**Date:** June 15, 2016

# Automated Enhancement Points

*Beta 1*

---

**OMG Document Number:** **ptc/2016-06-03**

**Standard document URL:** **http://www.omg.org/spec/AEP/1.0**

Associated files: Normative:
http://www.omg.org/spec/AEP/20151204/AutomatedEnhancementPoints.xmi

---

*Submitted by:*

CAST Software

*Supporting organizations:*

Accenture
Huawei

## USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

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

## LICENSES

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

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

## PATENTS

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

## GENERAL USE RESTRICTIONS



## DISCLAIMER OF WARRANTY

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

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

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

RESTRICTED RIGHTS LEGEND

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

TRADEMARKS

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

COMPLIANCE

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

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

**OMG's Issue Reporting Procedure**

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

# Table of Contents

# Table of Figures



# Table of Tables

# Preface

## About the Object Management Group

### OMG

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

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

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

### OMG Specifications

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

**Business Modeling Specifications Middleware Specifications**

- CORBA/IIOP
- Data Distribution Services
- Specialized CORBA

**IDL/Language Mapping Specifications**

**Modeling and Metadata Specifications**

- UML, MOF, CWM, XMI
- UML Profile

**Modernization Specifications**

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

- CORBAServices
- CORBAFacilities

**CORBA Embedded Intelligence Specifications**

**CORBA Security Specifications**

**OMG Domain Specifications**

**Signal and Image Processing Specifications**

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

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

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

## Issues

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

# 1. Scope

## 1.1 Purpose

The purpose of this specification is to establish an automated sizing measure, Automated Enhancement Points, which solves specific problems with OMG's approved specification for Automated Function Points when used to measure maintenance and enhancement work performed between two revisions of the software. Current functional sizing measures frequently produce inaccurate and misleading results when looking at functional size evolution and comparing the value to the effort expended during maintenance and enhancement. This problem is especially acute for analyzing productivity or evaluating contract performance. The solution proposed in Automated Enhancement Points establishes a standard measure that addresses 1) the complexity of the requested change, 2) anomalies in counting practices, and 3) measurement of the non-functional elements of an application. Automated Enhancement Points extend beyond the user transactional functions of a business application, the domain of functional measurement, to incorporate the components and elements that manage the non-functional, structural requirements of the application and allow it to perform in a modern multi-technology, multi-platform environment. As an additional benefit, this expansion beyond functional elements provides a foundation for extending automated software size measurement to organizations such as the Industrial Internet Consortium that involve highly algorithmic or embedded software, a domain that traditionally measured size only in lines of code.

## 1.2 Problems in Sizing Maintenance and Enhancement Work

There are three primary problems when measuring the size of work completed during maintenance and enhancement activities with functional measures such as Automated Function Points. The first problem involves differences in complexity of satisfying a change request. The second problem involves a counting anomaly in software sizing analysis where some activities to go unmeasured and others cancel out size gains with deletions. The third problem involves the failure to account for work performed on code that is not addressed by either IFPUG counting practice guidelines (ISO/IEC, 2009) or Automated Function Points (OMG, 2014). All three of these issues can cause excessive and unaccounted for variation in productivity analyses. Without correcting for this variation, productivity analyses can be misleading and effort estimates can have unacceptably large margins of error.

### 1.2.1 Problem 1 — Complexity of Changes

Requests to make changes involving enhancements, deletions, or modifications to an application can vary widely in difficulty although they affect the same number of Automated Function Points. The difficulty involved in making a change can reside in either the complexity of the change or the complexity of the code to which the change is being made, or both. Some changes are local and limited to a single object or file, while others require changes or have impacts propagated across several objects or files. The effort involved in these changes is quite different because of the need to examine possible unintended side

effects from more complex changes and propagated effects. When the number of Automated Function Points does not differ between two different levels of change complexity, the effort required to implement the more complex change can be severely underrepresented.

### 1.2.2 Problem 2 — Counting Anomalies

Maintenance and enhancement activities can involve the addition, modification, or deletion of code within a revision. When all of the affected code is measured within the functional elements of an application, the functional size of added code can be partially or fully offset by the functional size of deleted code between two revisions. For instance, current counting practices can cause the addition of 50 Automated Function Points to the code to be counted as a contribution of only 5 Automated Function Points if 45 Automated Function Points of dead or unneeded code are deleted from the application. Although substantial effort have been expended on these maintenance and enhancement activities, the final functional size measure for the change between two revisions is small or even zero, implying that little to no work was performed since the previous revision. In addition, modifying code will not change the number of manually counted or Automated Function Points if it changes the attributes of, but not the number of, functional elements counted.

As a result of these counting anomalies, the amount of work performed is frequently underrepresented by IFPUG counting guidelines or Automated Function Point counters. Productivity, effort, or cost analyses using size measures with these counting anomalies can contain wide variations in results that are caused more from unrepresentative calculations than from actual performance. Consequently, productivity analyses, calibration of cost estimates, benchmarking, and other analyses that compare size measures between revisions are inaccurate.

### 1.2.3 Problem 3 — Unaddressed Structural Components

Automated Function Points and the International Function Point Users Group (IFPUG) counting guidelines from which they were derived only address the user transaction components of a business application, leaving as much as half the application unaddressed in when measuring size. Both automated and manual Function Points measure size as it relates to elements of the data flow and storage that support the user transactions within an application. The pieces of software that are unaddressed by traditional functional size measures are designated as non-functional because they shall be performed by the software in order to support the completion of a user transaction. In essence, functional components allow users to perform their transactions, while the non-functional, structural components address the technical requirements that allow user transactions to be performed. Although functional requirements are usually platform-independent, functional requirements that primarily reside in the structural aspects of a system are dependent on attributes of the application's architecture, technology, and platform environment.

Examples of non-functional, structural components and elements of software include, but are not limited to:

- Validating data entry
- Mathematical operations
- Embedded/real-time response

- Formatting data
- Managing interactions between technologies or platforms
- Providing help

The problems described in clauses 1.2.2 and 1.2.3 are summarized visually in **Error! Reference source not found.**, with a focus in Figure 2 on problem from clause 1.2.2. The problem described in clause 1.2.1 is summarized visually in **Error! Reference source not found.**, where LOC = Lines of Code.

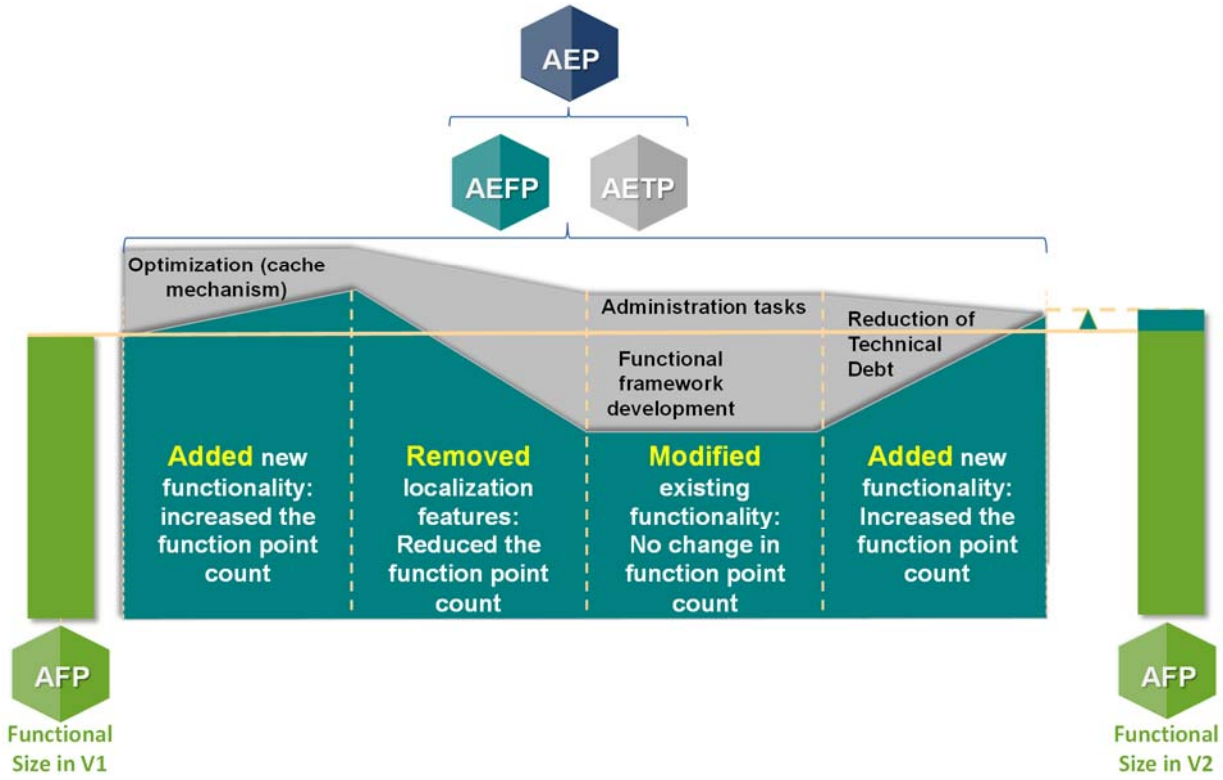


**Figure 1 Problems in using Automated Function Points to size maintenance and enhancement work.**

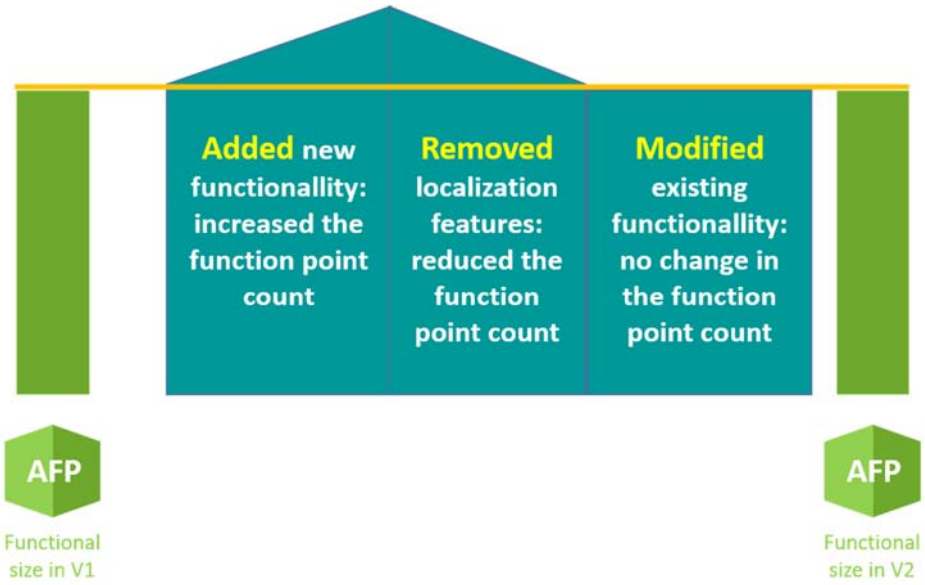


**Figure 2 Counting anomalies in using Automated Function Points to size maintenance and enhancement work – full offset of added function points by deleted function points.**

