:type="smm:Scope"
  name="ASCRM-CWE-120_CodeElementLanguages"
  class="MOF::Element"
  recognizer="ASCRM-CWE-120_CodeElementLanguages_Recognizer" />**

- relying on an **smm:Operation**

  **<measureElement xmi:id="ASCRM-CWE-120_CodeElementLanguages_Recognizer"
  xmi:type="smm:Operation"
  name="ASCRM-CWE-120_CodeElementLanguages_Recognizer"
  language="OCL"
  body="ASCRM:ASCRMLibrary::ASCRM-CWE-120.A_instanceOf_PatternInstance::PatternInstance().fulfillments().fulfilledBy().source().language()" />**

Figure 7.6 illustrates the SMM modeling with *ASCRM-CWE-120* pattern.

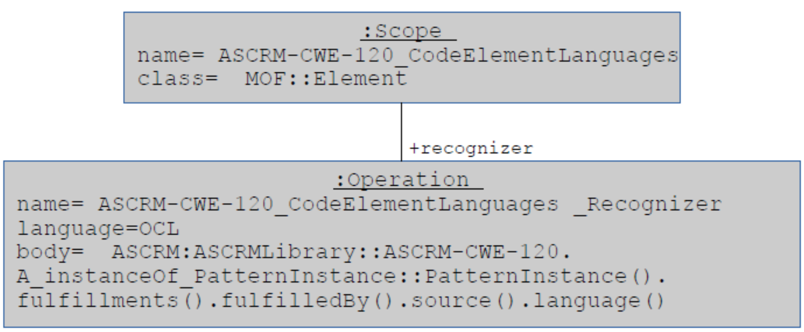

Figure 7.6 - ASCRM-CWE-120 occurrence languages identification with SMM Scope and Recognizer

Measure specifications

An **smm:Scope** measure (named as the role key with a '_CodeElementLanguages' suffix) and its **smm:Operation** recognizer (named recognizer (named as the pattern key with a '_CodeElementLanguages_Recognizer' suffix) shall be defined for each source code pattern from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*, as illustrated with the *ASCRM-CWE-120* pattern above.

7.3.3.3 Technological Diversity

Technological Diversity is the number of distinct languages in which the code elements of a single occurrence of a source code pattern are written, and shall computed as a simple counting applied to the occurrence implementation languages scopes.

For example, with *ASCRM-CWE-120*:

- an **smm:Counting** measure

  <measureElement xmi:type="smm:Counting"
  xmi:id="ASCRM-CWE-120_OccurrenceTechnologicalDiversity"
  name="ASCRM-CWE-120_OccurrenceTechnologicalDiversity"
  unit="Integer"
  scope="ASCRM-CWE-120_CodeElementLanguages"
  trait="LanguageCounting"
  category="FunctionalMetrics"
  shortDescription="Technological diversity of an occurrence of ASCRM-CWE-120 pattern, measured as the number of distinct languages" />

Figure 7.7 enriches Figure 7.6 and illustrates the SMM modeling with *ASCRM-CWE-120* pattern.

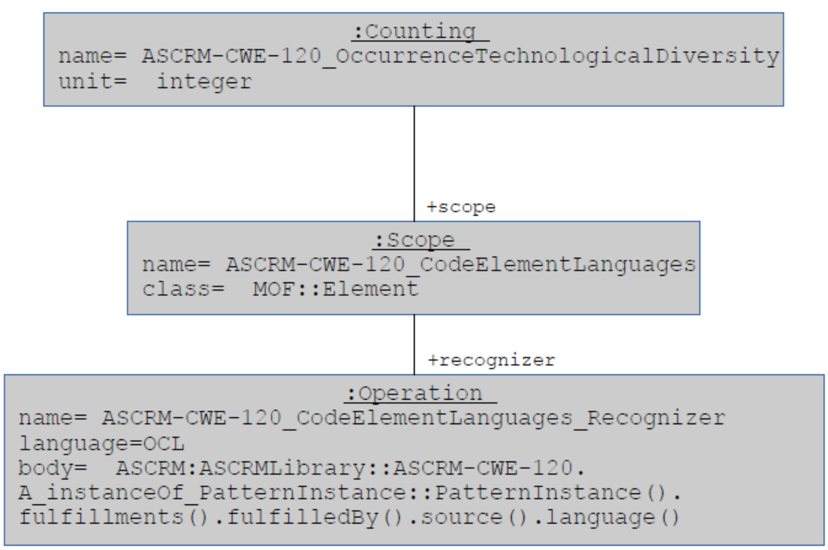

Figure 7.7 - ASCRM-CWE-120 occurrence languages identification with SMM Scope and Recognizer

Measure specifications

An **smm:Counting** measure (named as the pattern key with a '_OccurrenceTechnologicalDiversity' suffix) shall be defined for each source code pattern from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*, as illustrated with the *ASCRM-CWE-120* pattern above.

7.3.3.4 Complexity

Complexity – or Effort Complexity – shall be measured as defined in Automated Enhancement Points specifications, via an **smm:NamedMeasure**.

```
<measureElement xmi:type="smm:NamedMeasure"
xmi:id="ArtifactEffortComplexity"
name="ArtifactEffortComplexity"
unit="ImplementationPoint"
scope="AEP::Artifact"
trait="ImplementationComplexity"
formula="AEP::ArtifactEffortComplexity"
shortDescription="Code Element Effort Complexity according to AEP 1.0 specifications" />
```

aep.aep::Artifact is a subset of **kdm:code::ControlElement** and this measure will return non-null values for elements of this subset only.

To compute the Complexity overhead which contributes to the Adjustment Factor, the Low Complexity Effort value shall also be collected via a second **smm:NamedMeasure**. This is the lowest complexity value the implementation code elements could have had, considered to be the “best case scenario.”

```
<measureElement xmi:type="smm:NamedMeasure"
xmi:id="LowEffortComplexity"
name="LowEffortComplexity"
unit="ImplementationPoint"
scope="AEP::Artifact"
trait="ImplementationComplexity"
formula="AEP::wLowEC"
shortDescription="Code Element lowest Effort Complexity value according to AEP 1.0
specifications" />
```

For each implementation role, the ratio of the two above values defines a complexity overhead, via an smm:RatioMeasure.

For example, with *ASCRM-CWE-120-roles-targetTransformationSequence*:

```
<measureElement
xmi:id="ASCRM-CWE-120-roles-targetTransformationSequence_ComplexityOverhead"
name="ASCRM-CWE-120-roles-targetTransformationSequence_ComplexityOverhead"
xmi:type="smm:RatioMeasure"
unit="Real"
trait="ComplexityEstimating"
scope="ASCRM-CWE-120-roles-targetTransformationSequence_CodeElements"
shortDescription="Complexity overhead of code elements from ASCRM-CWE-120-roles-
targetTransformationSequence_ComplexityOverhead role, measured as their Effort Complexity
divided by the minimal Effort Complexity they could have" />
```

Figure 7.8 enriches Figure 7.5 and illustrates the SMM modeling with *ASCRM-CWE-120-roles-targetTransformationSequence* pattern.

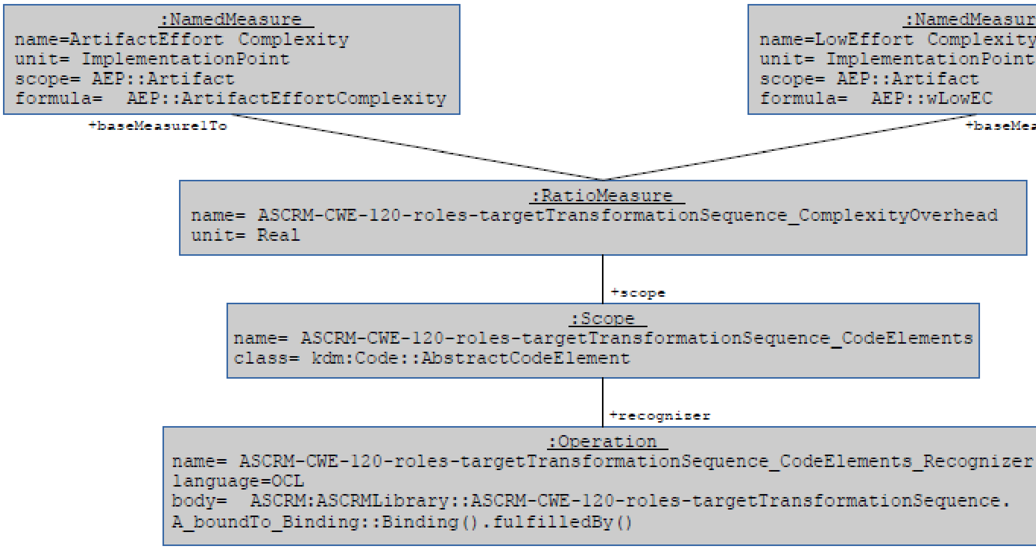

Figure 7.8 - ASCRM-CWE-120-roles-targetTransformationSequence role complexity overhead computation with SMM NamedMeasures, RatioMeasures, Scope, and Recognizer

Measure specifications

An **smm:RatioMeasure** measure (named as the role key with a '_ComplexityOverhead' suffix) shall be defined for each implementation role from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM* patterns, as illustrated with the *ASCRM-CWE-120-roles-targetTransformationSequence* role above.

7.3.3.5 Exposure

To measure exposure for all applicable source code pattern occurrences, the code element to be evaluated shall be determined by identifying the exposed role. The list of exposed pattern roles is only a subset of the list of implementation roles above.

Applicable roles are:

- *ASCSM*
    - ASCSM-CWE-120-roles-moveBufferStatement
    - ASCSM-CWE-129-roles-userInput
    - ASCSM-CWE-134-roles-userInput
    - ASCSM-CWE-22-roles-userInput
    - ASCSM-CWE-252-resource-roles-resourceAccessStatement
    - ASCSM-CWE-397-roles-controlElement

- ASCSM-CWE-434-roles-userInput
- ASCSM-CWE-456-roles-evaluationStatement
- ASCSM-CWE-606-roles-userInput
- ASCSM-CWE-667-roles-dataElementAccessStatement
- ASCSM-CWE-672-roles-resourceAccessStatement
- ASCSM-CWE-681-roles-typeCastExpression
- ASCSM-CWE-99-roles-userInput
- ASCSM-CWE-672-roles-resourceAccessStatement
- ASCSM-CWE-681-roles-typeCastExpression
- ASCSM-CWE-99-roles-userInput
- ASCSM-CWE-772-roles-resourceAllocationStatement
- ASCSM-CWE-78-roles-userInput
- ASCSM-CWE-789-roles-userInput
- ASCSM-CWE-79-roles-userInput
- ASCSM-CWE-798-roles-authenticationStatement
- ASCSM-CWE-835-roles-controlElement
- ASCSM-CWE-89-roles-userInput
- ASCSM-CWE-327-roles-cryptographicDeployedComponentInUse
- ASCSM-CWE-396-roles-controlElement

- *ASCRM*
  - ASCRM-CWE-397-roles-controlElement
  - ASCRM-CWE-396-roles-controlElement
  - ASCRM-CWE-456-roles-evaluationStatement
  - ASCRM-CWE-704-roles-typeCastExpression
  - ASCRM-CWE-772-roles-resourceAllocationStatement
  - ASCRM-CWE-120-roles-moveBufferStatement
  - ASCRM-RLB-1-roles-controlElement
  - ASCRM-CWE-252-data-roles-controlElement
  - ASCRM-RLB-2-roles-serializableStorableDataElement
  - ASCRM-RLB-3-roles-serializableStorableDataElement
  - ASCRM-RLB-4-roles-persistantStorableDataElement

- ASCRM-RLB-5-roles-lowLevelResourceManagementAPIList
- ASCRM-RLB-6-roles-childPointerDataElement
- ASCRM-RLB-7-roles-class
- ASCRM-RLB-8-roles-controlElement
- ASCRM-CWE-252-resource-roles-controlElement
- ASCRM-RLB-9-roles-comparisonStatement
- ASCRM-CWE-788-roles-bufferReferenceStatement
- ASCRM-RLB-10-roles-controlElement
- ASCRM-RLB-11-roles-controlElement
- ASCRM-RLB-12-roles-singletonClass
- ASCRM-RLB-13-roles-module
- ASCRM-RLB-14-roles-parentClass
- ASCRM-RLB-15-roles-class
- ASCRM-RLB-16-roles-parentClass
- ASCRM-RLB-17-roles-childClass
- ASCRM-RLB-18-roles-initialisationStatement
- ASCRM-RLB-19-roles-synchronousCallInstruction
- ASCRM-CWE-674-roles-controlElement

- *ASCMM*
  - ASCMM-MNT-1-roles-controlFlowJumpStatement
  - ASCMM-MNT-2-roles-class
  - ASCMM-MNT-3-roles-initialisationStatement
  - ASCMM-MNT-4-roles-controlElement
  - ASCMM-MNT-5-roles-loopElement
  - ASCMM-MNT-6-roles-controlElement
  - ASCMM-MNT-7-roles-module
  - ASCMM-MNT-8-roles-file
  - ASCMM-MNT-10-roles-controlElement
  - ASCMM-MNT-11-roles-controlElement
  - ASCMM-MNT-12-roles-callerObject
  - ASCMM-MNT-13-roles-controlElement
  - ASCMM-MNT-14-roles-controlElement

- ASCMM-MNT-15-roles-dataElementDeclarationStatement
- ASCMM-MNT-16-roles-class1
- ASCMM-MNT-17-roles-class
- ASCMM-MNT-18-roles-class
- ASCMM-MNT-19-roles-controlElement1
- ASCMM-MNT-20-roles-controlElement

- *ASCPEM*
  - ASCPEM-PRF-1-roles-initialisationStatement
  - ASCPEM-PRF-2-roles-controlElement
  - ASCPEM-PRF-3-roles-parentclass
  - ASCPEM-PRF-4-roles-queryStatement
  - ASCPEM-PRF-5-roles-selectSQLStatement
  - ASCPEM-PRF-6-roles-dataTable
  - ASCPEM-PRF-7-roles-dataTable
  - ASCPEM-PRF-8-roles-loopStatement
  - ASCPEM-PRF-9-roles-controlElement
  - ASCPEM-PRF-10-roles-controlElement
  - ASCPEM-PRF-11-roles-controlElement
  - ASCPEM-PRF-12-roles-aggregatingDataElement
  - ASCPEM-PRF-13-roles-controlElement
  - ASCPEM-PRF-14-roles-memoryAllocationStatement
  - ASCPEM-PRF-15-roles-methodElement

For each pattern applicable Role, the associated **smm:Scope** (named as the role name with a '_CodeElements' suffix), and its recognizer **smm:Operation** (named as the role name with a '_CodeElements_Recognizer' suffix) will be reused in the current process.