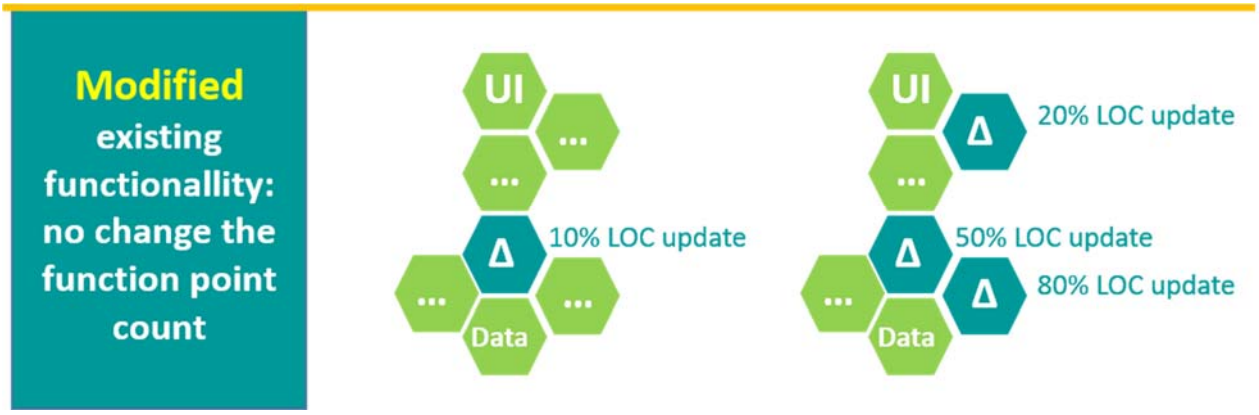


**Figure 3 Complexity of changes issue when using Automated Function Points to size maintenance and enhancement.**

## 1.3 Limitations of Existing Solutions to the Enhancement Sizing Problem

Solutions have been proposed for providing a function point-based measure to solve maintenance and enhancement sizing problems have been proposed by NESMA and IFPUG. The Netherlands Software Measurement Users Association (NESMA) proposed a method for solving the first two issues in clause 1.2,

that is, removing the counting anomalies and adjusting for the complexity of the change. Their *Function Point Analysis for Software Enhancement* (NESMA, 2009) is a method for evaluating change requests and enhancement proposals to estimate the number of Function Points to be credited for maintenance and enhancement work based on data evolutions. Although Automated Enhancement Points will build on some of NESMA's concepts, the NESMA method does not specify how to count the non-functional elements of an application, nor does it provide a specification to deal with algorithmic non-data-related evolutions.

The International Function Point Users Group (IFPUG) is developing a method for measuring the non-functional attributes of an application and its environment called the *Software Non-functional Assessment Proc*ess (SNAP). SNAP measures assess 14 non-functional elements in 4 categories as shown in Table 1. Computational elements within an application that perform these non-functional actions are not sized by traditional Function Point measures. Consequently, SNAP refers to these non-functional elements as the Technical components of an application. The specification of Automated Enhancement Points will retain 'Technical' in referring to the non-functional components of an application and their measures.

| Data Operations | Interface Design | Technical Environment | Architecture |
|---|---|---|---|
| Data entry | User interface changes | Multiple platforms | Component-based software |
| Logical/math operations | Help methods | Database technology | Multiple I/O interfaces |
| Data formatting | Multiple input methods | Batch processes | |
| Internal data movements | Multiple output methods | | |
| Delivering added value to users by data configuration | | | |

**Table 1 Non-functional (Technical) categories and elements in IFPUG's SNAP method**

SNAP is defined primarily for sizing an application from its requirements and is not defined in a manner that allows them to be calculated from source code. Consequently, SNAP relies on requirements analysis to assign the right categories and sub-categories to non-functional elements in an application. Since this determination cannot be achieved through automated analysis of the source code, the Automated Enhancement Points specification relies on analyzing changes in the software source code, regardless of the SNAP categories to which they would be assigned. Since SNAP measures are separate from Function Point counts within the IFPUG guidelines, no single size measure is calculated that incorporates both. Finally, SNAP measures provide a sizing specification that can be automated only for data element evolution. Therefore, while SNAP provides conceptual input, it is not sufficiently detailed for automating the sizing of work performed during a revision or revision.

## 1.4 Development of the Automated Enhancement Points Measure

The Consortium for IT Software Quality (CISQ) is a program of the OMG to create specifications for automating standard measures of software size and quality attributes from source code and submit them to OMG for approval as standards. Once approved these measures can be used by IT organizations, IT service providers, and software vendors in contracting, developing, testing, and accepting software applications. The Automated Function Points specification was approved as an OMG standard in 2014 and was quickly adopted by numerous public and private IT organizations.

One frequent use case motivating IT organizations to deploy Automated Function Points was for evaluating the productivity of maintenance and enhancement tasks. However, they often found large variations in their results that were difficult to interpret. Small changes often produced large Automated Function Point counts, while large changes often produced very few. Since they were unable to interpret productivity information reliably, CISQ sponsors prioritized this problem for solution. From Q4 2014 through Q3 2015 CISQ sponsors developed the concept and specification for Automated Enhancement Points. Two sponsors ran scripts on test applications to evaluate the measure's performance. The following specification represents a consensus among CISQ sponsors on a specification for Automated Enhancement Points. The name Enhancement Points was chosen because the objective of the measure is to size maintenance and enhancement work with a measure that sizes more than the functional attributes of an application.

# 2. Conformance

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

- ***Automated***—the analysis of the source code and the actual counting shall be fully automated. The initial inputs required to prepare the source code for analysis include the source code of the application, the artifacts and information needed to configure the application for operation, and any available description of the architectural layers in the application.
- ***Objective***—after the source code has been prepared for analysis using the information provided as inputs, the analysis, calculation, and presentation of results shall not require further human intervention. The analysis and calculation shall be able to repeatedly produce the same results and outputs on the same body of software.
- ***Transparent***—implementations that conform to this specification shall clearly list all source code (including versions), non-source code artifacts, and other information used to prepare the source code for submission to the analysis.
- ***Verifiable***—compliance with this specification requires that an implementation state the assumptions and heuristics it uses in sufficient detail that the calculations can be independently verified by third parties. Clause 7.21 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

The following OMG and ISO 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 Meta-model, version 1.3 (KDM), formal/2011-08-04
- Structured Metrics Meta-model, version 1.0 (SMM), formal/2012-01-05
- 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 (OCL), formal/2014-02-03
- Automated Function Points (AFP), formal/2014-01-03
- ISO/IEC 20926:2009 Software and system engineering -- Software measurement -- IFPUG Functional Size Measurement Method 2009.
- ISO/IEC 24570:2005 Software engineering -- NESMA functional size measurement method version 2.1 -- Definitions and counting guidelines for the application of Function Point Analysis

## 3.2 Non-normative


- Consortium for IT Software Quality (2010). http://www.it-cisq.org
- IFPUG (2014). Software Non-functional Assessment Practices Manual 2.1, Princeton Junction, NJ: International Function Point Users Group.
- McCabe, T. (1976). A measure of software complexity. IEEE Transactions on Software Engineering.
- NESMA (2009). Function Point Analysis for Software Enhancement, Version 2.2.1. Amsterdam: Netherlands Software Measurement Users Association.

# 4. Terms and Definitions

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

**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.

**Application Scope**

The Application Scope is composed of all computational objects within the boundary of a software application.

**Artifact**

Artifact is a computational object in a software application that is callable by name to perform some processing that can be either functional or technical. *(*KDM's code:MethodUnit, code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored', with non-empty 'name' attribute; *see Appendix A for examples in mainstream technologies)*

**Artifact Effort Complexity**

The Artifact Effort Complexity assesses the complexity of implementing the Artifact or changes to it based a composite score of five software metrics that assess the complexity of the software environment in which the Artifact is embedded: McCabe Cyclomatic Complexity, Lines of Code, Lines of Comment Code, Fan-In, and SQL Complexity

**Automated Enhancement Function Points (AEFP)**

The Automated Enhancement Function Points is an automated measure for sizing changes made to computational objects in the Functional Output between two revisions of an application. *(adapted from NESMA, 2009)*

**Automated Enhancement Function Points Scope, a.k.a. Functional Output**

The Automated Enhancement Function Points Scope is composed of all computational objects within the boundary of a software application that compose the modified functional features available in evolved user transactions and are measured by Automated Function Points.

**Automated Enhancement Points (AEP)**

The Automated Enhancement Points is an automated measure for sizing changes made to computational objects in the Application Scope between two revisions of an application.

**Automated Enhancement Technical Points (AETP)**

The Automated Enhancement Technical Points is an automated measure for sizing the changes made to computational objects in the Technical Output between two revisions.

**Automated Enhancement Technical Points Scope, a.k.a. Technical Output**

The Automated Enhancement Technical Points Scope is composed of all computational objects within the Automated Technical Points Scope that are added, modified, or deleted between two revisions.

**Automated Function Points (AFP)**

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

**Automated Function Points Data Entity**

Automated Function Points Data Entity is a data element within the Automated Function Point Scope that supports the implementation of data entities. *(Automated Function Points, formal 2014-01-03)*

**Automated Enhancement Function Points Equivalent ($AFP_{eq}$)**

The Automated Function Points Equivalent is an adjustment of Automated Enhancement Technical Points using the Equivalence Ratio to convert Implementation Points calculated for the Automated Enhancement Technical Points Scope into a statistically equivalent value of Automated Enhancement Function Points.

**Automated Function Points Implementation Artifacts**

Automated Function Points Implementation Artifacts are all Artifacts within the Automated Functional Points Implementation Scope.

**Automated Function Points Implementation Scope**

Automated Function Points Implementation Scope is composed of all computational objects within the boundary of a software application that compose the functional features available and are measured by Automated Function Points.

**Automated Function Points Transaction Entry Point**

Automated Function Points Transaction Entry Point is a computational object that support the interface with end users or third party applications, and used in the Automated Function Points specification as starting points of code paths towards data entities. *(formal 2014-01-03)*

**Automated Function Points Transaction Implementation Scope**

Automated Function Points Transaction Implementation Scope is composed of all computational objects in the Automated Function Points Implementation Scope that are within the boundary of a single transaction and are measured by Automated Function Points.

**Automated Technical Points (ATP)**

Automated Technical Points is a specification for automating the counting and sizing the non-functional, structural (Technical) elements of an application (i.e., all computational objects not included in the Automated Function Points Implementation Scope) in a manner similar to that used in calculating Automated Function Points.

**Automated Technical Points Implementation Artifacts**

Automated Technical Points Implementation Artifacts are all artifacts within the Automated Technical Points Scope.

**Automated Technical Points Implementation Scope**

Automated Technical Points Implementation Scope is composed of all Artifacts within the boundary of a software application that create and support the software technical foundation but are not measured by Automated Function Points, that is, Automated Technical Points Scope is the non-overlapping complement of Automated Function Points Scope with respect to the whole Application.

**Boundary**

The Boundary is a conceptual interface between an ensemble of computational objects and entities external to the boundary with which they interact; thus boundaries can be defined at many levels such as the whole application, any rigorously defined scope within the application, transactions, etc.

**Complexity Factor (CF)**

The Complexity Factor is a value used in calculating Automated Enhancement Function Points for transactions that adjusts for different complexities between additions, modifications, or deletions, and as well as the contributions of shared objects.

**Computational Object**

A Computational Object is a code element that can be detected during static analysis (KDM's code:ComputationalObject).

**Cyclomatic Complexity**

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

**Effort**

Effort measures an organization's investment in a project or defined collection of tasks that is measured in person hours, person days, or other locally defined unit of human work that is consistent in the work categories, staff positions, and time periods to be included in the measure.

**Effort Complexity (EC)**

The Effort Complexity assesses the complexity of adding, modifying, or deleting an Artifact based a composite score of five software metrics that assess the complexity of the software environment in which the Artifact is embedded, that is, its size, comment level, algorithmic complexity, data access complexity, and coupling.

**Enhancement**

An Enhancement is a change involving an addition, modification, or deletion applied to the software base of instructions comprising an application.

**Equivalence Ratio (ER)**

The Equivalence Ratio is an application-specific and revision-specific conversion ratio, for converting size measured in Implementation Points to a measure that is statistically equivalent to Automated Function Points.

**Function Points (FP)**

Function Points is a measure of software size calculated according to the counting practices of the International Function Points User Group guidelines. *(IFPUG, 2014)*

**Implementation Complexity**

The Implementation Complexity is a measure, at several levels, expressed in Implementation Points, defined by the sum of the Effort Complexities of all Artifacts within the scope boundary selected.

**Implementation Points (IP)**

Implementation Points is a software sizing measure that accounts for the difficulty of implementing or changing software Artifacts by aggregating the Effort Complexity scores for Artifacts within the scope of the measure to which Implementation Points are applied.

**Software Measure Element**

A Software Measure Element is a measure defined in terms of an attribute of software that affects size, and the measurement method for quantifying it, including optionally the transformation by a mathematical function. *(adapted from ISO/IEC 25023)*

**Software Product**

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

**Technical Scope of Measurement**

The Technical Scope of Measurement is composed of all software measure elements that are measured as part of calculating Automated Technical Points or Automated Enhancement Technical Points or any of their derivative measures.

# 5. Symbols (and Abbreviated Terms)

**AEFP** – Automated Enhanced Function Points

**AEP** – Automated Enhancement Points

**AER** – Adjusted Equivalence Ratio

**AETP** – Automated Enhancement Technical Points

**AFP** – Automated Function Points

**ATP** – Automated Technical Points

**CF** – Complexity Factor

**CISQ** – Consortium for IT Software Quality

**EC** – Effort Complexity

**ER** – Equivalence Ratio

**FP** – Function Points

**$AFP_{eq}$** – Automated Function Points equivalent

**IP** – Implementation Points

**KDM** – Knowledge Discovery Meta-model

**OCL** – Object Constraint Language

**OMG** – Object Management Group

**SMM** – Structured Metrics Meta-model

# 6. Method for Calculating Automated Enhancement Points (Normative)

## 6.1 Overview of Automated Enhancement Points

### 6.1.1 Application Level Scopes and Measures

The typical work performed by maintenance and development teams includes activities to support both the functioning of software features visible to end-users through their transactions (functional software), and the functioning of the software itself, as an autonomous entity that runs in a specific environment (technical software).

To solve the problem of unaddressed technical components, scopes of measurement shall be defined for all the computational elements, both functional and technical, of an application.

The computational elements containing the processing logic will be referred to as **Artifacts** throughout the remainder of this specification. Appendix A contains a list of Data Types that are considered Artifacts in programming languages commonly used in IT systems.

Thus, the Artifacts within the boundary of an application will be distributed among the following scopes:

- **Application Scope**—all Artifacts within the boundary of a software application.
- **Automated Function Point Scope**—all computational elements within the boundary of a software application that compose the functional features available in user transactions and are measured by Automated Function Points.
- **Automated Technical Point Scope**—all computational elements within the boundary of a software application that are not measured by Automated Function Points, that is, Artifacts which construct the technical foundation that enables the execution of the application's functional features.

Within these scopes we can define two size measures:

- **Automated Function Points (AFP)**—the measure specified in OMG's approved specification *formal/2014-01-03.*
- **Automated Technical Points (ATP)**—a sizing measure for the Artifacts of an application not measured by Automated Function Points. Although this measure is outside the scope of this specification, the computational process for producing Automated Enhancement Technical Points provides a blueprint for its calculation in the full Automated Technical Point Scope.

### 6.1.2 Application Maintenance and Enhancement Work Scopes and Measures

The calculation of Automated Enhancement Points requires three well-defined scopes of measurement within the Application Scope and its component Functional and Technical Scopes, aligned on the three application level scopes defined in previous section but limited to computational objects involved in maintenance and enhancement work.

These scopes of computational objects are the following:

- **Automated Enhancement Function Point Scope**—all computational objects (measured elements of data or transactional functions) within the Automated Function Points Scope that have been changed (added, modified, or deleted) between two revisions of an application.

- **Automated Enhancement Technical Point Scope**—the computational objects within the Automated Technical Points Scope that have been changed (added, modified, or deleted) between two revisions of an application.
- **Automated Enhancement Point Scope**—all computational objects within the combined Automated Enhancement Function Points Scope and Automated Enhancement Technical Points Scope between two revisions of an application.

An automated measure is specified for sizing the maintenance and enhancement work within each of the three scopes defined in the previous paragraph.

- **Automated Enhancement Function Points (AEFP)**—an Automated Function Point score calculated on only those computational elements within the Automated Enhancement Function Point Scope.
- **Automated Enhancement Technical Points (AETP)**—an automated measure for sizing the changes made to computational elements within the Automated Enhancement Technical Point Scope that is calculated as the sum of their Effort Complexities.
- **Automated Enhancement Points (AEP)**—an automated measure for sizing changes made to computational objects in the Automated Enhancement Point Scope and calculated as the sum of AEFP and AETP.

The relationships between these three scopes of measurement are presented in Figure 4. The content and sizes of these three scopes are specific to a pair of revision since the ensemble of computational objects added, modified, or deleted are dependent of the selected revisions. Therefore, all measures related to Automated Enhancement Points are calculated uniquely for changes from a specific revision to another specific revision, and these measures can vary widely across revision pairs based on differences in the scopes resulting from the changes implemented.

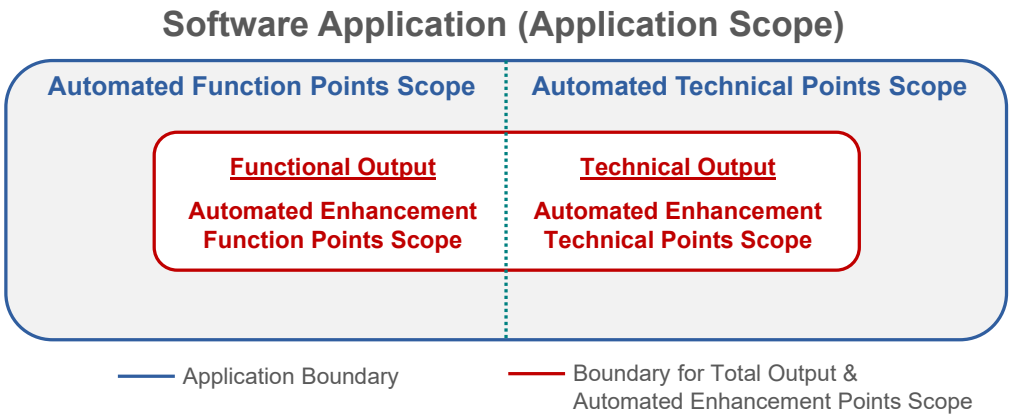


**Figure 4 Scopes of Measurements for Components of Automated Enhancement Points**

The combination of Automated Enhancement Function Points and Automated Enhancement Technical Points, called Automated Enhancement Points, can be used in productivity measurement and analysis programs.

### 6.1.3 Adjustment Factors and Implementation Points

The problems listed in clauses 1.2.1 and 1.2.2 involve difficulties in relating maintenance and enhancement effort to software changes. These difficulties are created by differences in the complexities of the changes being implemented and the software environment in which they are implemented. These measures shall be initially calculated on each Artifact involved in a change (added, modified, and deleted). The complexity adjustments used in calculating the constituent measures of Automated Enhancement Points include:

- **Effort Complexity (EC)**—assesses the complexity of adding, modifying, or deleting an Artifact based on a composite score of five software metrics that assess the complexity of the software environment in which the Artifact is embedded, that is, its size, comment level, algorithmic complexity, data access complexity, and coupling.
- **Complexity Factor (CF)**—a composite score calculated from values assigned for the complexity of changes made to individual computational objects in a transaction (respectively a data entity), and then applied to adjust the Automated Enhancement Function Points of a transaction (respectively a data entity) to accurately reflect the complexity of the changes implemented, such as not double-counting the score contributions of Artifacts shared between transactions.

Transactions and data entities with the Automated Enhancement Function Point Scope are adjusted by the Complexity Factor to account for the amount by which the complexity of the changes made to them affects their implementation effort. Automated Enhancement Function Points are then calculated by multiplying the Automated Function Points of the evolved transactions and data entities by the Complexity factor.

$$\mathbf{AEFP = CF \times AFP_{evolved\ AFP}}$$

The problem listed in clauses 1.2.3 involve the absence of any measurement of the evolution of the technical foundation of the software. The missing measurement is addressed by:

- **Implementation Points (IP)**—a software sizing measure that accounts for the difficulty of implementing or changing computational objects by aggregating the Effort Complexity scores for Artifacts within any given scope.
- Four salient scopes are the Automated Function Point Scope, the Automated Technical Point Scope, the Automated Enhancement Function Point Scope and the Automated Enhancement Technical Point Scope of a revision, leading to $\mathbf{IP_{AFP}}$, $\mathbf{IP_{ATP}}$, $\mathbf{IP_{AEFP}}$, and $\mathbf{IP_{AETP}}$ sizing measures respectively.
- $\mathbf{IP_{AFP}}$ is used later in the process, in conjunction with **AFP** value, to compute the Equivalence Ratio between **AFP** and $\mathbf{IP_{AFP}}$ in this revision of this software, allowing to get an **AFP equivalent** value for $\mathbf{IP_{AETP}}$

### 6.1.4 Calculation of Automated Enhancement Points

At this stage Implementation Points associated with Automated Technical Enhancement Points and Automated Enhancement Function Points are not comparable since Automated Enhancement Function Points are calculated differently than Implementation Points of Automated Enhancement Technical Points. The next step is to adjust the Implementation Points associated with Automated Enhancement Technical Points to be statistically equivalent to Automated Enhancement Function Points using an equivalence ratio.

**Equivalence Ratio (ER)**—calculated by dividing the Automated Function Points score of an application by the Implementation Points score associated with all computational objects within the Automated Function Point Scope of the application.

$$ER = AFP / IP_{AFP}$$

To improve their accuracy, organizations should collect Equivalence Ratio values over the various revisions of the applications in their portfolio, along with other descriptive information about type of application, technology, architecture, platform, etc., in order to develop standard Equivalence Ratios for distinct classes of applications

The Equivalence Ratio is applied to the Implementation Points derived from the Automated Enhancement Technical Points score to create a unit of measure that is statistically equivalent to the Implementation Points measure derived from Automated Enhancement Function Points.

$$AETP = ER \times IP_{AETP}$$

Once applied, the two equivalent scores for Implementation Points can be combined to produce the Automated Enhancement Point score:

$$AEP = AEFP + AETP$$

Consequently,

- Effort expended on changes within the Automated Enhancement Function Points Scope should be correlated with the Automated Enhancement Function Points.
- Effort expended on changes within the Automated Enhancement Technical Points Scope should be correlated with the Automated Enhancement Technical Points.
- Effort expended on changes within the Automated Enhancement Points Scope should be correlated with Automated Enhancement Points which is calculated by summing the Automated Enhancement Function Points with the equivalence-adjusted Automated Enhancement Technical Points.

An Automated Enhancement Points score is expressed as a form of Automated Function Point measure since

- Automated Enhancement Function Points equal Automated Function Points multiplied by a simple modifier

- Automated Enhancement Technical Points equal Implementation Points multiplied by an Implementation Points-to-Automated Function Points equivalence ratio.

In organizations without any Function Point history / affinity, or organizations whose software is not a natural match for Automated Function Point computation (such as scientific computing), the sum of $\mathbf{IP_{AEFP}}$ and $\mathbf{IP_{AETP}}$ proves a valuable alternative. $\mathbf{IP_{AEFP}} + \mathbf{IP_{AETP}}$ focusses on the algorithmic part of the software evolutions.