User input exposure considerations

In case of a source code pattern relying on user input, the number of distinct callers and call paths shall be 0, but the exposure is virtually infinite as the weakness is directly exposed to the outside world. From the security standpoint, the probability for an event – a malevolent use of the entry point into the system – to happen is “1.” This shall be considered when using exposure to manage decisions or outcomes related to Technical Debt.

The affected patterns are:

- ASCSM-CWE-129
- ASCSM-CWE-134
- ASCSM-CWE-22
- ASCSM-CWE-434
- ASCSM-CWE-606
- ASCSM-CWE-99
- ASCSM-CWE-78
- ASCSM-CWE-789
- ASCSM-CWE-79
- ASCSM-CWE-89

Number of distinct direct callers

The number of distinct direct callers shall be calculated as follows:

- identify a code element,
- build the set of code elements calling it, and
- compute the size of the set.

Measure specifications

1. The set of direct callers of any code element shall be determined as follows:
   - the applicable call links shall be identified by a first **smm:OCLOperation**

     ```
     <measureElement xmi:type="smm:OCLOperation"
     xmi:id="CallingActions"
     name="CallingActions"
     context="kdm:code::AbstractCodeElement"
     body="((oclIsTypeOf(kdm:action::CallableRelations) or
     oclIsTypeOf(kdm:action::DataRelations)) and to = self)" />
     ```

   - the callers shall be identified by a second **smm:OCLOperation**

     ```
     <measureElement xmi:type="smm:OCLOperation"
     xmi:id="CallingCodeElements"
     name="CallingCodeElements"
     context="kdm:code::AbstractCodeElement"
     body="(self.CallingActions.from())" />
     ```

2. The number of distinct direct callers of any code element shall be determined as follows:

- the size of the set of callers shall be computed by an **smm:Operation**

  ```
  <measureElement xmi:type="smm:OCLOperation"
  xmi:id="CallingCodeElementsNumber"
  name="CallingCodeElementsNumber"
  context="kdm:code::AbstractCodeElement"
  body="CallingCodeElements()-&amp;gt;size()" />
  ```

3. To measure the number of distinct callers for all implementation roles, the following measures shall apply the specified **smm:Operation** to the identified exposed role; e.g., with *ASCRM-CWE-396-roles-controlElement_CodeElements*

   - an **smm:OCLOperation** uses the **smm:OCLOperation** on the **smm:Scope**

     ```
     <measureElements xmi:type="smm:DirectMeasure"
     xmi:id="ASCRM-CWE-396-roles_controlElement_DirectExposure"
     name="ASCRM-CWE-396-roles_controlElement_DirectExposure"
     unit="Integer"
     scope="ASCRM-CWE-396-roles_controlElement_CodeElements"
     trait="ExposureSizing"
     category="FunctionalMetrics"
     shortDescription="Number of direct callers to the issue from ASCRM-CWE-396 Pattern"
     operation="CallingCodeElementsNumber"  />
     ```

An **smm:DirectMeasure** measure (named as the pattern key with a '_DirectExposure' suffix) shall be defined for each exposed pattern role from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*.

Figure 7.9 enriches Figure 7.5 and illustrates the SMM modeling with *ASCRM-CWE-120-roles-targetTransformationSequence* pattern.

Figure 7.9 - ASCRM-CWE-396-roles-controlElement role direct exposure computation with SMM OCLOperations, Operation, DirectMeasure, Scope, and Recognizer

Number of distinct call paths

The number of distinct call paths shall be computed in a manner similar to the McCabe Cyclomatic Complexity formula (CC = E – N + p) as follows:

- identify a code element,
- identify the call paths towards the code element,
- compute the number of nodes,
- compute the number of entry nodes to compute the number of edges needed to cycle back to the starting code element in order that the number of components is 1,
- compute the number of edges,
- subtract the number of nodes from the sum of the number of edges and the number of entry nodes, and
- add 1 to the difference to get the number of distinct call paths.

Measure specifications

A call graph for selected code elements shall be developed using the **:OCLOperation** from the previous paragraph.

- the call graph as recursive callers, identified by a first **smm:OCLOperation**

  <measureElement xmi:type="smm:OCLOperation"
  xmi:id="CallingGraph"
  name="CallingGraph"
  context="kdm:code::AbstractCodeElement"
  body="(closure(CallingCodeElements()))" />

The number of distinct call paths of any code element shall be computed as:

- the number of nodes, computed by an **smm:DirectMeasure**

```
<measureElement xmi:id="CallingGraphNodeNumber"
name="CallingGraphNodeNumber"
xmi:type="smm:DirectMeasure"
operation="CallingGraphNodeNumber_Value"/>
```

- and its **smm:DirectOperation**

```
<measureElement xmi:id="CallingGraphNodeNumber_Value"
name="CallingGraphNodeNumber_Value"
xmi:type="smm:Operation"
language="OCL"
body="CallingGraph()-&amp;gt;select(e: kdm:code::AbstractCodeElement)-&amp;gt;size()"/>
```

- the number of entry nodes, computed by an **smm:DirectMeasure**

```
<measureElement xmi:id="CallingGraphEntryNodeNumber"
name="CallingGraphEntryNodeNumber"


xmi:type="smm:DirectMeasure"
operation="CallingGraphEntryNodeNumber_Value" />
```

- and its **smm:Operation**

```
<measureElement xmi:id="CallingGraphEntryNodeNumber_Value"
name="CallingGraphEntryNodeNumber_Value"
xmi:type="smm:Operation"
language="OCL"
body="CallingGraph()-&amp;gt;select(e: kdm:code::AbstractCodeElement |
e.CallingCodeElementsNumber = 0 )-&amp;gt;size()"/>
```

- the number of edges, computed by an **smm:DirectMeasure**

```
<measureElement xmi:id="CallingGraphEdgeNumber"
name="CallingGraphEdgeNumber"
xmi:type="smm:DirectMeasure"
operation="CallingGraphEdgeNumber_Value" />
```

- and its **smm:Operation**

```
<measureElement xmi:id="CallingGraphEdgeNumber_Value"
name="CallingGraphEdgeNumber_Value"
xmi:type="smm:Operation"
language="OCL"
```

```
body="CallingGraph()-&amp;gt;select(e1, e2: kdm:code::AbstractCodeElement |
e1.CallingAction()-&amp;gt;includes(e2))-&amp;gt;size()"/>
```

- the sum of the number of edges and the number of entry nodes, computed by a first

```
smm:BinaryMeasure
<measureElement xmi:type="smm:BinaryMeasure"
xmi:id="CallingGraphEdgeAndEntryNodeNumber"
name="CallingGraphEdgeAndEntryNodeNumber"
unit="Integer"
functor="plus"
scope="kdm:code::AbstractCodeElement"
trait="ExposureSizing"
shortDescription="Calling graph number of edges and entry nodes" />
```

- the difference of the number of nodes from edges and entry nodes, computed by a second

```
smm:BinaryMeasure
<measureElement xmi:type="smm:BinaryMeasure"
xmi:id="CallingGraphBranchingFactor"
name="CallingGraphBranchingFactor"
unit="Integer"
functor="minus"
scope="kdm:code::AbstractCodeElement"
trait="ExposureSizing"
shortDescription="Calling graph branching factor" />
```

- the number of distinct call paths, computed by an **smm:RescaledMeasure**

```
<measureElement xmi:type="smm:RescaledMeasure"
xmi:id="GraphCallPathNumber"
name="GraphCallPathNumber"
unit="Integer"
scope="kdm:code::AbstractCodeElement"
trait="ExposureSizing"
shortDescription="Number of call paths to the Code Element"
offset="1"
multiplier="1" />
```

- the logarithmic transformation of the number of distinct call paths, computed by an **smm:RescaledMeasure**

```
smm:RescaledMeasure
<measureElement xmi:type="smm:RescaledMeasure"
xmi:id="LogGraphCallPathNumber"
name="LogGraphCallPathNumber"
unit="Real"
scope="kdm:code::AbstractCodeElement"
trait="ExposureSizing"
```

```
shortDescription="Log of the number of call paths to the Code Element"
operation="log( GraphCallPathNumber )" />
```

Finally, to measure the Exposure for all applicable pattern occurrences, the following measures shall apply the specified **:RescaleMeasure** to the identified exposed role.

For example, with *ASCRM-CWE-396-roles-controlElement_CodeElements*

- an **smm:RescaledMeasure** uses the **smm:RescaledMeasure** on the **smm:Scope**

```
<measureElements xmi:type="smm:RescaledMeasure"
xmi:id="ASCRM-CWE-396-roles-controlElement_Exposure"
name="ASCRM-CWE-396-roles-controlElement_Exposure"
unit="Real"
scope="ASCRM-CWE-396-roles-controlElement_CodeElements"
trait="ExposureSizing"
category="FunctionalMetrics"
shortDescription="Exposure to the issue from ASCRM-CWE-396-roles-controlElement role,
measured as 1 plus the log of the number of call paths to them"
offset="1"
multiplier="1" />
```

An **smm:DirectMeasure** measure (named as the pattern key with a '_Exposure' suffix) shall be computed for each pattern applicable Role from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*.

Figure 7.10 and Figure 7.11 enrich Figure 7.5 and illustrate the SMM modeling with *ASCRM-CWE-120-roles-targetTransformationSequence* pattern.