## 6.2 Developing the Application Model

### 6.2.1 Overview

The calculation of Automated Enhancement Points is performed between two revisions of the software, which are called “FromRevision“ and “ToRevision“, “ToRevision“ being the more recent of the two revisions.
Both revisions shall be analyzed to create an Application Model of the software for each revision or revision. The Application Model is composed of computational objects in the source code and their relationships. Some of these computational objects contain processing rules and logic. These computational objects, the ones with a name and that contain application logic to support the software processing, are called **Artifacts**. This means not all computational objects are Artifacts; for instance, data elements are not Artifacts. These are primarily the Methods, Functions, and Procedures from AFP 1.0 section 6.5. Appendix A lists Data Types that are considered Artifacts in many of the most frequently used IT systems programming languages.

The Application Model is used to define the

- **Transaction Automated Function Points Implementation Scope** of each transaction, i.e., the list of computational objects referenced by the AFP transaction entry points, as identified in the AFP specifications (§ 6.5.3);
- **Data Automated Function Points Implementation Scope** of each data entity, i.e., the list of computational objects owned by the AFP data entity, as identified in the AFP specifications (§6.5.2).

The union of all Transaction Automated Function Points Implementation Scopes composes the **Automated Function Points Scope** of the application. By complementarity, all computational objects in the Application Model but not in the Automated Function Points Scope compose the **Automated Technical Points Scope** of the application.

### 6.2.2 Comparison with the Application Model required by AFP 1.0

As with AFP 1.0, the Application Model is produced by analyzing the source code of the application to be sized. It shall contain the computational elements of the application and their relationships.
This is still the case with the current specifications.
However, there are some additional requirements:

- Cover both “FromRevision“ and “ToRevision“ revisions of the application (analysis of one revision or revision is not enough)

- Identification of all computational elements with an evolvedTo/evolvedFrom relationship, i.e., code elements “FromRevision“ that are found in “ToRevision“ as an evolved version of the computational element or go unchanged.
- Building directed graphs representing the direct dependencies of objects starting from the transactional entry points (no longer limited to the sole directed graphs connecting data elements and transactional entry points).
- Identification of all named computational elements containing processing logic (Artifacts) within and outside of the directed graphs
- Identification of all named computational elements containing processing logic (Artifacts) that are shared by multiple directed graphs

### 6.2.3 Representation in SMM of the two revisions

SMM enables the following modeling:

- One Observation of both revisions so that the base Application Model contains all required items
- Two Observation Scopes for this Observation to easily distinguish between the two revisions
- A set of Measures, Scopes, recognizer Operations which target any specific revision, the “FromRevision“ or “ToRevision“ revision.
- A set of Measures, Scopes, recognizer Operations which target both the “FromRevision“ and “ToRevision“ revisions at the same time

### 6.2.4 Detection of Transactional and Data Functions

Unchanged from AFP 1.0 section 6.5

### 6.2.5 Detection of Transaction AFP Implementation Scope

Transaction AFP Implementation Scopes are directed graphs:

- starting from the software User Interface (or public API),
- using field use, state change, method and function invocation, class inheritance, and interface implementation dependency relationships (that is, dependencies as defined in AFP 1.0 section 6.5),
- and ending with data functions, software boundary, or the absence of a further dependency relationship.

Contrary to AFP 1.0 specification requirements for which the existence of at least one code path between the transaction entry points and the data functions were enough, the comprehensiveness of the content of these directed graphs is critical for computing some of the intermediate measures in this specification and impacts the final calculation of Automated Enhancement Points.

The KDM references to compute Transaction AFP Implementation Scope are:

- logical block units (action:BlockUnit)
- composed of computational objects (code:ComputationalObject)

- connected via callable relations (action:CallableRelations), data relations (action:DataRelations), class relations (code:Extends), and interface relations (code:Implements)
- starting with a user interface (UI:UIDisplay)

### 6.2.5 Detection of Data AFP Implementation Scope

Data AFP Implementation Scopes are all the computational objects that are owned by the code elements used to detect the Data Function. “owned” refers to the special container relationship defined in the introduction of the SubPart I of KDM 1.3.

### 6.2.6 Detection of Artifacts

Artifacts are all named computational objects that contain application logic to support the software processing.
The KDM references to identify Artifacts are:

- code:MethodUnit,
- code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored',
- with non-empty ‘name’ attribute.

In addition:

- An Artifact cannot belong to another Artifact so as to avoid any duplicate count of the value of their code metrics (e.g.: counting SQL complexity of both parent and child would-be Artifacts would over-estimate the SQL complexity present in the software).
- Preprocessor directives are not considered as Artifacts even though some of them can alter the behavior of real Artifacts.

Each Artifact shall be analyzed in both revisions to determine whether it is:

- Added—when it exists in revision “ToRevision“ while it didn’t exist in revision “FromRevision“
- Deleted —when it existed in revision “FromRevision“ while it doesn’t exist in revision “ToRevision“
- Modified—when it exists in both revisions but whose source code changed between revision “FromRevision“ and revision “ToRevision“, creating the evolvedFrom/evolvedTo association needed for historical measurement *(SMM clause §17.1)*

### 6.2.7 Code Metric Requirements to Compute Artifact Effort Complexity

In order to compute Artifact Effort Complexity (EC), the following code metrics are used to feed the EC formula:

- Fan-In
- McCabe Cyclomatic Complexity
- Number of code lines, excluding comments and empty lines
- Number of comment lines

When SQL queries are involved, the following additional code metrics are used:

- Number of SQL tables involved
- Number of subqueries

- Usage of ‘group by’ statement
- Usage of ‘update’ statement
- Number of columns returned by select statements

The KDM and SMM references for these requirements are:

- Artifact Cyclomatic Complexity (cf. SMM 1.0 chapter 19 section 19.3.2)
- Artifact number of Lines of Code (cf. SMM 1.0 chapter 19 section 19.2.4 and specifically figure 19.13 for control element lines of code)
- Artifact commentedness ratio (cf. SMM 1.0 chapter 19 section 19.6)
- Artifact number of used SQL Tables: count of relational tables (data:RelationalTables) with read or write action from the Artifact (data:ReadsColumnSet or data:WritesColumnSet)
- Artifact number of used SQL Table Columns: count of item unit data elements (code:ItemUnit) from relational tables (data:RelationalTables) involved in data read action from the Artifact (data:ReadsColumnSet)
- Artifact number of subqueries: count of action elements performing a data write or read action (data:ReadsColumnSet or data:WritesColumnSet) nested in an action element performing a data read or write action (data:ReadsColumnSet or data:WritesColumnSet).
- Artifact number of used “Group By” SQL statement
- Artifact number of used “Update” SQL statement: count of data write action from the Artifact (data:WritesColumnSet)
- Artifact Fan-In: count of inward callable relations (action:CallableRelations) where the Artifact is the target control element (to:ControlElement)

## 6.3 Measurement Calculations for each Artifact

For each Artifact within the Application Scope, for both “FromRevision” and “ToRevision” Observation Scopes, its Effort Complexity shall be computed as follow.

The Effort Complexity EC of an Artifact is a function of the following five parameters:

1. Artifact Cyclomatic Complexity, as defined by McCabe, leads to Artifact assignment to one of the four categories below, using thresholds.
2. Artifact Size, as measured by the number of Lines of Code, leads to Artifact assignment to one of the four categories below, using thresholds.
3. Artifact Comment level, as measured by the number of Lines of Code and the number of Lines of Comment Code, leads to Artifact assignment to one of the four categories below, using thresholds
4. Artifact Coupling, as measured by the Fan-In, leads to Artifact assignment to one of the four categories below, using thresholds.
5. Artifact SQL Complexity, based on the number of SQL tables involved in the SQL queries, the number of subqueries, the presence of a group by statement, the update nature, and the

number of columns returned by select statements, leads to Artifact assignment to one of the four categories below, using thresholds.

### 6.3.1 Artifact Cyclomatic Complexity category assignment

Artifact shall be assigned one of the following four categories, based on the comparison of their Cyclomatic Complexity value and thresholds:

- Low Complexity category:
  Cyclomatic Complexity < 5
- Moderate Complexity category:
  Cyclomatic Complexity >= 5 && Cyclomatic Complexity < 15
- High Complexity category:
  Cyclomatic Complexity >= 15 && Cyclomatic Complexity < 30
- Very High Complexity category:
  Cyclomatic Complexity >= 30

Corresponding elements in the specification associated normative XMI file are:

(using scope Artifact: Scope, that is, named callable computational objects such as functions, procedures, and methods)

- ArtifactCyclomaticComplexity: NamedMeasure
  To compute the Artifact's McCabe Cyclomatic Complexity
- CyclomaticComplexityLevel: Ranking
  To assign a Cyclomatic Complexity level (low, moderate, high, very high) based on its value (ArtifactCyclomaticComplexity: NamedMeasure) and a set of thresholds (managed as parameters)

### 6.3.2 Artifact Size category assignment

Artifact shall be assigned one of the following four categories, based on the comparison of their number of Lines of Code and thresholds:

- Small Size category:
  Lines of Code < 10
- Average Size category:
  Lines of Code >= 10 && Lines of Code < 50
- Large Size category:
  Lines of Code >= 50 && Lines of Code < 200
- Very Large Size category:
  Lines of Code >= 200

Corresponding elements in the specification associated normative XMI file are:

(using scope Artifact: Scope, that is, named callable computational objects such as functions, procedures, and methods)

- ArtifactLinesOfCode: NamedMeasure
  To count the number of Lines of Code
- LinesOfCodeLevel: Ranking
  To assign a Sizing level (small, medium, large, very large) based on its value (ArtifactLinesOfCode: NamedMeasure) and a set of thresholds (managed as parameters)

### 6.3.3 Artifact Lack of Comment Level category assignment

Artifact shall be assigned one of the following four categories, based on the comparison of their Comment level and thresholds:

- Low Lack of Comment/Code category:
  Comment Level > 15%
- Average Lack of Comment/Code category:
  Comment Level <= 15% && Comment Level > 7%
- High Lack of Comment/Code category:
  Comment Level <= 7% && Comment Level > 3%
- Very High Lack of Comment/Code category:
  Comment Level <= 3%

Corresponding elements in the specification associated normative XMI file are:

(using scope Artifact: Scope, that is, named callable computational objects such as functions, procedures, and methods)

- ArtifactLinesOfCode: NamedMeasure
  To count the number of Lines of Code
- ArtifactLinesOfCommentCode: NamedMeasure
  To count the number of Lines of Comment Code
- ArtifactLinesOfCodeAndCommentCode: CollectiveMeasure
  To count the sum of the number of Lines of Code (ArtifactLinesOfCode: NamedMeasure) and the number of Lines of Comment Code (ArtifactLinesOfCommentCode: NamedMeasure)
- ArtifactCommentRatio: RatioMeasure
  To compute the ratio of Lines of Comment Code (ArtifactLinesOfCommentCode: NamedMeasure) per number of Lines of Code and Comment Code (ArtifactLinesOfCodeAndCommentCode: CollectiveMeasure)
- LackOfCommentLevel: Ranking
  To assign a Lack of Comment level (low, moderate, high, very high) based on its value (ArtifactCommentRatio: RatioMeasure) and a set of thresholds (managed as parameters)

### 6.3.4 Artifact Coupling category assignment

Artifact shall be assigned one of the following four categories, based on the comparison of their Artifact Coupling value and thresholds:

- Low Coupling category:
  Number of Callers < 4
- Moderate Coupling category:
  Number of Callers >= 4 && Number of Callers < 10
- High Coupling category:
  Number of Callers >= 10 && Number of Callers < 30
- Very High Coupling category:
  Number of Callers >= 30

Corresponding elements in the specification associated normative XMI file are:

(using scope Artifact: Scope, that is, named callable computational objects such as functions, procedures, and methods)

- ArtifactFanIn: DirectMeasure
  To count the number of references
- CouplingLevel: Ranking
  To assign a SQL complexity level (low, moderate, high, very high) based on its value (ArtifactFanIn: DirectMeasure) and a set of thresholds (managed as parameters)

### 6.3.5 Artifact SQL Complexity category assignment

Artifact shall be assigned one of the following four categories, based on the comparison of their Artifact SQL Complexity value and thresholds:

- Low SQL Complexity category:
  SQL Artifact Complexity < 10
- Moderate SQL Complexity category:
  SQL Artifact Complexity >= 10 && SQL Artifact Complexity < 40
- High SQL Complexity category:
  SQL Artifact Complexity >= 40 && SQL Artifact Complexity < 70
- Very High SQL Complexity category:
  SQL Artifact Complexity >= 70

With, the Artifact SQL Complexity rated from 0 to 100, 0 for lowest Complexity and 100 for Highest complexity.

SQL Artifact Complexity =
50 if Artifact with a Query on more than 4 tables
+ 10 if Artifact with a Subquery
+ 10 if Artifact with a GROUP BY
+ 10 if Artifact with a Complex SELECT clause
+ 10 if Artifact with an UPDATE statement

+ 10 if Artifact with Raw SQL Complexity Higher than 30

Corresponding elements in the specification associated normative XMI file are:

(using scope SQLQuery: Scope, that is SQL queries which reads and writes data)

- QueryNumberOfSQLTables: DirectMeasure
  To count the number of SQL tables used by a SQL query (using scope kdm:data::DataActions)

(using scope Artifact: Scope, that is, named callable computational objects such as functions, procedures, and methods)

- ArtifactMaxNumberOfSQLTablesPerQuery: CollectiveMeasure
  To compute at the Artifact level, the maximum number of SQL tables used by any of its contained SQL queries (QueryNumberOfSQLTables: DirectMeasure)
- ArtifactNumberOfGroupBySQLStatement: DirectMeasure
  To count the number of “Group By” SQL statement in Artifact’s SQL queries
- ArtifactNumberOfSQLSubqueries: DirectMeasure
  To count the number of sub-queries in Artifact’s SQL queries
- ArtifactNumberOfUpdateSQLStatement: DirectMeasure
  To count the number of “Update” SQL statement in Artifact’s SQL queries
- ArtifactNumberOfUsedSQLTableColumns: DirectMeasure
  To count the number of SQL table columns used in Artifact’s SQL queries
- ArtifactNumberOfUsedSQLTables: DirectMeasure
  To count the number of SQL tables used in Artifact’s SQL queries

(using scope Artifact: Scope, that is, named callable computational objects such as functions, procedures, and methods)

- wArtifactMaxNumberOfSQLTablesPerQuery: RescaledMeasure
  To compute the contribution to overall SQL complexity based on the maximal number of used SQL tables per SQL query in the Artifact (ArtifactMaxNumberOfSQLTablesPerQuery: CollectiveMeasure)
- wArtifactNumberOfUsedSQLTables: RescaledMeasure
  To compute the contribution to overall SQL complexity based on the number of used SQL tables in Artifact's SQL queries (ArtifactNumberOfUsedSQLTables: DirectMeasure)
- wArtifactNumberOfUsedSQLTableColumns: RescaledMeasure
  To compute the contribution to overall SQL complexity based on the number of used SQL table columns in Artifact's SQL queries (ArtifactNumberOfUsedSQLTableColumns: DirectMeasure)
- wArtifactNumberOfSQLSubqueries: RescaledMeasure
  To compute the contribution to overall SQL complexity based on the number of sub-queries in Artifact's SQL queries (ArtifactNumberOfSQLSubqueries: DirectMeasure)
- wArtifactNumberOfGroupBySQLStatement: RescaledMeasure
  To compute the contribution to overall SQL complexity based on the number of "Group By" Statement in Artifact's SQL queries (ArtifactNumberOfGroupBySQLStatement: DirectMeasure)
- wArtifactNumberOfUpdateSQLStatement: RescaledMeasure
  To compute the contribution to overall SQL complexity based on the number of "Update" Statement in Artifact's SQL queries (ArtifactNumberOfUpdateSQLStatement: DirectMeasure)
- ArtifactSQLComplexity: CollectiveMeasure
  To compute the overall SQL complexity value by summing the various contributions (wArtifactNumberOfUpdateSQLStatement: RescaledMeasure, wArtifactNumberOfGroupBySQLStatement: RescaledMeasure, wArtifactNumberOfSQLSubqueries: RescaledMeasure, wArtifactNumberOfUsedSQLTableColumns: RescaledMeasure, wArtifactNumberOfUsedSQLTables: RescaledMeasure, wArtifactMaxNumberOfSQLTablesPerQuery: RescaledMeasure)
- SQLComplexityLevel: Ranking
  To assign a SQL complexity level (low, moderate, high, very high) based on its value (ArtifactSQLComplexity: CollectiveMeasure) and a set of thresholds (managed as parameters)

### 6.3.6 Artifact Effort Complexity category assignment

Once the above assignments to categories are done for an artifact, the overall Effort Complexity score is then calculated as follows:

- An Artifact falls into the Very High Effort Complexity category, when ONE of the following is true:
  - Cyclomatic Complexity is Very High
  - SQL Complexity is Very High
  - Artifact Size is Very Large AND Lack of Comment index is Very High AND Artifact Coupling is Very High
- An Artifact falls into the High Effort Complexity category, when ONE of the following is true:
  - Cyclomatic Complexity is High

    - SQL Complexity is High
    - Cyclomatic Complexity is Moderate AND SQL Complexity is Moderate AND (Artifact Size is Very Large OR Lack of Comment is Very High OR Artifact Coupling is Very High)
    - Artifact Size is Large AND Lack of Comment is High AND Artifact Coupling is High
- An Artifact falls into the Moderate Effort Complexity category, when ONE of the following is true:
    - Cyclomatic Complexity is Moderate
    - SQL Complexity is Moderate
    - Cyclomatic Complexity is Low AND SQL Complexity is Low AND (Artifact Size is Large OR Lack of Comment is High OR Artifact Coupling is High)
    - Artifact Size is Average AND Lack of Comment is Average AND Artifact Coupling is Moderate
- An Artifact falls into the Low Effort Complexity category, otherwise.

Corresponding elements in the specification associated normative XMI file are:

(using scope Artifact: Scope, that is, named callable computational objects such as functions, procedures, and methods)

- wArtifactSQLComplexityLevel: RescaledMeasure
  *To compute the SQL complexity level bit into the overall effort complexity, based on its level (SQLComplexityLevel: Ranking)*
- wArtifactLinesOfCodeLevel: RescaledMeasure <- LinesOfCodeLevel: Ranking
  *To compute the Size level bit into the overall effort complexity, based on its level (LinesOfCodeLevel: Ranking)*
- wArtifactLackOfCommentLevel: RescaledMeasure <- LackOfCommentLevel: Ranking
  *To compute the Lack of Comment level bit into the overall effort complexity, based on its level (LackOfCommentLevel: Ranking)*
- wArtifactCouplingLevel: RescaledMeasure <- CouplingLevel: Ranking
  *To compute the Coupling level bit into the overall effort complexity, based on its level (CouplingLevel: Ranking)*
- wArtifactCyclomaticComplexityLevel: RescaledMeasure <- CyclomaticComplexityLevel: Ranking
  *To compute the Cyclomatic Complexity level bit into the overall effort complexity, based on its level (CyclomaticComplexityLevel: Ranking)*
- ArtifactEffortComplexityValue: CollectiveMeasure
  *To compute an Effort Complexity binary value summing the contributing bits (wArtifactSQLComplexityLevel: RescaledMeasure, wArtifactLinesOfCodeLevel: RescaledMeasure, wArtifactLackOfCommentLevel: RescaledMeasure, wArtifactCouplingLevel: RescaledMeasure, wArtifactCyclomaticComplexityLevel: RescaledMeasure)*
- ArtifactEffortComplexityIndex: RescaledMeasure <- ArtifactEffortComplexityValue: CollectiveMeasure
  *To compute an aggregated index value based on the presence of specific contributing bits in the Effort Complexity value (ArtifactEffortComplexityValue: CollectiveMeasure)*
- ArtifactEffortComplexityLevel: Ranking
  *To turn into the Effort Complexity level (low, moderate, high, very high) the Effort Complexity index (ArtifactEffortComplexityIndex: RescaledMeasure)*

### 6.3.7 Artifact Effort Complexity final value

Each Artifact Effort Complexity category gets an Effort Complexity value, which can be overridden for specific technologies. Default values are as follow:

- Very High Effort Complexity category: 1.2
- High Effort Complexity category: 0.7
- Moderate Effort Complexity category: 0.2
- Low Effort Complexity category: 0.1

Corresponding elements in the specification associated normative XMI file are:

(using scope Artifact: Scope, that is, named callable computational objects such as functions, procedures, and methods)

- ArtifactEffortComplexity: RescaledMeasure
  To compute the final Effort Complexity, based on the level (ArtifactEffortComplexityLevel: Ranking)

## 6.4 Measurement Calculations for each Transaction AFP Implementation Scope

Both revisions shall be analyzed and measured according to the Automated Function Point specification to identify transactional functions and data functions *(AFP clauses 6.5.2 and 6.5.3)*.
The Effort Complexity associated with each Transaction AFP are calculated as the sum of the Effort Complexities of all the Artifacts within the Transaction AFP Implementation Scope, that is:

$$\mathbf{EC_{Transaction\ AFP} = \Sigma\ (EC_{Artifact})}$$

Corresponding elements in the specification associated normative XMI file are:

(using scope UpdatedTransactionalAFP_Latest: Scope, that is, EO and EI with evolved implementation artifacts as defined by AFPTransactionImplementationArtifacts: OCLOperation)

- EffortComplexityTotalInLatest: CollectiveMeasure
  To compute the total Effort Complexity of updated transaction AFP in latest revision, summing the Artifact Effort Complexity (ArtifactEffortComplexity: RescaledMeasure) on Artifacts from its implementation scope in latest revision

(using scope UpdatedTransactionalAFP_Previous: Scope, that is, implementation artifacts as defined by AFPTransactionImplementationArtifacts: OCLOperation in the previous revision)

- EffortComplexityTotalInPrevious: CollectiveMeasure
  To compute the total Effort Complexity of updated transaction AFP in previous revision, summing Artifact Effort Complexity (ArtifactEffortComplexity: RescaledMeasure) on Artifacts from its implementation scope in previous revision

When dealing with the whole Application Scope, including both “FromRevision” and “ToRevision” Observation Scopes, additional Effort Complexity shall be computed:

- The Effort Complexity for the evolved Artifacts within the Transaction AFP Implementation Scope

$$\mathbf{{}_{Evolved}EC_{Transaction\ AFP} = \Sigma_{\ Evolved}\ (EC_{Artifact})}$$

Corresponding elements in the specification associated normative XMI file are:

(using scope ArtifactInUpdatedTransactionalAFP_Added: Scope, that is, added implementation artifacts as defined by AFPTransactionImplementationArtifacts: OCLOperation from updated transactional AFP)

- EffortComplexityAdded: CollectiveMeasure
  To compute the Effort Complexity of added Artifacts in updated transactional AFP, summing their Effort Complexity (ArtifactEffortComplexity: RescaledMeasure)