Figure 7.10 - ASCRM-CWE-396-roles-controlElement role direct exposure computation with SMM OCLOperations, Operation, DirectMeasure, Scope, and Recognizer

Figure 7.11 - ASCRM-CWE-396-roles-controlElement role exposure computation with SMM OCLOperations, Operations, RescaledMeasures, BinaryMeasures, Scope, and Recognizer (part II)

7.3.3.6 Concentration

The concentration shall be computed as follows:

1. Count the number of occurrences of the any specific pattern role (e.g., with ASCSM-CWE-120-roles-moveBufferStatement)

   - defined by an **smm:DirectMeasure**

     <measureElement
     xmi:id="ASCRM-CWE-120-roles-moveBufferStatement_Concentration"
     name="ASCRM-CWE-120-roles-moveBufferStatement_Concentration"
     xmi:type="smm:DirectMeasure"
     unit="Integer"
     trait="SharingLevelEstimating"
     scope="ASCRM-CWE-120-roles-moveBufferStatement_CodeElements"
     shortDescription="Remediation sharing opportunity of code elements from ASCRM-CWE-120-roles-moveBufferStatement_Concentration role, measured as the inverse of the number of occurrences they are involved in"
     operation="NumberOfOccurrences" />

- relying on an **smm:Operation**

```
<measureElement xmi:id="NumberOfOccurrences"
name="NumberOfOccurrences"
xmi:type="smm:Operation"
language="OCL"
body="self.A_Binding_fulfilledBy::Binding()-&amp;gt;select(b: Binding |
p.A_PatternInstance_fulfillments::PatternInstance.instanceOf.isInASCMM or
p.A_PatternInstance_fulfillments::PatternInstance.instanceOf.isInASCRM or
p.A_PatternInstance_fulfillments::PatternInstance.instanceOf.isInASCPEM or
p.A_PatternInstance_fulfillments::PatternInstance.instanceOf.isInASCSM)-&amp;gt;size()"/>
```

- which uses the following four **smm:OCLOperation**

```
<measureElement xmi:type="smm:OCLOperation"
xmi:id="isInASCMM"
name="isInASCMM"
context="SPMS:Definitions::PatternDefinition"
body="Set{'ASCSM-CWE-120','ASCSM-CWE-129','ASCSM-CWE-134','ASCSM-CWE-
22','ASCSM-CWE-252-resource','ASCSM-CWE-327','ASCSM-CWE-396','ASCSM-CWE-
397','ASCSM-CWE-434','ASCSM-CWE-456','ASCSM-CWE-606','ASCSM-CWE-667','ASCSM-
CWE-672','ASCSM-CWE-681','ASCSM-CWE-99','ASCSM-CWE-772','ASCSM-CWE-78','ASCSM-
CWE-789','ASCSM-CWE-79','ASCSM-CWE-798','ASCSM-CWE-835','ASCSM-CWE-89'}-
&amp;gt;includes(self.id)" />

<measureElement xmi:type="smm:OCLOperation"
xmi:id="isInASCPEM"
name="isInASCPEM"
context="SPMS:Definitions::PatternDefinition"
body="Set{'ASCPEM-PRF-1','ASCPEM-PRF-10','ASCPEM-PRF-11','ASCPEM-PRF-
12','ASCPEM-PRF-13','ASCPEM-PRF-14','ASCPEM-PRF-15','ASCPEM-PRF-2','ASCPEM-PRF-
3','ASCPEM-PRF-4','ASCPEM-PRF-5','ASCPEM-PRF-6','ASCPEM-PRF-7','ASCPEM-PRF-
8','ASCPEM-PRF-9'}-&amp;gt;includes(self.id)" />

<measureElement xmi:type="smm:OCLOperation"
xmi:id="isInASCRM"
name="isInASCRM"
context="SPMS:Definitions::PatternDefinition"
body="Set{'ASCRM-CWE-120','ASCRM-CWE-252-data','ASCRM-CWE-252-resource','ASCRM-
CWE-396','ASCRM-CWE-397','ASCRM-CWE-456','ASCRM-CWE-674','ASCRM-CWE-
704','ASCRM-CWE-772','ASCRM-CWE-788','ASCRM-RLB-1','ASCRM-RLB-10','ASCRM-RLB-
11','ASCRM-RLB-12','ASCRM-RLB-13','ASCRM-RLB-14','ASCRM-RLB-15','ASCRM-RLB-
16','ASCRM-RLB-17','ASCRM-RLB-18','ASCRM-RLB-19','ASCRM-RLB-2','ASCRM-RLB-
3','ASCRM-RLB-4','ASCRM-RLB-5','ASCRM-RLB-6','ASCRM-RLB-7','ASCRM-RLB-8','ASCRM-
RLB-9'}-&amp;gt;includes(self.id)" />

<measureElement xmi:type="smm:OCLOperation"
xmi:id="isInASCMM"
name="isInASCMM"
context="SPMS:Definitions::PatternDefinition"
```

body="Set{'ASCMM-MNT-1','ASCMM-MNT-10','ASCMM-MNT-11','ASCMM-MNT-12','ASCMM-MNT-13','ASCMM-MNT-14','ASCMM-MNT-15','ASCMM-MNT-16','ASCMM-MNT-17','ASCMM-MNT-18','ASCMM-MNT-19','ASCMM-MNT-2','ASCMM-MNT-20','ASCMM-MNT-3','ASCMM-MNT-4','ASCMM-MNT-5','ASCMM-MNT-6','ASCMM-MNT-7','ASCMM-MNT-8'}->includes(self.id)" />

Figure 7.12 enriches Figure 7.5 and illustrates the SMM modeling with *ASCRM-CWE-120-roles-moveBufferStatement* pattern.

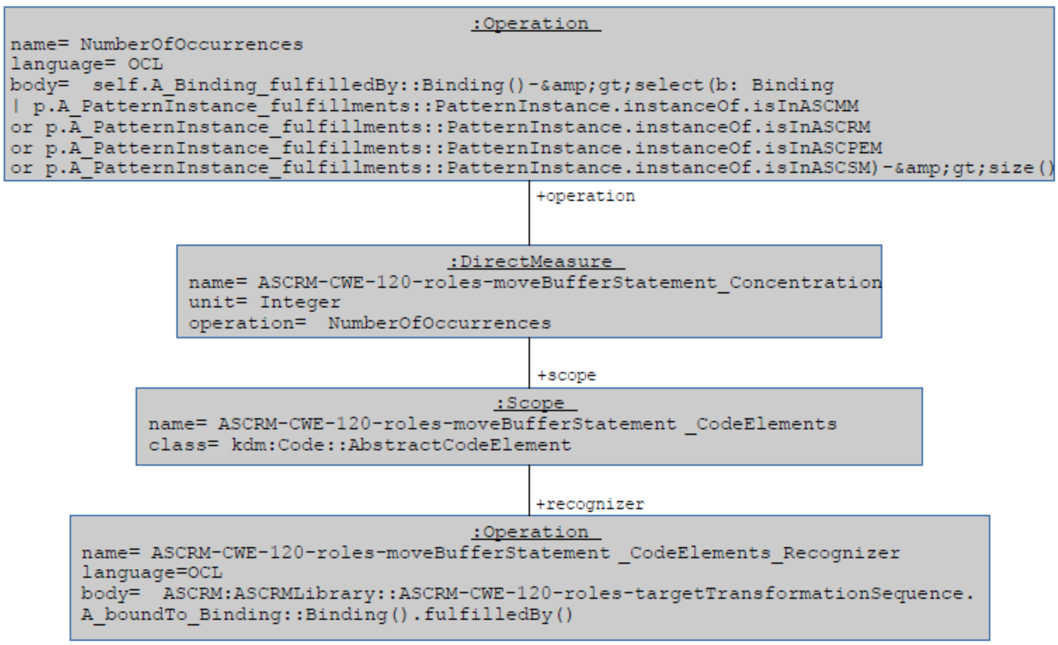

Figure 7.12 - ASCRM-CWE-120-roles-moveBufferStatement role concentration with SMM Operation, DirectMeasure, Scope, and Recognizer

Measure specifications

For each implementation role from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM* patterns, an **smm:OCLOperation** (named as the pattern key with a '_Concentration' suffix) shall be defined.

For each implementation role, the **smm:Scope** (named as the role name with a '_CodeElements' suffix), and its recognizer **smm:Operation** (named as the role name with a '_CodeElements_Recognizer' suffix) will be reused in the current process.

7.3.3.7 Occurrence Gap Size

This sub clause shall only be applicable when the pattern relies on role that model values and threshold values that are not to be exceeded. The Occurrence Gap Size is the extent of the gap to be closed to remediate the weakness, measured as the difference between the values and the thresholds.

The affected patterns are:

- ASCMM-MNT-11: Callable and Method Control Element Excessive Cyclomatic Complexity Value
- ASCMM-MNT-13: Callable and Method Control Element Excessive Number of Parameters
- ASCMM-MNT-17: Class Element Excessive Inheritance Level
- ASCMM-MNT-18: Class Element Excessive Number of Children
- ASCMM-MNT-2: Class Element Excessive Inheritance of Class Elements with Concrete Implementation
- ASCMM-MNT-4: Callable and Method Control Element Number of Outward Calls
- ASCMM-MNT-6: Commented Code Element Excessive Volume
- ASCMM-MNT-8: Source Element Excessive Size
- ASCPEM-PRF-10: Non-SQL Named Callable and Method Control Element with Excessive Number of Data Resource Access
- ASCPEM-PRF-12: Storable and Member Data Element Excessive Number of Aggregated Storable and Member Data Elements
- ASCPEM-PRF-4: Data Resource Read and Write Access Excessive Complexity
- ASCPEM-PRF-6: Large Data Resource ColumnSet Excessive Number of Index Elements
- ASCPEM-PRF-7: Large Data Resource ColumnSet with Index Element of Excessive Size
- ASCPEM-PRF-9: Non-Stored SQL Callable Control Element with Excessive Number of Data Resource Access

For each of the occurrences of these patterns, the occurrence gap size shall be computed the following way:

- Retrieve the value of the roles modeling the exceeding values.
- Retrieve the value of the roles modeling the threshold values.
- Compute the difference.

The difference formulae are:

- ASCMM-MNT-11-roles-cyclomaticComplexity - ASCMM-MNT-11-roles-cyclomaticComplexityThresholdValue
- ASCMM-MNT-13-roles-parameterNumber - ASCMM-MNT-13-roles-parameterNumberThreshold
- ASCMM-MNT-14-roles-numberOfDataOperations - ASCMM-MNT-14-roles-numberOfDataOperationsThresholdValue
- ASCMM-MNT-17-roles-numberOfInheritanceLevels - ASCMM-MNT-17-roles-NumberOfInheritanceLevelsThresholdValue
- ASCMM-MNT-18-roles-numberOfChildren - ASCMM-MNT-18-roles-numberofChildrenThresholdValue
- ASCMM-MNT-2-roles-numberOfConcreteClassInheritances - ASCMM-MNT-2-roles-numberOfConcreteClassInheritancesThresholdValue
- ASCMM-MNT-4-roles-numberOfOutwardReferences - ASCMM-MNT-4-roles-numberOfOutwardReferencesThresholdValue

- ASCMM-MNT-6-roles-percentageOfCommentedOutInstructions - ASCMM-MNT-6-roles-percentageOfCommentedOutInstructionsThresholdValue
- ASCMM-MNT-8-roles-numberOfLinesOfCode - ASCMM-MNT-8-roles-numberOfLinesOfCodeThresholdValue
- ASCPEM-PRF-10-roles-numberOfDataQueries - ASCPEM-PRF-10-roles-numberOfDataQueriesThresholdValue
- ASCPEM-PRF-12-roles-numberOfAggregatedDataElements - ASCPEM-PRF-12-roles-numberOfAggregatedObjectsThresholdValue
- (ASCPEM-PRF-4-roles-numberOfJoins - ASCPEM-PRF-4-roles-numberOfJoinsThresholdValue) + (ASCPEM-PRF-4-roles-numberOfSubQueries - ASCPEM-PRF-4-roles-numberOfSubQueriesThresholdValue)
- ASCPEM-PRF-6-roles-numberOfTableIndices - ASCPEM-PRF-6-roles-numberOfTableIndicesThresholdValue
- ASCPEM-PRF-7-roles-indexRange - ASCPEM-PRF-7-roles-indexRangeThresholdValue
- ASCPEM-PRF-9-roles-numberOfDataQueries - ASCPEM-PRF-9-roles-numberOfDataQueriesThresholdValue

They require to get values from the following roles:

- ASCMM-MNT-11-roles-cyclomaticComplexity
- ASCMM-MNT-11-roles-cyclomaticComplexityThresholdValue
- ASCMM-MNT-13-roles-parameterNumber
- ASCMM-MNT-13-roles-parameterNumberThresholdValue
- ASCMM-MNT-14-roles-numberOfDataOperations
- ASCMM-MNT-14-roles-numberOfDataOperationsThresholdValue
- ASCMM-MNT-18-roles-numberOfChildren
- ASCMM-MNT-18-roles-numberOfChildrenThresholdValue
- ASCMM-MNT-4-roles-numberOfOutwardReferences
- ASCMM-MNT-4-roles-numberOfOutwardReferencesThresholdValue
- ASCMM-MNT-6-roles-percentageOfCommendedOutInstructions
- ASCMM-MNT-6-roles-percentageOfCommentedOutInstructionsThresholdValue
- ASCMM-MNT-8-roles-numberOfLinesOfCode
- ASCMM-MNT-8-roles-numberOfLinesOfCodeThresholdValue
- ASCPEM-PRF-10-roles-numberOfDataQueries
- ASCPEM-PRF-10-roles-numberOfDataQueriesThresholdValue
- ASCPEM-PRF-12-roles-numberOfAggregatedDataElements
- ASCPEM-PRF-12-roles-numberOfAggregatedObjectsThresholdValue
- ASCPEM-PRF-4-roles-numberOfJoins
- ASCPEM-PRF-4-roles-numberOfJoinsThresholdValue
- ASCPEM-PRF-4-roles-numberOfSubQueries
- ASCPEM-PRF-4-roles-numberOfSubQueriesThresholdValue
- ASCPEM-PRF-6-roles-numberOfTableIndices
- ASCPEM-PRF-6-roles-numberOfTableIndicesThresholdValue

- ASCPEM-PRF-7-roles-indexRange
- ASCPEM-PRF-7-roles-indexRangeThresholdValue
- ASCPEM-PRF-9-roles-numberOfDataQueries
- ASCPEM-PRF-9-roles-numberOfDataQueriesThresholdValue

To do so, an smm:Operation and an smm:DirectMeasure shall be defined (e.g., with *ASCMM-MNT-11-roles-cyclomaticComplexity*):

- **<measureElement**

  xmi:type="smm:DirectMeasure"
  xmi:id="ASCMM-MNT-11-roles-cyclomaticComplexity"
  name="ASCMM-MNT-11-roles-cyclomaticComplexity"
  operation="ASCMM-MNT-11-roles-cyclomaticComplexity_Value"
  unit="Integer"
  trait="OccurrenceGapSizing"
  scope="ASCMM-MNT-11_Occurrence"
  shortDescription="Value of ASCMM-MNT-11-roles-cyclomaticComplexity role" />

- relying on:

  <measureElement
  xmi:id="ASCMM-MNT-11-roles-cyclomaticComplexity_Value"
  name="ASCMM-MNT-11-roles-cyclomaticComplexity_Value"
  xmi:type="smm:Operation"
  language="OCL"
  body="ASCMM:ASCMMLibrary::ASCMM-MNT-11-roles-cyclomaticComplexity_Value.A_boundTo_Binding::Binding().fulfilledBy()"
  trait="OccurrenceGapSizing"/>

The occurrence gap size is then an smm:BinaryMeasure computing the difference according to the formulae above. For example, with ASCMM-MNT-11:

- **<measureElement**

  xmi:type="smm:BinaryMeasure"
  xmi:id="ASCMM-MNT-11_OccurrenceGapSize"
  name="ASCMM-MNT-11_OccurrenceGapSize"
  functor="minus"
  unit="integer"
  scope="ASCMM-MNT-11_Occurrence"
  trait="OccurrenceGapSizing"
  shortDescription="Occurrence gap size of ASCMM-MNT-11 pattern" />

Measure specifications

For each applicable patterns from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM* patterns (listed above), an **smm:BinaryMeasure** (named as the pattern key with a '_OccurrenceGapSize' suffix) shall be defined.

For each applicable implementation role (listed above), the **smm:DirectMeasure** (named as the role name without any suffix), and its **smm:Operation** (named as the role name with a '_Value' suffix) shall be defined.

In the case of *ASCPEM-PRF-4*, as the pattern relies on two gaps, two intermediate **smm:BinaryMeasure** (named *ASCPEM-PRF-4_OccurrenceGapSize_Part1* and *ASCPEM-PRF-4_OccurrenceGapSize_Part2*) shall be defined to handle each gap.

7.3.3.8 Evolution Status

This sub clause shall only be applicable when two revisions of the software are available for measurement.

Involved code elements

The evolution status of involved code elements shall be computed the following way:

- For each implementation role, use the defined scope to identify code elements.
- For each code element, its status shall be identified as added, updated, deleted, or unchanged based on the following guidelines:
    - 'added' in latest Revision, when there is no code element which evolved into it.
    - 'deleted' from previous Revision, when there is no code element into which it evolved.
    - 'updated' in latest Revision, where the evidence in the source code that its implementation evolved.
    - 'unchanged' if the code element remains identical in the two revisions.

To identify the evolution status of any code element, a set of **smm:OCLOperation** for each code element shall be determined.

- Added

```
<measureElement xmi:type="smm:OCLOperation"
xmi:id="isAddedElement"
name="isAddedElement"
context="kdm:Core::Element"
body="(isInLatestRevision and not fromRevisionMeasurementScope()-&amp;gt;exists(e:
kdm:Core::Element | e.evolvedTo = self))"
trait="EvolutionStatus"
shortDescription="Evolutions status measured code element: TRUE if added between
revisions"/>
```

- deleted

```
<measureElement xmi:type="smm:OCLOperation"
xmi:id="isDeletedElement"
name="isDeletedElement"
context="kdm:Core::Element"
body="(isInPreviousRevision and not toRevisionMeasurementScope()-&amp;gt;exists(e:
kdm:Core::Element | e.evolvedFrom = self))"
trait="EvolutionStatus"
shortDescription="Evolutions status measured code element: TRUE if deleted between
revisions"/>
```

- updated

```
<measureElement xmi:type="smm:OCLOperation"
xmi:id="isUpdatedElement"
name="isUpdatedElement"
context="kdm:Core::Element"
body="(isInLatestRevision and toRevisionMeasurementScope()-&amp;gt;exists(e:
kdm:Core::Element | e.evolvedTo = self and self.source &amp;lt;&amp;gt; e.source))"
trait="EvolutionStatus"
shortDescription="Evolutions status measured code element: TRUE if updated between
revisions"/>
```

- unchanged

```
<measureElement xmi:type="smm:OCLOperation"
xmi:id="isUnchangedElement"
name="isUnchangedElement"
context="kdm:Core::Element"
body="(isInLatestRevision and not (isUpdatedElement or isAddedElement))"
trait="EvolutionStatus"
shortDescription="Evolutions status measured code element: TRUE if unchanged between
revisions"/>
```

Occurrence

The computation of the evolution status of each occurrence shall include the following additional steps.

1. The analyzer shall check to determine if the roles are implemented by code elements evolved from code elements implementing the same roles in the previous release.
   - either with unchanged code elements, identified via a first **smm:OCLOperation**

```
<measureElement xmi:type="smm:OCLOperation"
xmi:id="hasAllItsCodeElementsEvolvedFromCodeElementsInBindingOfSameRole"
name="hasAllItsCodeElementsEvolvedFromCodeElementsInBindingOfSameRole"
context="SPMS:Observations::Binding"
body="self.fullfiled()-forAll(e: kdm:Core::Element |
e.evolvedFrom.A_Binding_fulfilledBy::Binding()-&amp;gt;exist(b: Binding | b.boundTo =
self.boundTo ) )"
trait="EvolutionStatus"
shortDescription="Evolutions status role implemetation: TRUE if all code elements
implementing a binding of the same role in previous release"/>
```

   - either with unchanged or updated code elements, identified via a second **smm:OCLOperation**

```
<measureElement xmi:type="smm:OCLOperation"
 xmi:id="hasAllItsCodeElementsEvolvedFromCodeElementsInBindingOfSameRole"
name="hasAllItsCodeElementsEvolvedFromCodeElementsInBindingOfSameRole"
context="SPMS:Observations::Binding"
```

```
body="self.fullfiled()-forAll(e: kdm:Core::Element |
e.evolvedFrom.A_Binding_fulfilledBy::Binding()-&amp;gt;exist(b: Binding | b.boundTo =
self.boundTo ) )"
trait="EvolutionStatus"
shortDescription="Evolutions status role implemetation: TRUE if all code elements
implementing a binding of the same role in previous release"/>
```

2. An occurrence shall be considered as:
   - unchanged, if all its roles are implemented by unchanged code elements evolved from code elements implementing the same roles in the previous release, identified via a first **smm:OCLOperation**

```
<measureElement xmi:type="smm:OCL
<measureElement xmi:type="smm:OCLOperation" xmi:id="isUnchangedOccurrence"
name="isUnchangedOccurrence" context="SPMS:Observations::PatternInstance"
body="self.fulfillments()-&amp;gt;forAll(b: SPMS:Observations::Binding |
b.hasAllItsCodeElementsUnchangedFromCodeElementsInBindingOfSameRole )"
trait="EvolutionStatus"
shortDescription="Evolutions status occurrence: TRUE if unchanged between revisions"/>
```

   - updated, if not unchanged and all its roles are implemented by code elements evolved from code elements implementing the same roles in the previous release, identified via a second **smm:OCLOperation**

```
<measureElement xmi:type="smm:OCLOperation" xmi:id="isUpdatedOccurrence"
name="isUpdatedOccurrence"
context="SPMS:Observations::PatternInstance"
body="self.fulfillments()-&amp;gt;forAll(b: SPMS:Observations::Binding |
b.hasAllItsCodeElementsUnchangedFromCodeElementsInBindingOfSameRole ) and not
self.isUnchangedOccurrence"
trait="EvolutionStatus"
shortDescription="Evolutions status occurrence: TRUE if updated between revisions"/>
```

   - added, if in "ToRevision" revision but not updated nor unchanged, identified via a third

```
smm:OCLOperation
<measureElement xmi:type="smm:OCLOperation"
xmi:id="isAddedOccurrence"
name="isAddedOccurrence"
context="SPMS:Observations::PatternInstance"
body="self.isInLatest and not self.isUnchangedOccurrence and not
self.isUpdatedOccurrence"
trait="EvolutionStatus"
shortDescription="Evolutions status occurrence: TRUE if added between revisions"/>
```

7.3.4 Adjustment Factor

For each occurrence, the adjustment factor shall be calculated as the simple product of the following contributions:

- Technological diversity
- Complexity overhead average, across all implementation roles
- Exposure overhead average, across all exposed implementation roles

- Sharing opportunity average, across all implementation roles
- Occurrence Gap Size, when applicable

Note that the evolution status information is not used for adjustment.

7.3.4.1 Technological diversity contribution

The contribution from the occurrence technological diversity specified in sub-clause 34 is direct, that is, the number of languages in which the occurrence is implemented is used as the Technological Diversity input to the adjustment factor calculation.

7.3.4.2 Complexity overhead average contribution

The contribution from the complexity overhead specified in 7.3.3.4 for each implementation role is a simple average (e.g., with *ASCRM-CWE-120*):

```
<measureElement xmi:type="smm:CollectiveMeasure"
xmi:id="ASCRM-CWE-120_OccurrenceComplexityOverheadAverage"
name="ASCRM-CWE-120_OccurrenceComplexityOverheadAverage"
unit="Real"
accumulator="average"
scope="ASCRM-CWE-120_Occurrence"
trait="ComplexityEstimating"
category="FunctionalMetrics"
shortDescription="Complexity overhead average of an occurrence of ASCRM-CWE-120 pattern,
measured as the AEP complexity overhead when compared to simplest complexity" />
```

Figure 7.13 illustrates the SMM modeling with *ASCRM-CWE-120* pattern.

Figure 7.13 - ASCRM-CWE-120 ocurrence complexity overhead average with SMM CollectiveMeasure and RatioMeasures

Measure specifications

An **smm:CollectiveMeasure** measure (named as the pattern key with a '_OccurrenceComplexityOverheadAverage' suffix) shall be defined for each source code pattern from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*, as illustrated with the *ASCRM-CWE-120* pattern above.

7.3.4.3 Exposure overhead average contribution

The contribution from the exposure specified in 7.3.3.5 for each implementation role is a simple average. It is considered an overhead vis-à-vis the 'best case scenario' where the exposure value is "1."

For example, with *ASCRM-CWE-120*.

**<measureElement xmi:type="smm:CollectiveMeasure"
xmi:id="ASCRM-CWE-120_OccurrenceExposureOverheadAverage"
name="ASCRM-CWE-120_OccurrenceExposureOverheadAverage"
unit="Real"
accumulator="average"
scope="ASCRM-CWE-120_Occurrence"
trait="ExposureEstimating"
category="FunctionalMetrics"
shortDescription="Exposure overhead average of an occurrence of pattern, measured as the exposure overhead when compared to simplest exposure of 1" />**

Figure 7.14 illustrates the SMM modeling with *ASCRM-CWE-120* pattern.

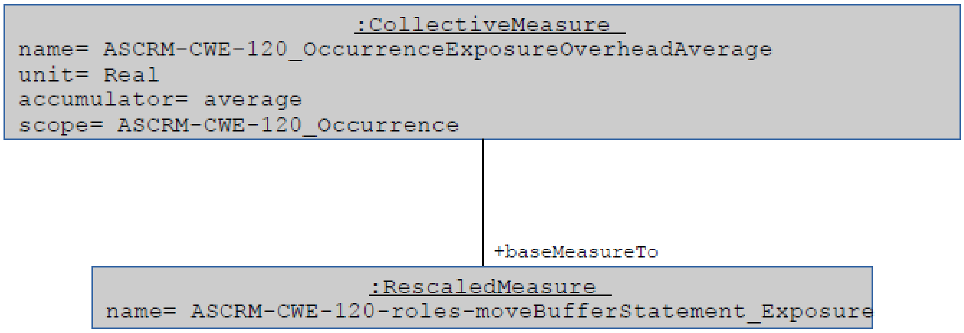

Figure 7.14 - ASCRM-CWE-120 occurrence complexity overhead average with SMM CollectiveMeasure and RatioMeasures

Measure specifications

An **smm:CollectiveMeasure** measure (named as the pattern key with a '_OccurrenceExposureOverheadAverage' suffix) shall be defined for each source code pattern from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*, as illustrated with the *ASCRM-CWE-120* pattern above.

7.3.4.4 Sharing opportunity average contribution

The contribution from the sharing opportunity specified in 7.3.3.6 for each implementation role is a simple average. It is considered an opportunity to share the effort vis-à-vis the nominal situation where the concentration value is 1 (e.g., with *ASCRM-CWE-120*).

<measureElement xmi:type="smm:CollectiveMeasure"
xmi:id="ASCRM-CWE-120_OccurrenceSharingOpportunityAverage"
name="ASCRM-CWE-120_OccurrenceSharingOpportunityAverage"
unit="Real"
accumulator="average"
scope="ASCRM-CWE-120_Occurrence"
trait="SharingLevelEstimating"
category="FunctionalMetrics"
shortDescription="Sharing opportunity average of an occurrence of ASCRM-CWE-120 pattern, measured as the number of distinct occurrences sharing code elements" />

Figure 7.15 illustrates the SMM modeling with *ASCRM-CWE-120* pattern.