(using scope ArtifactInUpdatedTransactionalAFP_Deleted: Scope, that is, deleted implementation artifacts as defined by AFPTransactionImplementationArtifacts: OCLOperation from updated transactional AFP)

- EffortComplexityDeleted: CollectiveMeasure
  To compute the Effort Complexity of deleted Artifacts in updated transactional AFP, summing their Effort Complexity (ArtifactEffortComplexity: RescaledMeasure)

(using scope ArtifactInUpdatedTransactionalAFP_Updated: Scope, that is, updated implementation artifacts as defined by AFPTransactionImplementationArtifacts: OCLOperation from updated transactional AFP)

- EffortComplexityUpdated: CollectiveMeasure
  To compute the Effort Complexity of updated Artifacts in updated transactional AFP, summing their Effort Complexity (ArtifactEffortComplexity: RescaledMeasure)

(using scope UpdatedTransactionalAFP: Scope)

- EffortComplexityProcessed: CollectiveMeasure
  To compute the total Effort Complexity of evolved Artifacts in updated transactional AFP, summing the added, deleted, and updated values (EffortComplexityAdded: CollectiveMeasure, EffortComplexityUpdated: CollectiveMeasure, EffortComplexityDeleted: CollectiveMeasure)

- The Effort Complexity for the evolved shared Artifacts within the Transaction AFP Implementation Scope

$$_{\text{Evolved, Shared}}\mathbf{EC}_{\text{Transaction AFP}} = \Sigma_{\text{Evolved, Shared}}\,(\mathbf{EC}_{\text{Artifact}})$$

Corresponding elements in the specification associated normative XMI file are:

(using scope SharedArtifactInUpdatedTransactionalAFP_Added: Scope, that is, shared added implementation artifacts as defined by AFPTransactionImplementationArtifacts: OCLOperation from updated transactional AFP)

- SharedEffortComplexityAdded: CollectiveMeasure
  To compute the Effort Complexity of shared added Artifacts in updated transactional AFP, summing their Effort Complexity (ArtifactEffortComplexity: RescaledMeasure)

(using scope SharedArtifactInUpdatedTransactionalAFP_Deleted: Scope, that is, shared deleted implementation artifacts as defined by AFPTransactionImplementationArtifacts: OCLOperation from updated transactional AFP)

- SharedEffortComplexityDeleted: CollectiveMeasure
  To compute the Effort Complexity of shared deleted Artifacts in updated transactional AFP, summing their Effort Complexity (ArtifactEffortComplexity: RescaledMeasure)

(using scope SharedArtifactInUpdatedTransactionalAFP_Updated: Scope, that is, shared updated implementation artifacts as defined by AFPTransactionImplementationArtifacts: OCLOperation from updated transactional AFP)

- SharedEffortComplexityUpdated: CollectiveMeasure
  To compute the Effort Complexity of shared updated Artifacts in updated transactional AFP, summing their Effort Complexity (ArtifactEffortComplexity: RescaledMeasure)

(using scope UpdatedTransactionalAFP: Scope)

- SharedEffortComplexityProcessed: CollectiveMeasure
  To compute the total Effort Complexity of evolved Artifacts in updated transactional AFP, summing the added, deleted, and updated values (SharedEffortComplexityAdded: CollectiveMeasure, SharedEffortComplexityUpdated: CollectiveMeasure, SharedEffortComplexityDeleted: CollectiveMeasure)

## 6.5 Measurement Calculations for evolved Transaction and Data AFP

Compared to simple AFP measurement calculations, the analysis of transactional and data functions involves in the current specifications the following additional steps:

- Identification of evolved Transaction and Data AFP
- For all evolved Transaction and Data AFP
  - computation of an evolution **Complexity Factor**
  - computation of the **Automated Enhancement Function Points**

Regarding the identification of evolved Transaction and Data AFP:

- An **added transaction or data entity** exists in revision “ToRevision“ but didn’t exist in “FromRevision“, and its value in Automated Function Points is denoted as $\mathbf{AFP_{ToRevision}}$ , in essence its AFP value calculated in “ToRevision“.
- A **deleted transaction or data entity** no longer exists in revision “ToRevision“ but existed in revision “FromRevision“, and its size in Automated Function Points is denoted $\mathbf{AFP_{FromRevision}}$, in essence its AFP value calculated in “FromRevision“.

- A **split data entity** is a data entity whose DET, once part of the same data entity in revision "FromRevision", are part of more than one data entity in revision "ToRevision"
- A **merged data entity** is a data entity whose DET were part of more than one data entity in revision "FromRevision"
- A **data entity with changed type** is a data entity which was identified as an Internal Logical File in revision "FromRevision" (respectively an External Input File) and is identified as an External Input File in revision „ToRevision" (respectively an Internal Logical File)
- A **modified transaction** exists when one or more of the computational objects in its Implementation Scope is modified, added to the processing flow, or removed from the processing flow. Computational object modification status is based on their source code checksum, as measured in revisions "FromRevision" and "ToRevision". The AFP value of modified transactions is denoted as $\mathbf{AFP_{ToRevision}}$ in essence its AFP value calculated in "ToRevision".
- A **modified data entity** exists when one or more of the computational objects supporting the DET is modified, and when the data entity is not a split data entity or a merged data entity. The AFP value of modified data entities is denoted as $\mathbf{AFP_{ToRevision}}$ in essence its AFP value calculated in "ToRevision".

The amount of modification performed on an evolved transaction or data entity is accounted by the Complexity Factor (CF), which is used to adjust the transaction's or data entity's AFP value.

The Complexity Factor (CF) for transactional functions is determined as follows:

- Added transaction—CF value is always 1
- Deleted transaction—CF value is always 0.4
- Modified transaction—CF values for transactions that did not contain shared Artifacts that were changed are presented in Table 2 and are based on two input parameters:
  a. **Evolved Effort Complexity**—the share of the Effort Complexity of the full transaction that was affected by the Effort Complexities of the changed Artifacts.
  b. **Effort Complexity Variation**—a percentage of the total Effort Complexity of the full transaction, to account for the Effort Complexity that was added to or removed from changed Artifacts.

| Evolved Effort Complexity ($EC_{evolved}$) | Effort Complexity Variation ($EC_{variation}$) | | | |
|---|---|---|---|---|
| | ≤ ⅓ (x 100%) | ≤ ⅔ (x 100%) | ≤ 100% | > 100% |
| ≤ ⅓ x 100% | 0.25 | 0.50 | 0.75 | 1.00 |
| ≤ ⅔ x 100% | 0.50 | 0.75 | 1.00 | 1.25 |
| ≤ 100% | 0.75 | 1.00 | 1.25 | 1.50 |
| > 100% | 1.00 | 1.25 | 1.50 | 1.75 |

**Table 2 Determination of Effort Complexity Variation**

- CF values for transactions that did contain shared Artifacts that were changed are determined as follows:
    a. If 100% of the modified Artifacts in a transaction are shared components, CF = 0.25
    b. If more than 75% of the modified Artifacts are shared components, CF is capped at 0.50
    c. If more than 50% of the modified Artifacts are shared components, CF is capped at 0.75

Corresponding elements in the specification associated normative XMI file are:

- For the Effort Complexity variation of updated Transaction AFP

(using scope UpdatedTransactionalAFP: Scope)

- EffortComplexityTotalVariation: BinaryMeasure
  To compute the total Effort Complexity net variation, as the difference between Its latest value (EffortComplexityTotalInLatest: CollectiveMeasure) and its previous value (EffortComplexityTotalInPrevious: CollectiveMeasure)
- RatioEffortComplexityTotalVariation: RatioMeasure
  To compute the ratio of the Effort Complexity net variation (EffortComplexityTotalVariation: BinaryMeasure) divided by the total Effort Complexity value in previous revision (EffortComplexityTotalInPrevious: CollectiveMeasure)
- wRatioEffortComplexityTotalVariation: RescaledMeasure
  To compute the contribution of the ratio of Effort Complexity variation (RatioEffortComplexityTotalVariation: RatioMeasure) into the overall Complexity Factor

- For the processed Effort Complexity of updated Transaction AFP

(using scope UpdatedTransactionalAFP: Scope)

- RatioEffortComplexityProcessed: RatioMeasure
  To compute the ratio of the Effort Complexity processed (EffortComplexityProcessed: CollectiveMeasure) divided by the total Effort Complexity value in previous revision (EffortComplexityTotalInPrevious: CollectiveMeasure)
- wRatioEffortComplexityProcessed: RescaledMeasure
  To compute the contribution of the ratio of Effort Complexity processed (RatioEffortComplexityProcessed: RatioMeasure) into the overall Complexity Factor

- For the Complexity Factor of updated Transaction AFP before the shared processed Effort Complexity cap kicks in

(using scope UpdatedTransactionalAFP: Scope)

- sRatioEffortComplexity: CollectiveMeasure
  To compute the sum of the two Effort Complexity contributions ( wRatioEffortComplexityTotalVariation: RescaledMeasure, wRatioEffortComplexityProcessed: RescaledMeasure)
- RawUpdatedTransactionalAFPComplexityFactor: RescaledMeasure
  To compute a raw Complexity Factor for updated transactional AFP, based on the sum of Effort Complexity contributions (sRatioEffortComplexity: CollectiveMeasure)

- For the shared processed Effort Complexity cap and resulting Complexity Factor of updated Transaction AFP

(using scope UpdatedTransactionalAFP: Scope)

- RatioSharedEffortComplexity: RatioMeasure
  To compute the ratio of the shared Effort Complexity processed (SharedEffortComplexityProcessed: CollectiveMeasure) divided by the Effort Complexity processed (EffortComplexityProcessed: CollectiveMeasure)
- capRatioSharedEffortComplexity: RescaledMeasure
  To compute the capping value for the Complexity Factor of updated transactional AFP due to the sharing of Artifacts, based on the ratio of shared Effort Complexity (RatioSharedEffortComplexity: RatioMeasure)
- UpdatedTransactionalAFPComplexityFactor: CollectiveMeasure
  To compute the final value for the Complexity Factor of updated transactional AFP, as the minimum of the raw value (RawUpdatedTransactionalAFPComplexityFactor: RescaledMeasure) and the capping value (capRatioSharedEffortComplexity: RescaledMeasure)

- For the added and deleted Transaction AFP

(using scope AddedTransactionalAFP: Scope, that is, EO and EI added in the latest revision)

- AddedTransactionalAFPComplexityFactor: RescaledMeasure
  *The adjustment Complexity Factor to use for added transactional AFP*

(using scope DeletedTransactionalAFP: Scope, that is, EO and EI removed from the latest revision)

- DeletedTransactionalAFPComplexityFactor: RescaledMeasure
  *The adjustment Complexity Factor to use for deleted transactional AFP*

The Complexity Factor (CF) for data functions is determined as follows:

- Added data—CF value is always 1 for AFP calculated on added data, excluding 'added data' that result from a split and merge activity between the two revisions
- Deleted data—CF value is always 0.4 for AFP calculated on deleted data, excluding 'delete data' resulting from a split and merge activity between the two revisions
- Merged data—CF value is always 0.4 for AFP calculated on merged data
- Split data—CF value is always 0.4 for AFP calculated on split data
- Data with changed type—CF value is always 0.4 for AFP calculated on data with changed type (e.g., EIF ⇒ ILF) but no change of DET score
- Modified data—CF value of a modified data function is based on one input parameter as presented in Table 3:

| % of DETs affected by the change | ≤ ⅓ (x 100%) | ≤ ⅔ (x 100%) | ≤ 100% | > 100% |
|---|---|---|---|---|
| Modified Data CF | 0.25 | 0.50 | 0.75 | 1.00 |

**Table 3 Complexity Factor for Modified Data**

Corresponding elements in the specification associated normative XMI file are:

- For updated Data AFP

(using scope DETInUpdatedDataAFP_Added: Scope, that is, added DET owned by updated data AFP)

- AddedDETInLatest: Counting
  To count added DET in updated data AFP

(using scope DETInUpdatedDataAFP_Updated: Scope, that is, updated DET owned by updated data AFP)

- UpdatedDETInLatest: Counting
  To count updated DET in updated data AFP

(using scope DETInUpdatedDataAFP_Deleted: Scope, that is, deleted DET owned by updated data AFP)

- DeletedDETInLatest: Counting
  To count deleted DET in updated data AFP

(using scope UpdatedDataAFP: Scope, that is, EIF and ILF with evolved implementation as defined by AFPDataImplementationScope: OCLOperation)

- EvolvedDETInLatest: CollectiveMeasure
  To count evolved DET in updated data AFP as the sum of added, updated, and deleted DET
  (AddedDETInLatest: Counting, UpdatedDETInLatest: Counting, DeletedDETInLatest: Counting)

(using scope DETInUpdatedDataAFP_Previous: Scope, that is, DET owned by updated data AFP in previous revision)

- TotalDETInPrevious: Counting
  To count the total number of DET in updated data AFP in previous revision

(using scope UpdatedDataAFP: Scope)

- DETChangeRatio: RatioMeasure
  To compute the ratio of changed DET as the count of evolved DET (EvolvedDETInLatest: CollectiveMeasure) divided by the total number of DET in previous revision (TotalDETInPrevious: Counting)
- UpdatedDataAFPComplexityFactor: RescaledMeasure
  To compute the Complexity Factor of updated data AFP, based on the DET Change ratio (DETChangeRatio: RatioMeasure)

- For other evolved Data AFP

(using scope AddedDataAFP: Scope, that is, EIF and ILF added in the latest revision)

- AddedDataAFPComplexityFactor: RescaledMeasure
  The adjustment Complexity Factor to use for added data AFP

(using scope DeletedDataAFP: Scope, that is, EIF and ILF removed from the latest revision)

- DeletedDataAFPComplexityFactor: RescaledMeasure
  The adjustment Complexity Factor to use for deleted data AFP

(using scope MergedDataAFP: Scope, that is, EIFs and ILFs from previous revision merged into single EIF and ILF in the latest revision)

- MergedDataAFPComplexityFactor: RescaledMeasure
  The adjustment Complexity Factor to use for merged data AFP

(using scope SplitDataAFP: Scope, that is, single EIF and ILF from previous revision split into EIFs and ILFs in the latest revision)

- SplitDataAFPComplexityFactor: RescaledMeasure
  The adjustment Complexity Factor to use for split data AFP

This Complexity Factor is used to adjust AFP values for added and deleted transactions and data entities.

- $\mathbf{AEFP_{added/modified\ AFP} = CF_{added/modified} * AFP_{ToRevision}}$
- $\mathbf{AEFP_{deleted\ AFP} = CF_{deleted} * AFP_{FromRevision}}$
- $\mathbf{AEFP_{split/merged/changed\ type\ Data\ AFP} = CF_{split/merged/changed\ type} * AFP_{ToRevision}}$

Corresponding elements in the specification associated normative XMI file are:

- For AFP values

(using scope kdm:Core::Element, as defined in AFP 1.0)

- weightExternalOutput: NamedMeasure
  To compute the weight of External Ouput Transactional AFP
- weightExternalInput: NamedMeasure
  To compute the weight of External Input Transactional AFP
- weightExternalInterfaceFile: NamedMeasure
  To compute the weight of External Interface File Data AFP
- weightInternalLogicalFile: NamedMeasure
  To compute the weight of Internal Logical File Data AFP
- weightAutomatedFunctionPoints: CollectiveMeasure
  To compute an AFP size regardless of the AFP nature, by summing the results of the various AFP weights, knowing that only one result is not null by definition (weightExternalOutput: NamedMeasure, weightExternalInput: NamedMeasure, weightExternalInterfaceFile: NamedMeasure, weightInternalLogicalFile: NamedMeasure)

- For resulting AEFP values of Transaction AFP

(using scope AddedTransactionalAFP: Scope, that is, EO and EI added in the latest revision)

- weightAddedTransactionalFunctionPoints: BinaryMeasure
  To compute the final contribution of added transactional AFP into Automated Enhancement Function Point value, as the product of the associated complexity factor (AddedTransactionalAFPComplexityFacto: RescaledMeasure) by the AFP value (weightAutomatedFunctionPoints: CollectiveMeasure)

(using scope DeletedTransactionalAFP: Scope, that is, EO and EI removed from the latest revision)

- weightDeletedTransactionalFunctionPoints: BinaryMeasure
  To compute the final contribution of deleted transactional AFP into Automated Enhancement Function Point value, as the product of the associated complexity factor (DeletedTransactionalAFPComplexityFactor: RescaledMeasure) by the AFP value (weightAutomatedFunctionPoints: CollectiveMeasure)

(using scope UpdatedTransactionalAFP: Scope)

- weightUpdatedTransactionalFunctionPoints: BinaryMeasure
  To compute the final contribution of updated transactional AFP into Automated Enhancement Function Point value, as the product of the associated complexity factor (UpdatedTransactionalAFPComplexityFactor: RescaledMeasure) by the AFP value (weightAutomatedFunctionPoints: CollectiveMeasure)

- For resulting AEFP values of Transaction AFP

(using scope AddedDataAFP: Scope, that is, EIF and ILF added in the latest revision)

- weightAddedDataFunctionPoints: BinaryMeasure
  To compute the final contribution of added data AFP into Automated Enhancement Function Point value, as the product of the associated complexity factor (AddedDataAFPComplexityFactor: RescaledMeasure) by the AFP value (weightAutomatedFunctionPoints: CollectiveMeasure)

(using scope DeletedDataAFP: Scope, that is, EIF and ILF removed from the latest revision)

- weightDeletedDataFunctionPoints: BinaryMeasure
  To compute the final contribution of deleted data AFP into Automated Enhancement Function Point value, as the product of the associated complexity factor (DeletedDataAFPComplexityFactor: RescaledMeasure) by the AFP value (weightAutomatedFunctionPoints: CollectiveMeasure)

(using scope MergedDataAFP: Scope, that is, EIFs and ILFs from previous revision merged into single EIF and ILF in the latest revision)

- weightMergedDataFunctionPoints: BinaryMeasure
  To compute the final contribution of merged data AFP into Automated Enhancement Function Point value, as the product of the associated complexity factor (MergedDataAFPComplexityFactor: RescaledMeasure) by the AFP value (weightAutomatedFunctionPoints: CollectiveMeasure)

(using scope SplitDataAFP: Scope, that is, single EIF and ILF from previous revision split into EIFs and ILFs in the latest revision)

- weightSplitDataFunctionPoints: BinaryMeasure
  To compute the final contribution of added data AFP into Automated Enhancement Function Point value, as the product of the associated complexity factor (SplitDataAFPComplexityFactor: RescaledMeasure) by the AFP value (weightAutomatedFunctionPoints: CollectiveMeasure)

(using scope UpdatedDataAFP: Scope)

- weightUpdatedDataFunctionPoints: BinaryMeasure
  To compute the final contribution of updated data AFP into Automated Enhancement Function Point value, as the product of the associated complexity factor (UpdatedDataAFPComplexityFactor: RescaledMeasure) by the AFP value (weightAutomatedFunctionPoints: CollectiveMeasure)

## 6.5 Measurement Calculations for the Automated Enhancement Functional and Technical Points Scopes

The Automated Enhancement Function Points score for transactions and data entities within the Automated Enhancement Function Point Scope is the sum of their AFP values adjusted by the Complexity Factor:

$$AEFP = \Sigma (CF_{added/modified} * AFP_{ToRevition}) + \Sigma (CF_{deleted} * AFP_{FromRevision}) + \Sigma (CF_{split/merged/changed\ type} * AFP_{ToRevision})$$

Corresponding elements in the specification associated normative XMI file are:

(using scope LatestRevision: Scope)

- AutomatedEnhancementFunctionPoint: CollectiveMeasure
  To compute the total Automated Enhancement Function Point value by summing all contributions of evolved data and transactional AFP (weightAddedDataFunctionPoints: BinaryMeasure, weightAddedTransactionalFunctionPoints: BinaryMeasure, weightDeletedDataFunctionPoints: BinaryMeasure, weightDeletedTransactionalFunctionPoints: BinaryMeasure, weightMergedDataFunctionPoints: BinaryMeasure, weightSplitDataFunctionPoints: BinaryMeasure, weightDataWithChangedTypeFunctionPoints: BinaryMeasure, weightUpdatedDataFunctionPoints: BinaryMeasure, weightUpdatedTransactionalFunctionPoints: BinaryMeasure)

The Implementation Points associated with Automated Enhancement Technical Points are calculated as the sum of the Effort Complexities of all the Artifacts within the Automated Enhancement Technical Points Scope, that is:

$$\mathbf{IP_{AETP} = \Sigma\ (EC_{AETP})}$$

Corresponding elements in the specification associated normative XMI file are:

(using scope EvolvedATPArtifacts: Scope)

- ImplementationPoints_EvolvedATPArtifacts: CollectiveMeasure
  To compute the Implementation Point value of the evolved ATP Scope, by summing all the Artifact Effort Complexity values (ArtifactEffortComplexity: RescaledMeasure)

The Implementation Points associated with Automated Enhancement Technical Points will be multiplied by the Equivalence Ratio to transform them into a measure that is equivalent to the Automated Enhancement Function Points.

## 6.6 Measurement Calculations for Automated Function Points Scope

The Implementation Points associated with Automated Function Points are calculated as the sum of the Effort Complexities of all the Artifacts within the Automated Function Points Scope, that is:

$$\mathbf{IP_{AFP} = \Sigma\ (EC_{AFP})}$$

The **Equivalence Ratio** (EC) calculated by dividing the Automated Function Points score of an application by the Implementation Points score associated with all computational objects within the Automated Function Point Scope of the application.

$$\mathbf{ER = AFP\ /\ IP_{AFP}}$$

Corresponding elements in the specification associated normative XMI file are:

(using scope LatestRevision: Scope)

- AutomatedFunctionPoints_Latest: CollectiveMeasure
  To compute the total Automated Function Point value in the latest revision, by summing all the Automated Function Point contributions (weightAutomatedFunctionPoints: CollectiveMeasure)

(using scope LatestAFPRevision: Scope)

- ImplementationPoints_LatestAFPScope: CollectiveMeasure
  To compute the Implementation Point value of the AFP Scope in the latest revision, by summing all the Artifact Effort Complexity values (ArtifactEffortComplexity: RescaledMeasure)

(using scope LatestRevision: Scope)

- EquivalenceRatio_Latest: RatioMeasure
  To compute the Equivalence Ratio as the total Automated Function Point value (AutomatedFunctionPoints_Latest: CollectiveMeasure) divided by the total Implementation Point value of the AFP Scope (ImplementationPoints_LatestAFPScope: CollectiveMeasure) in the latest revision

## 6.7 Measurement Calculations for the whole Application Scope

The Equivalence Ratio is applied to the Implementation Points derived from the Automated Enhancement Technical Points score to create a unit of measure that is statistically equivalent to the Implementation Points measure derived from Automated Enhancement Function Points.

$$\mathbf{AETP = ER \times IP_{AETP}}$$

Corresponding elements in the specification associated normative XMI file are:

(using scope EvolvedATPArtifacts: Scope)

- AutomatedEnhancementTechnicalPoint: BinaryMeasure
  To compute the Automated Enhancement Technical Point value as the product of the Equivalence Ratio (EquivalenceRatio_Latest: RatioMeasure) by the Implementation Point value of evolved ATP Scope (ImplementationPoints_EvolvedATPArtifacts: CollectiveMeasure)

The Automated Enhancement Points score is calculated as the sum of Automated Enhancement Function Points and Automated Enhancement Technical Points scores:

$$\mathbf{AEP = AEFP + AETP}$$

Corresponding elements in the specification associated normative XMI file are:

(using scope LatestRevision: Scope)

- AutomatedEnhancementPoint: BinaryMeasure
  To compute the Automated Enhancement Point value as the sum of the AETP (AutomatedEnhancementTechnicalPoint: BinaryMeasure) and of the AEFP (AutomatedEnhancementFunctionPoint: CollectiveMeasure)

## 6.8 Output Generation

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

- What is the functional size of the software enhancement? What is the functional size of feature enhancements?
- What is the implementation size of the software enhancement? What is the implementation size of feature enhancements?
- Where are the software enhancement taking place?
- What are the assumptions used in the process?