Figure 7.15 - ASCRM-CWE-120 occurrence sharing opportunity average with SMM CollectiveMeasure and RescaledMeasures

Measure specifications

An **smm:CollectiveMeasure** measure (named as the pattern key with a '_OccurrenceSharingOpportunityAverage' suffix) shall be defined for each source code pattern from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*, as illustrated with the *ASCRM-CWE-120* pattern above.

7.3.4.5 Occurrence gap size contribution

The contribution from the occurrence gap size specified in 7.3.3.7 is direct, that is, the difference between exceeding value and threshold value not to exceed is used as input to the adjustment factor calculation.

7.3.4.6 Adjustment factor computation

For each occurrence, the adjustment factor shall be computed as the product of all four contributions.

For example, with *ASCRM-CWE-120*.

<measureElement xmi:type="smm:CollectiveMeasure"
xmi:id="ASCRM-CWE-120_OccurrenceAdjustmentFactor"
name="ASCRM-CWE-120_OccurrenceAdjustmentFactor"
accumulator="product"
unit="Real"
scope="ASCRM-CWE-120_Occurrence"
trait="RemediationEffortEstimating"
category="FunctionalMetrics"

shortDescription="Contextual Factor to adjust Raw Remediation Effort to remove one occurrence of ASCRM-CWE-120 in latest Revision" />

Figure 7.16 illustrates the SMM modeling with *ASCRM-CWE-120* pattern.

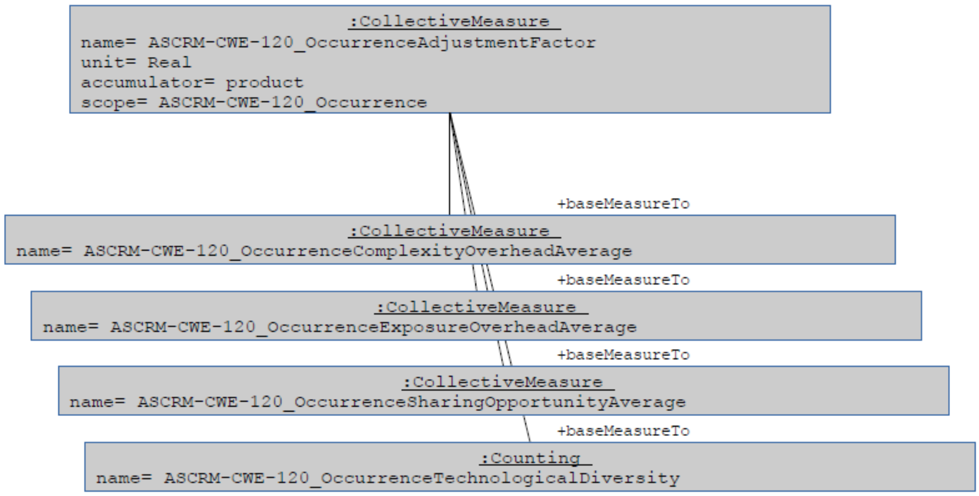

Figure 7.16 - ASCRM-CWE-120 occurrence adjustment factor with SMM CollectiveMeasures and Counting

Figure 7.17 illustrates the SMM modeling with ASCMM-MNT-11, for which a fifth contribution from the Occurrence Gap Size is also part of the computation.

Figure 7.17 - ASCMM-MNT-11 occurrence adjustment factor with SMM CollectiveMeasures, BinaryMeasure, and Counting

Measure specifications

An **smm:CollectiveMeasure** measure (named as the pattern key with a '_OccurrenceAdjustmentFactor' suffix) shall be defined for each source code pattern from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*, as illustrated with the *ASCRM-CWE-120* pattern above.

7.3.5 Adjusted Remediation Effort

For each occurrence, the adjusted remediation effort is simply the product of the unadjusted remediation effort value from 7.3.2 by the adjustment factor value from 7.3.4. For example, with *ASCRM-CWE-120*:

```
<measureElement xmi:type="smm:BinaryMeasure"
xmi:id="ASCRM-CWE-120_OccurrenceRemediationEffort"
name="ASCRM-CWE-120_OccurrenceRemediationEffort"
functor="multiply"
unit="effort(minutes)"
scope="ASCRM-CWE-120_Occurrence"
trait="RemediationEffortEstimating"
category="FunctionalMetrics"
shortDescription="Remediation Effort to remove one occurrence of ASCRM-CWE-120 in latest
Revision" />
```

Figure 7.18 illustrates the SMM modeling with the *ASCRM-CWE-120* pattern.

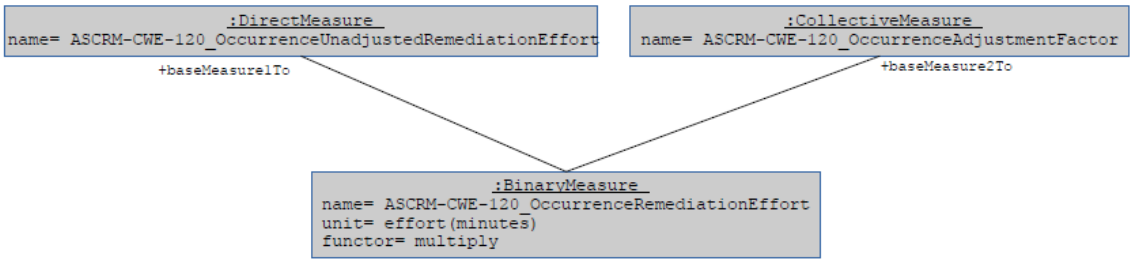

Figure 7.18 - ASCRM-CWE-120 occurrence "adjusted" remediation effort with SMM BinaryMeasure, CollectiveMeasure, and DirectMeasure

Measure specifications

An **smm:BinaryMeasure** measure (named as the pattern key with a '_OccurrenceRemediationEffort' suffix) shall be defined for each source code pattern from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*, as illustrated with the *ASCRM-CWE-120* pattern above.

7.4 Quantification of Remediation Effort at the Pattern Level

The Pattern Remediation Effort values are simply the sum for each pattern of the Occurrence Remediation Effort values described in 7.3.5.

This summation shall be done with an **smm:CollectiveMeasure**. For example, with the *ASCRM-CWE-120* pattern:

```
<measureElement xmi:type="smm:CollectiveMeasure"
xmi:id="ASCRM-CWE-120_PatternRemediationEffort"
name="ASCRM-CWE-120_PatternRemediationEffort"
accumulator="sum"
unit="effort(minutes)"
scope="toRevisionMeasurementScope"
trait="RemediationEffortEstimating"
category="FunctionalMetrics"
shortDescription="Remediation Effort to remove all occurrences of ASCRM-CWE-120 in latest
Revision" />
```

Figure 7.19 illustrates the SMM modeling with *ASCRM-CWE-120* pattern.

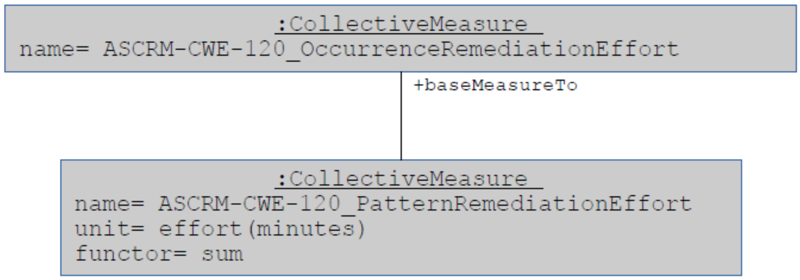

Figure 7.19 - ASCRM-CWE-120 pattern remediation effort with SMM CollectiveMeasures

Measure specifications

An **smm:CollectiveMeasure** measure (named as the pattern key with a '_PatternRemediationEffort' suffix) shall be defined for each source code pattern from *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM*, as illustrated with the *ASCRM-CWE-120* pattern above.

7.5 Quantification of Remediation Effort for CISQ Quality Characteristics

Remediation efforts shall be calculated for each of the CISQ Quality Characteristics:

- Automated Reliability Remediation Effort Measure (ARREM).
- Automated Security Remediation Effort Measure (ASREM).
- Automated Performance Efficiency Remediation Effort Measure (APEREM).
- Automated Maintainability Remediation Effort Measure (AMREM).

The *AMREM*, *ARREM*, *APEREM*, and *ASREM* values shall be computed by summing the remediation efforts for applicable source code patterns included in the *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM* specifications respectively.

7.5.1.1.1 Pattern applicability considerations

Although designed as technology-agnostic specifications, *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM* contain source code patterns that are not applicable to all programming languages. When a pattern is not applicable, there are no occurrences to process.

Measures' specifications

- *AMREM* is an **smm:CollectiveMeasure that shall** sum the pattern-level remediation effort measure values from 7.4 (note that the **smm:MeasureRelationship** elements towards pattern level measures are not shown here):
    - **<measureElements xmi:id="ATDM-ATDMLibrary-AutomatedMaintainabilityRemediationEffortMeasureInLatest" xmi:type="smm:CollectiveMeasure" name="AutomatedMaintainabilityRemediationEffortMeasure" accumulator="sum" scope="LatestRevision"**

**trait="RemediationEffortEstimating"
unit="effort(minutes)"
baseMeasureTo="
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-1_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-10_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-11_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-12_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-13_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-14_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-15_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-16_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-17_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-18_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-19_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-2_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-20_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-3_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-4_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-5_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-6_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-7_PatternRemediationEffort
AutomatedMaintainabilityRemediationEffortMeasure_to_ASCMM-MNT-8_PatternRemediationEffort" />**

- *ARREM* is an **smm:CollectiveMeasure that shall** sum the pattern-level remediation effort measure values from 7.4 (note that the **smm:MeasureRelationship** elements towards pattern level measures are not shown here):
  - **<measureElements xmi:id="ATDM-ATDMLibrary-AutomatedReliabilityRemediationEffortMeasureInLatest"
    xmi:type="smm:CollectiveMeasure"
    name="AutomatedReliabilityRemediationEffortMeasure"
    accumulator="sum" scope="LatestRevision"
    trait="RemediationEffortEstimating"
    unit="effort(minutes)"
    baseMeasureTo="**

AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-CWE-120_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-CWE-252-data_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-CWE-252-resource_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-CWE-396_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-CWE-397_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-CWE-456_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-CWE-674_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-CWE-704_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-CWE-772_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-CWE-788_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-1_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-10_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-11_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-12_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-13_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-14_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-15_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-16_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-17_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-18_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-19_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-2_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-3_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-4_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-5_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-6_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-7_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-

8_PatternRemediationEffort
AutomatedReliabilityRemediationEffortMeasure_to_ASCRM-RLB-
9_PatternRemediationEffort" />

- ASREM is an **smm:CollectiveMeasure that shall** sum the pattern-level remediation effort measure values from 7.4 (note that the **smm:MeasureRelationship** elements towards pattern level measures are not shown here):

    - <measureElements xmi:id="ATDM-ATDMLibrary-AutomatedSecurityRemediationEffortMeasureInLatest"
      xmi:type="smm:CollectiveMeasure"
      name="AutomatedSecurityRemediationEffortMeasure"
      accumulator="sum"
      scope="LatestRevision"
      trait="RemediationEffortEstimating"
      unit="effort(minutes)"
      baseMeasureTo="
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-120_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-129_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-134_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-22_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-252-resource_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-327_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-396_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-397_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-434_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-456_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-606_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-667_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-672_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-681_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-99_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-772_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-78_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-789_PatternRemediationEffort
      AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-79_PatternRemediationEffort

AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-798_PatternRemediationEffort AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-835_PatternRemediationEffort AutomatedSecurityRemediationEffortMeasure_to_ASCSM-CWE-89_PatternRemediationEffort" />

- *APEREM* is an **smm:CollectiveMeasure that shall** sum the pattern-level remediation effort measure values from 7.4 (note that the **smm:MeasureRelationship** elements towards pattern level measures are not shown here):
  - **<measureElements xmi:id="ATDM-ATDMLibrary-AutomatedPerformanceRemediationEffortMeasureInLatest" xmi:type="smm:CollectiveMeasure" name="AutomatedPerformanceEfficiencyRemediationEffortMeasure" accumulator="sum" scope="LatestRevision" trait="RemediationEffortEstimating" unit="effort(minutes)" baseMeasureTo=" AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-1_PatternRemediationEffort AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-10_PatternRemediationEffort AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-11_PatternRemediationEffort AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-12_PatternRemediationEffort AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-13_PatternRemediationEffort AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-14_PatternRemediationEffort AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-15_PatternRemediationEffort AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-2_PatternRemediationEffort AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-3_PatternRemediationEffort AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-4_PatternRemediationEffort AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-5_PatternRemediationEffort AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-6_PatternRemediationEffort AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-7_PatternRemediationEffort AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-8_PatternRemediationEffort AutomatedPerformanceEfficiencyRemediationEffortMeasure_to_ASCPEM-PRF-9_PatternRemediationEffort" />**

The AMREM, ARREM, APEREM, and ASREM flow are displayed in Figures 7.20, 7.21, 7.22, and 7.23 respectively.