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:

- Automated Enhancement Points (AEP) value
- Automated Enhancement Function Points (AEFP) and Automated Enhancement Technical Points (AETP) values
- Implementation Points values for Automated Enhancement Function Points and Automated Enhancement Technical Points scopes (IP(AEFP) and IP(AETP))
- Location of each Data Function and Transactional Function underlying elements in the source code (Data AFP Implementation Scopes and Transactional AFP Implementation Scopes),
  - with their evolution status (added, deleted, updated, unchanged),
  - with their sharing status,
  - and with their Artifact nature (Transactional AFP Implementation Artifacts)
- Effort Complexity of each Artifact, with the details of the underlying category assignments (Cyclomatic Complexity level, Size level, Comment level, SQL Complexity level and value, Coupling level)
- List of evolved Data AFP (added, deleted, modified, merged, split, with changed type) and Transactional AFP (added, deleted, modified), with their Complexity Factor
  - In case of variable value – that is, for each modified Data and Transactional AFP – the Complexity Factor input – that is, for each modified Data AFP, the percentage of changed DET and for each modified Transactional AFP, the percentage of Effort Complexity variation for all Artifacts in the Implementation Scope, the percentage of Effort

Complexity in evolved Artifacts, and the percentage of Effort Complexity in evolved shared Artifacts –

- A complete list of inputs used by to generate the outputs.

## 6.9 Outline of the Automated Enhancement Points Calculation Process

For both “FromRevision“ and “ToRevision“ revisions:

- Collect required input
- Generate the application model
- Compute AFP scopes and metrics
  - AFP detection in application model (according to formal/2013-01-02)
  - AFP complexity and sizing (according to OMG AFP)
  - AFP Implementation Scopes in application model (new: not required in OMG AFP) (leading implicitly to computation of the ATP scope, complementary to the AFP scope in the whole software)
  - Shared AFP Elements scope
- Compute Implementation scopes and metrics
  - Artifacts EC in application model
  - $IP_{AFP}$
  - $IP_{ATP}$

For “ToRevision“ revision, using “FromRevision“ revision as its previous revision:

- Generate Artifacts evolution scopes
  - Added, Updated, and Deleted Artifacts between application models
  - Added, Updated, and Deleted AFP Artifacts
  - Added, Updated, and Deleted Shared ATP Artifacts
- Compute evolved implementation scopes and metrics
  - $Scope_{AEFP}$
  - $Scope_{AETP}$
- Identify evolved AFP
  - Added, Updated, and Deleted Transactional AFP
  - Added, Updated, Deleted, Merged, Split, ‘with changed type’ Data AFP
- Compute complexity factor for evolved AFP
  - $EC_{variation}$, $EC_{evolved}$, $EC_{shared}$
- Compute resulting AEFP
- Compute AETP
  - EC for all Artifacts in $Scope_{AETP}$
- Compute functional equivalent form of $IP_{AEFP}$
  - ER from AFP and $IP_{AFP}$
  - $AETP_{EQ}$ from ER and $IP_{AETP}$
- Compute AEP from AEFP and $AETP_{EQ}$

# 7. Usage Scenarios (Informative)

The usage scenarios following the applicable use cases are the following

- Gain visibility into the amount of enhanced or changed functional features between selected revisions of an application
- Gain visibility into the implementation complexity of the enhanced or changed functional features between selected revisions of an application
- Gain visibility into the implementation complexity of the enhanced or changed technical foundation software between selected revisions of an application
- Get the functional equivalent amount of enhanced or changed technical foundation software between selected revisions of an application
- Get the productivity of maintenance and enhancement work between selected revisions of an application
- Get the functional productivity between selected revisions of an application

## 7.1 Delivered Amount of Software Features Enhancement

To get a measurement of the amount of enhanced or changed functional features (those traditionally measured with Automated Function Points between two software revisions, use the Automated Enhancement Function Points value which:

- focuses only on the enhancements that impact directly the functional features traditionally measured with by Automated Function Points, overlooking the enhancements that impact the software technical foundation, and
- is expressed in an Enhancement Function Point units, which are aligned with the Automated Function Points unit in terms of concept and magnitude.

This will aid consumption of the results by business-oriented audiences, whose focus is on functional feature evolutions only, and not of the required amount of work.

With Automated Enhancement Function Points, managers can monitor and benchmark how maintenance and enhancement projects impact the functional features of an application. This measure enables them to define a measure for functional productivity for maintenance and enhancement work as described in clause 8.2.

## 7.2 Overall Functional Enhancement Productivity

To get a measurement for the productivity of implementing functional enhancements, simply divide the Automated Enhancement Function Points value by the Effort expended on the work. Note that in this scenario:

- the Effort value has to consider the same scope of work as is measured by the Automated Enhancement Function Points Scope,
- this productivity measure only relates to functional maintenance and enhancement work, and does not account for maintenance and enhancements on the technical foundation software.

## 7.3 Implementation Complexity of Enhancing Software Features

To get a measurement of the implementation complexity for maintenance and enhancement work on functional features between two software revisions, use of Implementation Points calculated on the Automated Enhancement Function Points value which:

- focuses only on the enhancements that impact directly the functional features traditionally measured with Automated Function Points, overlooking changes that impact the software technical foundation,
- but takes into account the complexity of implementing the changes to account for the amount of work required to enhance the functional features.

With Automated Enhancement Function Points and their Implementation Points value, managers can monitor and benchmark the impact of maintenance and enhancement projects on the functional features, taking into account the required Effort. This enables an analysis of Return on Investment for functional features, but these analyses will not reflect the value of improving the technical foundation of the application.

## 7.4 Implementation Complexity of Enhancing Technical Foundation Software

To measure the implementation complexity of maintenance and enhancement work on the technical foundation software between two software revisions, use Implementation Points calculated on the Automated Enhancement Technical Points value which:

- focuses only on the enhancements that impact directly the technical foundation software, overlooking the enhancements that impact the functional features,
- takes into account the complexity of implementing these changes to account for the amount of work required to enhance the technical foundation software.

## 7.5 Functional vs. Technical Investment Ratio

This ratio comparing the amount of change in functional software to that in technical foundation software helps educate the business on where, how, and why development investment dollars/Euros/etc. are spent. With Automated Enhancement Function Points and the Implementation Points derived from both the Automated Enhancement Function Points value and the Automated Enhancement Technical Points value, managers can monitor and benchmark not only how maintenance and enhancement projects impact the software features, but also how much change this requires in the technical foundation software as well. However, this ratio does not provide a direct measure of the amount of technical work that can be easily understood by the business-oriented audiences.

The formula for measuring the share of maintenance and enhancement activities that deal with software features is based in Implementation Points and is computed as $IP_{AEFP}$ / ($IP_{AETP}$ + $IP_{AEFP}$). Similarly, the formula for measuring the share of maintenance and enhancement activities that deal with technical foundation software is computed as $IP_{AETP}$ / ($IP_{AETP}$ + $IP_{AEFP}$). This ratio is an objective measure that helps explain how much work is required on the technical foundation software (as deemed necessary by the

software architects and development teams) to support the functional evolutions requested by the lines of business.

## 7.6 Functional Equivalent Amount of Technical Foundation Software Enhancement

To get a measurement of the technical foundation software enhancements between two software revisions that is statistically equivalent Automated Function Points, use of Automated Enhancement Technical Points value adjusted by the Equivalence Ration which:

- focuses only on the enhancements that impact exclusively the technical foundation software, overlooking the enhancements that impact the functional features,
- Is expressed in a $AFP_{eq}$ units, which is statistically equivalent to Automated Function Point units in terms of magnitude,

This measure helps ease the consumption of the results by business-oriented audiences, who focus on visible feature evolutions only (and not of the technical work on the technical foundation software that does not translate into enhanced software features).

## 7.7 Overall Enhancement Implementation Productivity

Automated Enhancement Function Points, Automated Enhancement Technical Points, and the Implementation Point measures based on each or them, managers can monitor and benchmark all aspects of the maintenance and enhancement work performed on an application, using raw or AFP-equivalent units to adapt to the different audiences. These measures also support the calculation of Automated Enhancement Points which enables a productivity indicator for maintenance and enhancement work defined as Automated Enhancement Points / Effort. In this scenario, it is key to note that:

- The Effort value has to match the same scopes as Automated Enhancement Function Points Scope, Automated Enhancement Technical Points Scope,
- This is overall maintenance and enhancement productivity, which accounts for work on both the functional features of an application and the technical foundation software that supports them. This provides a better accounting of the work produced by the total effort spent, and is therefore a superior productivity measure for maintenance and development work.

# Appendix A: Artifact Data Types

### SQL Languages

- Function
- Procedure
- Package function
- Package procedure
- Trigger
- View

### C/C++ Language

- C/C++ Function
- C/C++ Method
- C/C++ Constructor
- C/C++ Destructor

Notes:

- C/C++ Macros are not counted as Artifacts as they are most of the time a single line of code shortcut.
- Only C++ objects with a code definition are considered to be artifacts
- C++ Methods/Function declared in a .h but not implemented in a C++ file are not considered to be artifacts

### Visual Basic Language

- VB Event
- VB Function
- VB Property Get
- VB Property Let
- VB Property Set
- VB Sub

### Java Language

- Java Constructor
- Java Method
- Java Initializer

Notes:

- The automatically generated Java Methods like '_jspService' are not considered as Artifacts.

- The Java Classes that belong to standard libraries and custom libraries (i.e. their source code is not available) are not considered as Artifacts.

### Mainframe

- Cobol Program
- JCL Job
- JCL Procedure
- IMS Segment
- IMS DB PCB

### MS.NET Languages

- Method
- Property Set
- Property Get
- AddOn
- RemoveOn
- Fire
- Constructor
- Destructor
- Event
- eFunction
- eSub
- ePropertyGet
- ePropertySet
- ePropertyLet
- eEvent
- eFile

### Web Languages (JSP/ASP/JS)

- Method
- eFunction
- eSub
- eMethod
- ePropertyGet
- ePropertySet
- ePropertyLet
- eEvent
- Java Method
- eFile

Notes:

- eFiles are included in the count only when they contain executable source code (file extensions like *.jsp, *.asp, *.js), excluding from the list HTML files and other images, configuration files …

### SAP ABAP

- ABAP Form
- ABAP Function
- ABAP Event Block
- ABAP Module
- ABAP Method
- ABAP Constructor
- ABAP Event Method
- ABAP File Level Code of custom programs, user-exits and includes
- ABAP Event
- WebDynpro Supply Function
- WebDynpro Event Handler
- WebDynpro Method