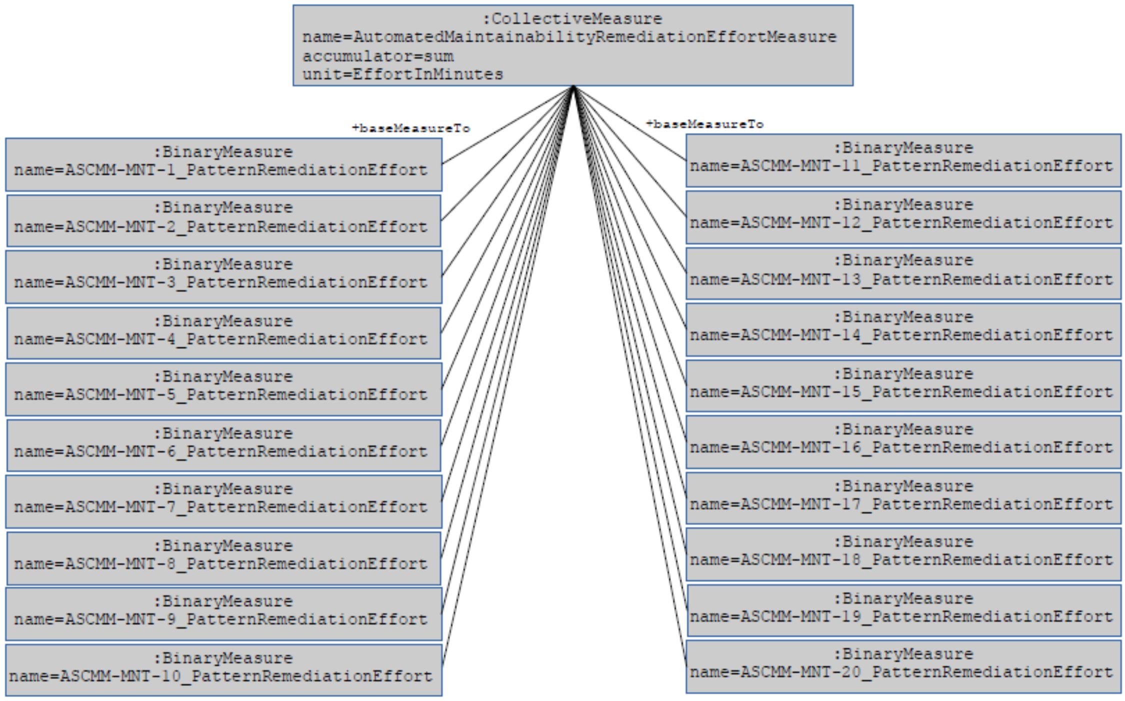

Figure 7.20 - AMREM Flow

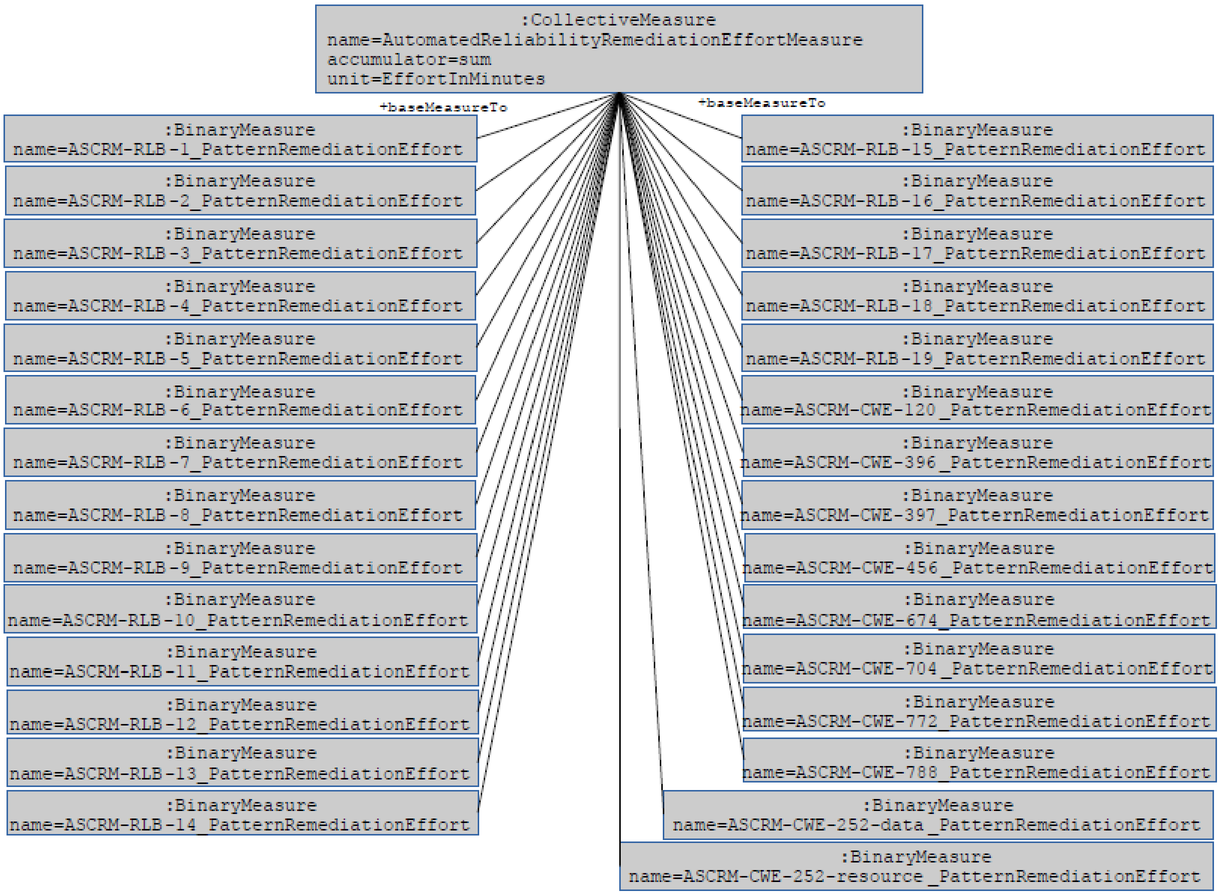

Figure 7.21 - ARREM Flow

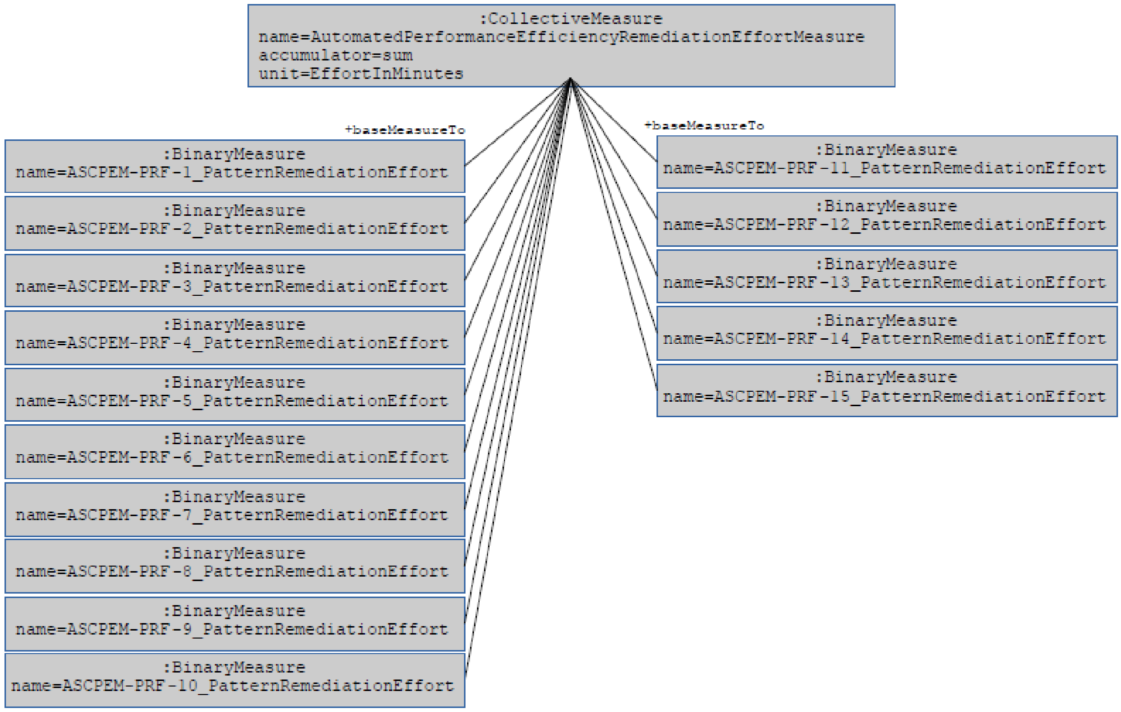

Figure 7.22 - APEREM Flow

Figure 7.23 - ASREM Flow

7.6 Quantification of Remediation Effort at the Software Level (ATDM)

The Automated Technical Debt Measure (ATDM) shall be calculated by summing the remediation efforts of all patterns in the CISQ Quality Characteristic specifications (ASCMM, ASCRM, ASCPEM, ASCSM) specifications, counting only once the remediation effort of patterns that are shared between multiple specifications.

Shared Pattern considerations

Shared patterns shall be identified based on the Comment **:PatternSection** of patterns defined in the *ASCMM*, *ASCRM*, *ASCPEM*, and *ASCSM* specifications. When computing the overall *ATDM* value, occurrences of shared patterns shall be counted only once. Shared patterns include:

- ASCSM-CWE-120 and ASCRM-CWE-120
- ASCSM-CWE-456 and ASCRM-CWE-456
- ASCSM-CWE-772 and ASCRM-CWE-772
- ASCSM-CWE-252 and ASCRM-CWE-252-resource
- ASCSM-CWE-396 and ASCRM-CWE-396

- ASCRM-RLB-10 and ASCPEM-PRF-1
- ASCRM-RLB-13 and ASCMM-MNT-7

In the measure specifications below, only the following patterns are used:

- ASCRM-CWE-120
- ASCRM-CWE-456
- ASCRM-CWE-772
- ASCRM-CWE-252-resource
- ASCRM-CWE-396
- ASCRM-CWE-397
- ASCPEM-PRF-1
- ASCMM-MNT-7

Measure specifications

- *ATDM* is an **smm:CollectiveMeasure that shall** sum the pattern-level remediation effort measure values from 7.4 (note that the **smm:MeasureRelationship** elements towards pattern level measures are not shown here):
    - **<measureElements xmi:id="ATDM-ATDMLibrary-AutomatedTechnicalDebtPrincipalMeasureInLatest" xmi:type="smm:CollectiveMeasure" name="AutomatedTechnicalDebtPrincipalMeasure" accumulator="sum" scope="LatestRevision" trait="RemediationEffortEstimating" unit="effort(minutes)" baseMeasureTo=" AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-1_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-10_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-11_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-12_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-13_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-14_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-15_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-16_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-17_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-18_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-19_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-2_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-20_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-3_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-4_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-5_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-**

6_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-7_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCMM-MNT-8_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-CWE-120_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-CWE-252-data_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-CWE-252-resource_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-CWE-396_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-CWE-397_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-CWE-456_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-CWE-674_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-CWE-704_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-CWE-772_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-CWE-788_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-1_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-11_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-12_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-14_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-15_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-16_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-17_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-18_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-19_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-2_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-3_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-4_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-5_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-6_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-7_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-8_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCRM-RLB-9_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-CWE-129_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-CWE-134_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-CWE-22_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-CWE-327_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-

**CWE-434_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-CWE-
606_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-
CWE-667_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-CWE-
672_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-
CWE-681_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-CWE-
99_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-
CWE-78_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-CWE-
789_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-
CWE-79_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-CWE-
798_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-
CWE-835_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCSM-CWE-
89_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-PRF-
1_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-
PRF-10_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-PRF-
11_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-
PRF-12_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-PRF-
13_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-PRF-
14_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-PRF-
15_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-PRF-
2_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-
PRF-3_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-PRF-
4_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-
PRF-5_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-PRF-
6_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-
PRF-7_PatternRemediationEffort
AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-PRF-
8_PatternRemediationEffort AutomatedTechnicalDebtPrincipalMeasure_to_ASCPEM-
PRF-9_PatternRemediationEffort" />**

The ATDM calculation flow is displayed in Figure 7.24.

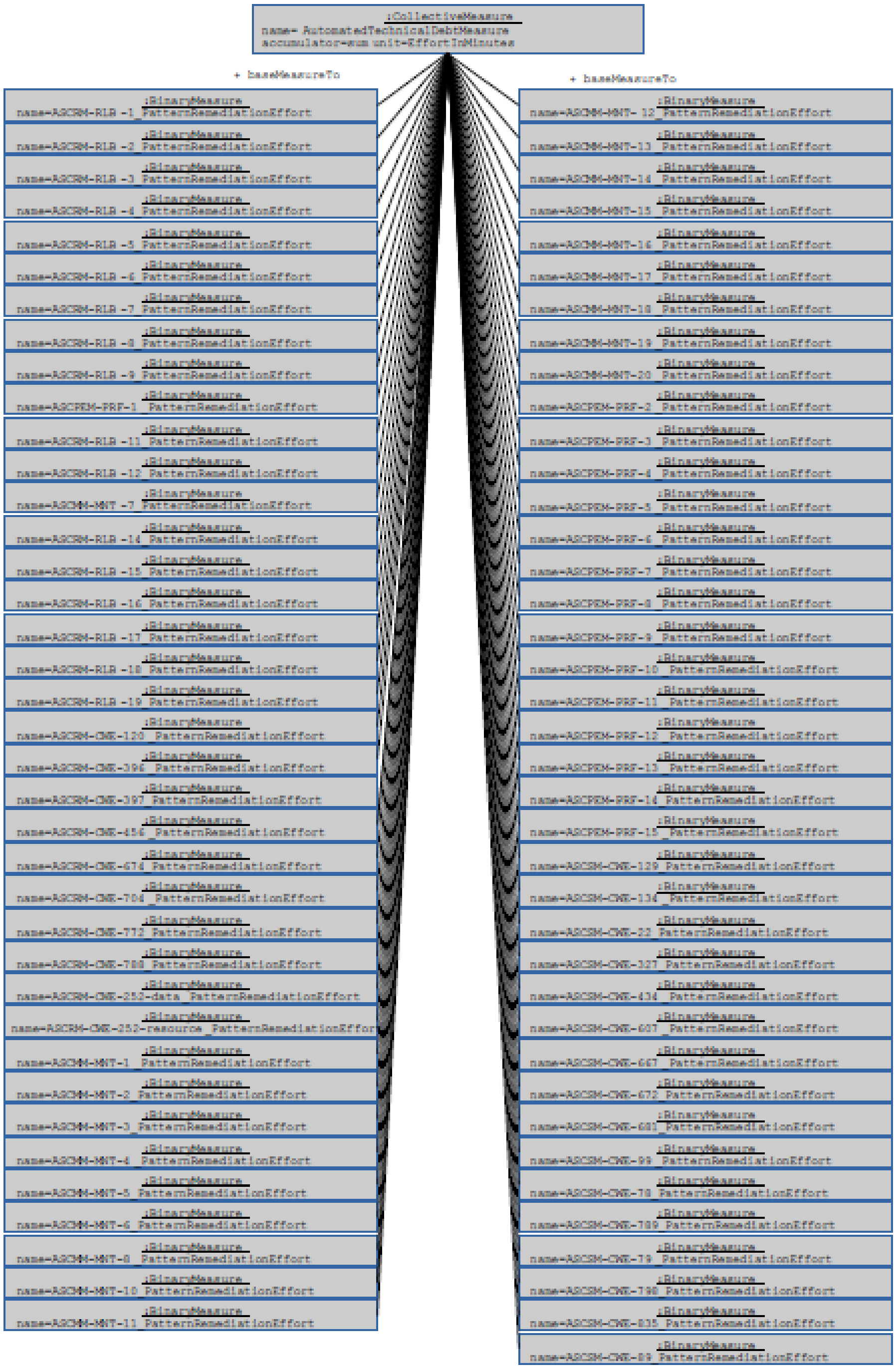

Figure 7.24 - ATDM Flow

7.7 Summary of Remediation Effort Parameters

7.7.1 ASCSM Remediation Configuration

Table 7.1 lists the values to be used to compute unadjusted remediation effort for each occurrence in 7.3.2 for ASCSM source code patterns.

Table 7.1: Configuration of unadjusted remediation effort per ASCSM occurrence

Time to fix (minutes)			**ASCSM pattern name**
Default	**Range**		
	Lo	**Hi**	
30	15	60	ASCSM-CWE-120 Buffer Copy without Checking Size of Input
50	15	180	ASCSM-CWE-129 Array Index Improper Input Neutralization
60	20	180	ASCSM-CWE-134 Format String Improper Input Neutralization
60	20	180	ASCSM-CWE-22 Path Traversal Improper Input Neutralization
50	20	180	ASCSM-CWE-252-resource Unchecked Return Parameter Value of named Callable and Method Control Element with Read, Write, and Manage Access to Platform Resource
120	45	300	ASCSM-CWE-327 Broken or Risky Cryptographic Algorithm Usage
50	20	180	ASCSM-CWE-396 Declaration of Catch for Generic Exception
50	20	180	ASCSM-CWE-397 Declaration of Throws for Generic Exception
90	20	240	ASCSM-CWE-434 File Upload Improper Input Neutralization
50	20	120	ASCSM-CWE-456 Storable and Member Data Element Missing Initialization
60	20	120	ASCSM-CWE-606 Unchecked Input for Loop Condition
120	30	240	ASCSM-CWE-667 Shared Resource Improper Locking
90	30	180	ASCSM-CWE-672 Expired or Released Resource Usage
60	20	180	ASCSM-CWE-681 Numeric Types Incorrect Conversion
50	20	240	ASCSM-CWE-99 Improper Control of Resource Identifiers ('Resource Injection')
120	20	270	ASCSM-CWE-772 Missing Release of Resource after Effective Lifetime
90	30	150	ASCSM-CWE-78 OS Command Injection Improper Input Neutralization
50	20	120	ASCSM-CWE-789 Uncontrolled Memory Allocation
120	20	270	ASCSM-CWE-79 Cross-site Scripting Improper Input Neutralization
90	20	180	ASCSM-CWE-798 Hard-Coded Credentials Usage for Remote Authentication
90	30	150	ASCSM-CWE-835 Loop with Unreachable Exit Condition ('Infinite Loop')
90	20	180	ASCSM-CWE-89 SQL Injection Improper Input Neutralization

7.7.2 ASCRM remediation configuration

Table 7.2 lists the values to be used to compute unadjusted remediation effort for each occurrence in 7.3.2 for ASCRM source code patterns.

Table 7.2: Configuration of unadjusted remediation effort per ASCRM occurrence

Time to fix (minutes)			**ASCRM pattern name**
Default	**Range**		
	Lo	**Hi**	
30	15	60	ASCRM-CWE-120 Buffer Copy without Checking Size of Input
50	20	120	ASCRM-CWE-252-data Unchecked Return Parameter Value of named Callable and Method Control Element with Read, Write, and Manage Access to Data Resource
50	20	120	ASCRM-CWE-252-resource Unchecked Return Parameter Value of named Callable and Method Control Element with Read, Write, and Manage Access to Platform Resource
50	20	120	ASCRM-CWE-396 Declaration of Catch for Generic Exception
50	20	90	ASCRM-CWE-397 Declaration of Throws for Generic Exception
60	20	150	ASCRM-CWE-674 Uncontrolled Recursion
30	20	60	ASCRM-CWE-456 Storable and Member Data Element Missing Initialization
60	20	150	ASCRM-CWE-704 Incorrect Type Conversion or Cast
120	45	300	ASCRM-CWE-772 Missing Release of Resource after Effective Lifetime
50	20	90	ASCRM-CWE-788 Memory Location Access After End of Buffer
50	20	120	ASCRM-RLB-1 Empty Exception Block
40	20	90	ASCRM-RLB-2 Serializable Storable Data Element without Serialization Control Element
90	20	300	ASCRM-RLB-3 Serializable Storable Data Element with non-Serializable Item Elements
90	20	180	ASCRM-RLB-4 Persistant Storable Data Element without Proper Comparison Control Element
240	90	420	ASCRM-RLB-5 Runtime Resource Management Control Element in a Component Built to Run on Application Servers
40	20	120	ASCRM-RLB-6 Storable or Member Data Element containing Pointer Item Element without Proper Copy Control Element
60	30	230	ASCRM-RLB-7 Class Instance Self Destruction Control Element
120	45	300	ASCRM-RLB-8 Named Callable and Method Control Elements with Variadic Parameter Element
40	20	120	ASCRM-RLB-9 Float Type Storable and Member Data Element Comparison with Equality Operator
90	30	300	ASCRM-RLB-10 Data Access Control Element from Outside Designated Data Manager Component
120	30	240	ASCRM-RLB-11 Named Callable and Method Control Element in Multi-Thread Context with non-Final Static Storable or Member Element

Time to fix (minutes)			ASCRM pattern name
60	20	120	ASCRM-RLB-12 Singleton Class Instance Creation without Proper Lock Element Management
240	60	360	ASCRM-RLB-13 Inter-Module Dependency Cycles
120	50	300	ASCRM-RLB-14 Parent Class Element with References to Child Class Element
50	20	180	ASCRM-RLB-15 Class Element with Virtual Method Element without Virtual Destructor
90	40	300	ASCRM-RLB-16 Parent Class Element without Virtual Destructor Method Element
90	30	120	ASCRM-RLB-17 Child Class Element without Virtual Destructor unlike its Parent Class Element
120	45	300	ASCRM-RLB-18 Storable and Member Data Element Initialization with Hard-Coded Network Resource Configuration Data
90	30	240	ASCRM-RLB-19 Synchronous Call Time-Out Absence

7.7.3 ASCPEM remediation configuration

Table 7.3 lists the values to be used to compute unadjusted remediation effort for each occurrence in 7.3.2 for ASCPEM source code patterns.

Table 7.3: Configuration of unadjusted remediation effort per ASCPEM occurrence

Time to fix (minutes)			ASCPEM pattern name
Default	**Range**		
	Lo	**Hi**	
60	20	180	ASCPEM-PRF-1 Static Block Element containing Class Instance Creation Control Element
30	20	90	ASCPEM-PRF-2 Immutable Storable and Member Data Element Creation
120	20	300	ASCPEM-PRF-3 Static Member Data Element outside of a Singleton Class Element
360	120	600	ASCPEM-PRF-4 Data Resource Read and Write Access Excessive Complexity
150	60	300	ASCPEM-PRF-5 Data Resource Read Access Unsupported by Index Element
240	60	480	ASCPEM-PRF-6 Large Data Resource ColumnSet Excessive Number of Index Elements
360	120	600	ASCPEM-PRF-7 Large Data Resource ColumnSet with Index Element of Excessive Size
180	50	300	ASCPEM-PRF-8 Control Elements Requiring Significant Resource Element within Control Flow Loop Block
240	90	540	ASCPEM-PRF-9 Non-Stored SQL Callable Control Element with Excessive Number of Data Resource Access
300	90	540	ASCPEM-PRF-10 Non-SQL Named Callable and Method Control Element with Excessive Number of Data Resource Access

Time to fix (minutes)			ASCPEM pattern name
300	90	480	ASCPEM-PRF-11 Data Access Control Element from Outside Designated Data Manager Component
120	30	300	ASCPEM-PRF-12 Storable and Member Data Element Excessive Number of Aggregated Storable and Member Data Elements
300	180	600	ASCPEM-PRF-13 Data Resource Access not using Connection Pooling capability
180	45	360	ASCPEM-PRF-14 Storable and Member Data Element Memory Allocation Missing De-Allocation Control Element
90	30	210	ASCPEM-PRF-15 Storable and Member Data Element Reference Missing De-Referencing Control Element

7.7.4 ASCMM remediation configuration

Table 7.4 lists the values to be used to compute unadjusted remediation effort for each occurrence in 7.3.2 for ASCMM source code patterns.

Table 7.4: Configuration of unadjusted remediation effort per ASCMM occurrence

Time to fix (minutes)			ASCMM pattern name
Default	**Range**		
	Lo	**Hi**	
90	45	180	ASCMM-MNT-1 Control Flow Transfer Control Element outside Switch Block
180	45	420	ASCMM-MNT-2 Class Element Excessive Inheritance of Class Elements with Concrete Implementation
30	20	90	ASCMM-MNT-3 Storable and Member Data Element Initialization with Hard-Coded Literals
360	60	600	ASCMM-MNT-4 Callable and Method Control Element Number of Outward Calls
60	20	240	ASCMM-MNT-5 Loop Value Update within the Loop
30	20	90	ASCMM-MNT-6 Commented-out Code Element Excessive Volume
300	60	600	ASCMM-MNT-7 Inter-Module Dependency Cycles
180	40	420	ASCMM-MNT-8 Source Element Excessive Size
120	**60**	**300**	ASCMM-MNT-10 Named Callable and Method Control Element Multi-Layer Span
120	**50**	**300**	ASCMM-MNT-11 Callable and Method Control Element Excessive Cyclomatic Complexity Value
120	**50**	**360**	ASCMM-MNT-12 Named Callable and Method Control Element with Layer-skipping Call
180	**50**	**420**	ASCMM-MNT-13 Callable and Method Control Element Excessive Number of Parameters
180	**30**	**300**	ASCMM-MNT-14 Callable and Method Control Element Excessive Number of Control Elements involving Data Element from Data Manager or File Resource

Time to fix (minutes)			ASCMM pattern name
40	**20**	**90**	ASCMM-MNT-15 Public Member Element
40	**20**	**120**	ASCMM-MNT-16 Method Control Element Usage of Member Element from other Class Element
300	**60**	**600**	ASCMM-MNT-17 Class Element Excessive Inheritance Level
300	**60**	**600**	ASCMM-MNT-18 Class Element Excessive Number of Children
40	**20**	**150**	ASCMM-MNT-19 Named Callable and Method Control Element Excessive Similarity
30	**20**	**90**	ASCMM-MNT-20 Unreachable Named Callable or Method Control Element

7.8 Output Generation

The last step of the automated process shall generate the output. The output shall be a human readable report that contains sufficient detail to answer the following questions:

- What is the amount of Automated Tecnical Debt (ATDM)?
- What is the amount of Remediation Effort required for each of the Quality Characteristic measures (Automated Maintainability/Reliability/Performance Efficiency/Security)?
- What is the amount of ATDM added between two revisions?
- What is the amount of ATDM concentrated in any set of code elements?
- What are the exposures of individual occurrences in the ATDM?
- What are the assumptions used in calculating ATDM?

The generated output file format shall be a common text file format (e.g., .txt or .csv) to allow for importing to other tools such as Excel or a commercial software estimating package. The output shall include the following artifacts:

- At the measurement level
    - ASCSM, ASCRM, ASCPEM, ASCMM measurement input
    - Remediation effort configuration input (if not the default values)
    - AEP Effort Complexity measurement input (if not the default values)
- At the software revision level
    - ATDM value
    - AMREM, ARREM, APEREM, and ASREM values
- At the pattern level, for all patterns
    - Pattern remediation effort values
- At the occurrence level, for all occurrences of all patterns
    - Occurrence remediation effort values

- Occurrence adjustment factor values
- Occurrence complexity and exposure overhead average values
- Occurrence sharing opportunity average values
- Occurrence technological diversity values
- Occurrence evolution status

- At the role level, for all occurrences of all patterns
  - List of code elements implementing a role
  - Complexity of role implementation code elements
  - Concentration of role implementation code elements
  - Evolution status of role implementation code elements
  - Direct and indirect exposure of role implementation code elements (applicable roles only)

8 Automated Technical Debt Measure (ATDM) Usage Scenarios (informative)

8.1 Risk Mitigation

The following scenarios illustrate ways in which the Automated Technical Debt Measure (ATDM) and qualitifcation measures can be used to help mitigate the risk of the Technical Debt associated with IT applications.

8.1.1 ATDM and its Component Effort Values for AMREM, ARREM, APEREM, and ASREM

Principle

Compare the ATDM value and individual CISQ Quality Characteristic remediation values (AMREM, ARREM, APEREM, ASREM).

This comparison helps determine when the total ATDM value (normalized by size, if needed) is unequally distributed between Technical Debt Items associated with Security, Performance Efficiency, or Reliability.

8.1.2 Exposure

Principle

Chart the occurrences of Technical Debt Items by exposure values to evaluate Risk Propagation and remediate destabilizing exposures.

This distribution helps identify which Technical Debt Items possess the greatest risk levels in terms of cost to remediate, and possible destabilization resulting from remediation activities.

8.1.3 Evolution Status

Principle

Chart the ATDM value by the evolution status occurrences across releases.

This distribution helps identify trends in the management of Technical Debt. For instance, how much legacy Technical Debt exists in an application, and how much is being added or remediated in each subsequent release. Evolution status can also be used in analyzing trends in the operational risks and cost of ownership associated with the Technical Debt as it is measured across releases.

8.2 Priority Setting

The following scenarios illustrate the ways measures defined in ATDM specifications can be used to help set priorities for remediating Technical Debt Items.

8.2.1 ATDM and its component effort values for AMREM, ARREM, APEREM, ASREM

Principle

Use the CISQ Quality Characteristic remediation values (AMREM, ARREM, APEREM, ASREM) to prioritize and allocate resources among the Quality Characteristics for remediating Technical Debt Items.

8.2.2 Technological Diversity

Principle

Chart occurrences of Technical Debt Items by their Technological Diversity. This distribution identifies Technical Debt Items:

- which will require synchronization between multiple teams involved in a remediation during the development and release cycle
- which can be handled by a single team

8.2.3 Exposure

Principle

Chart occurrences to Technical Debt Items by the range of exposure values. This distribution helps identify Technical Debt Items with:

- the highest Risk Propagation and Fix Destabilization exposure so they can be remediated first during the release development cycle to remove the most impacting issues with enough time before the release to handle potential side-effects of the fix.
- the highest Fix Destabilization exposure but lower Risk Propagation exposure so they can be remediated next during the release development cycle to remove issues while there is enough time to handle potential side-effects of the fix.
- the lowest Fix Destabilization exposure that are to be removed near the end of the release development cycle to remove issues without jeopardizing the stability of the release.

8.2.4 Evolution Status

Principle

Chart occurrences of Technical Debt Items by the evolution of each occurrence.

This distribution helps identify added Technical Debt Items that should be removed first to avoid letting future enhancements build on top of them, making them more difficult to remove in the future and increasing their potential negative impacts.

8.3 Productivity Measurement

The following scenario illustrates the way ATDM measures can be used in productivity analysis.

8.3.1 Evolution Status

Principle

Filter the occurrences of Technical Debt Items that were "added" in their evolution status.

Adjust productivity figures for the current release by including the remediation effort of source code patterns implemented in the current release but not remediated until a future release. Remediation effort passed to future revisions is often counted as new work rather than rework, thus inflating productivity numbers.

8.4 Calculating a Contextual Technical Debt Measure (CTDM)

The Contextual Technical Debt Measure (CTDM) is an alternative to the Automated Technical Debt Measure, because it is adapted to the context of a specific organization or application. The adaption process is multifaceted and concerns one or more of the following non mutually aspects:

- the list of patterns to consider: a subset of the patterns from ASCMM, ASCRM, ASCPEM, and ASCSM; or a set including source code patterns not included in these Quality Characteristic measures,
- different values for remediation effort: different unadjusted Remediation Effort formulas, different unadjusted Remediation Effort formulas,
- the use of different formulas for adjustment factors, or their deactivation, and
- the use of additional adjustment factors.

However, these adjustments are incorporated at the expense of benchmarking, which cannot be accomplished with CTDM except among applications where the CTDM adjustments are identical.

The following sub-clauses illustrate sample variations regarding adjustment factors.

8.4.1 Technological Diversity

Principle

Adjust the Technological Diversity adjustment factor to better reflect the organization's ability to deal with occurrences involving multiple technologies.

Illustrations

1. Turn off (that is, ignore from computation) the Technological Diversity adjustment factor if the organization is organized around cross-technology teams.
2. Compute an alternative technological diversity penalty factor equal to the power of the number of distinct technologies, with a power value smaller than 1, to model a smooth coordination of different teams, and greater than 1, to model the infrequent involvement of different teams.

8.4.2 Exposure

Principle

Adjust the Exposure adjustment factor to better reflect the organization's ability to avoid destabilization of the software via automated testing.

Illustrations

1. Turn off (that is, ignore from computation) the Exposure adjustment factor if the organization is so mature regarding automated non-regression testing that teams can update the code without fear of side effects.
2. Compute an alternative exposure adjustment factor using one of the following formulas:
   - with an asymptote: max-1/(range number+1)^{power}
   - without an asymptote: (range number)^{power}
   - where range number is a logarithmic transformation of the exposure values, to account for combinatorial nature of the exposure and make them human-friendly: |log (exposure + 1)|

8.4.3 Concentration

Principle

Adjust the Concentration adjustment factor to better reflect the organization's strategy regarding the removal of Technial Debt occurrences.

Illustration

Turn off (that is, ignore from computation) the Concentration adjustment factor if the organization is willing to remove occurrences one at a time, that is, without considerations about other occurrences involving the same code elements.

8.4.4 Evolution Status

8.4.4.1 Occurrence

Principle

Adjust the remediation effort for a Technical Debt Item with an evolution qualification measure to factor in the opportunity to remove an occurrence more easily when it was injected into the software during the current release cycle.

Illustration

Consider an occurrence evolution reward factor of .50 for added occurrences.

8.4.4.2 Code elements

Principle

Adjust the remediation effort for a Technical Debt Item with an evolution qualification measure to factor in the opportunity to remove an occurrence more easily when the code elements involved were recently updated.

Illustration

Consider a code element evolution reward factor of .75 for updated code elements.

8.4.4.3 Limitation

Please note that the use of such adjustment factors makes the measures evolve over time, even if the software is not evolved in any way, as the occurrences "grow old" and the opportunity to remove them more easily vanishes.

8.5 Technical Debt Value Communication

The following scenarios illustrate ways in which the Automated Technical Debt Measure (ATDM) and the Contextual Technical Debt Measure (CTDM) can be used to help communicate about Technical Debt with non-technical audiences, facilitate acceptance, and reap the benefits of the Technical Debt metaphor.

8.5.1 Problem statement

ATDM and CTDM are estimating the effort to remove all occurrences of the selected patterns (from ASCSM, ASCRM, ASCPEM, ASCMM specifications, or from a user-defined list).

First, this is equivalent to a strategy of zero tolerance to defects which may be too stringent (and very likely unnecessary) to implement to all applications, as well as too expensive due to the sheer number of occurrences to remove. This leads to remediation effort values so large they are difficult to accept (even if justifiable), ultimately creating a push back against the whole measurement program.

Second, there is conceptual debate about the content of Technical Debt. Some say Technical Debt should only account for items that organizations have the intention to remove at some point in time. In other words, if organizations do not plan to completely remove all occurrences of each pattern, they are not to be considered in the Technical Debt measurement.

Third, some organizations manage quality objectives, such as internal or external Service Level Agreements. That is, they define some requirements on the number of issues that are considered acceptable. In this context, when quality objectives are set with a certain tolerance value, it means that only the occurrences whose removal is needed to reach the target level of tolerance will be effectively removed; the remaining occurrences will remain for lack of incentive to do so. In these frequent situations, the Technical Debt values that are meaningful for the management are the estimations of the effort and cost to reach target values (as opposed to the estimation of the effort and cost to get the total absence of occurrences).

8.5.2 Recommended approach

8.5.2.1 When quality objectives are set

CISQ recommends the computation of the amount of Automated Technical Debt Measure that is required to reach quality objectives that are set for each application.

As the scope of the measure is adjusted with contextual information, this computation should be exposed as a Contextual Technical Debt Measure to avoid confusion.

The immediate benefits of such approach are:

1. a more relevant value,
   because it would be aligned with organization's existing management practices, as opposed to a value relative to a hypothetical "zero tolerance" situation;
2. a more acceptable value,
   because it would be smaller, having filtered out effort and cost amounts that are not ultimately applicable.

8.5.2.2 When quality objectives are not set

In case there are no quality objectives set, CISQ recommends the computation of the amount of Automated Technical Debt Measure required to reach arbitrary yet meaningful quality levels (such as the sigma levels).

The immediate benefits are:

1. a perspective on quality levels,
   especially as there are no objectives set, to educate and help justify quality improvement initiatives (e.g., showing that there is an effort to plan to reach a sigma 3 level can resonate with non-technical management audience familiar with these concepts);
2. a more acceptable value,
   because it would be smaller, having considered the removal of some occurrences only (removing all occurrences would be completely unrealistic when dealing with an application for which there are no objectives set).

8.5.3 Limitations

Benchmarking

The adjustments regarding the tolerance are incorporated at the expense of benchmarking, which cannot be accomplished with CTDM except among applications where the CTDM adjustments are identical or acceptably different.

"Acceptably different" means there are differences in the adjustment criteria but that the organization is accepting and adhering to these differences and their impact on the way to interpret the results.

As an example, if two applications are assigned different tolerance levels, the organization must use the CTDM measures knowingly: the measured values shall not be used to compare the Technical Debt for these two applications but they shall be used to compare the distance to their respective quality objectives, using a Technical Debt metaphor.

Value range

As soon as the tolerance level is not zero, this means that some occurrences will have to be removed and some occurrences will be allowed to remain.

Each of the candidate occurrence for any given pattern leads to the same unadjusted remediation effort. However, as soon as the adjustment factors kick in, the adjusted remediation effort will very likely differ.

Therefore, the effort required to remove enough occurrences to reach the quality objective for this pattern becomes a value range, with a minimum value obtained by targeting the occurrences with the smaller adjusted remediation effort values, and with a maximum value obtained by targeting the occurrences with the largest adjusted remediation effort values. Obviously, to keep using a single value, the median or the mean value can be used.