**Date:** March 2009

# Architecture-Driven Modernization (ADM): Software Metrics Meta-Model (SMM)

## FTF - Beta 1

---

**OMG Document Number: ptc/2009-03-03**

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

**Associated File(s)*: http://www.omg.org/spec/SMM/20080501**

**http://www.omg.org/spec/SMM/20080601**

---

* Original file(s): XMI (admtf/08-05-05), XSD (admtf/08-06-05)

This OMG document replaces the submission document (admtf/08-05-04, Alpha). It is an OMG Adopted Beta Specification and is currently in the finalization phase. Comments on the content of this document are welcome, and should be directed to *issues@omg.org* by April 1, 2009.

You may view the pending issues for this specification from the OMG revision issues web page *http://www.omg.org/issues/*.

The FTF Recommendation and Report for this specification will be published on July 1, 2009. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

## USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group 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 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

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

## DISCLAIMER OF WARRANTY

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

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

## RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c) (1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

## TRADEMARKS

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/technology/agreement.)

# Table of Contents

# Preface

### OMG

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

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

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

# OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A Specifications Catalog is available from the OMG website at:

*http://www.omg.org/technology/documents/spec_catalog.htm*

Specifications within the Catalog are organized by the following categories:

## OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

## OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM)

## Platform Specific Model and Interface Specifications

- CORBAservices
- CORBAfacilities
- OMG Domain specifications

- OMG Embedded Intelligence specifications
- OMG Security 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
140 Kendrick Street
Building A, 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 standards. Please consult *http://www.iso.org*

# Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

**Helvetica/Arial - 10 pt. Bold:** OMG Interface Definition Language (OMG IDL) and syntax elements.

**`Courier - 10 pt. Bold:`** Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

# 1 Scope

This specification defines a meta-model for representing measurement information related to software, its operation and its design. Referred to as the Software Metrics Meta-model (SMM), this specifaction is an extensible meta-model for exchanging software-related measurement information concerning existing software assets (designs, implementations, or operations). A standard for the exchange of measures is important given the role that measures play in software engineering and design.

The SMM include elements representing the concepts needed to express a wide range of software measures. The specification does include a library of software measures, but it is not asserting that the listed measures constitute standards themselves. Software measurement field is fairly young especially with respect to modernization. As the field matures, the measures considered as standard are likely to change.

The SMM is instead a specification for the definition of measures and the representation of their measurement results. The library serves to demonstrate the specification and provide a representation for many currently popular software measures.

The SMM is part of the Architecture Driven Modernization (ADM) roadmap and fullfills the metric needs of the ADM roadmap scenarios and as well as other information technology scenarios.

The SMM specifies the representation of measures without detailing the representation of the entities measured. SMM anticipates that those entities are represented in other OMG meta-models. Measures of software artifacts or their features that are defined within the SMM, the Knowledge Discovery Metamodel (KDM), the Abstract Syntax Tree Metamodel (ASTM), another ADM roadmap meta-model, or another OMG meta-model may arise as:

- Counts. (Lines of code measures exemplify the mechanism.)
- Direct applications of named measurements. (One such named measure is Cyclomatic Complexity.)
- Simple algebraic change of scales of already defined numeric measures (e.g., the translation to ‘choice points’ from Cyclomatic complexity).
- Simple algebraic aggregations of numeric artifact features, including other measures, over sets of software artifacts. (Determining the complexity of an application by summing the complexities of the application’s elements demonstrates this process.)
- Simple range-based grading or classification of already defined numeric measures. (Cyclomatic reliable/unreliable quadrants are one such a grading.)
- Qualitative evaluations where the range of evaluations can be mapped to a linear order.

Useful metrics must go beyond static (or dynamic) code analysis and technical performance to include factors related to information utility and acceptance of the system by the organization(s) participating in an enterprise. To be objective and repeatable, such metrics need to be based on technical characteristics of the system. Given a meta-model representation of such characteristics, the SMM will facilitate the exchange of such measures.

Given the evolutionary nature of system development and the predicate value of metrics with respect to “downstream” problems, metrics are gathered into trends or viewed from historical perspective. As shown in Section 14 , SMM addresses the issues of trend and history to model for system development as long as the historical links of the measured entities are provided.

Consistent with other models defined by OMG, the SMM will be defined using the MOF meta-modeling language. As such, it will have a standard textual representation presented by XMI. Consequently, the exchange of metrics defined by SMM will be in the XMI. These models will, similarly, be compatible with MOF repositories for storage and retrieval by various tools.

# 2 Conformance

The principle goal of SMM is the exchange of measurements about software. To be SMM compliant, a tool must completely support SMM model elements. An implementation can provide:

- The capability to generate XMI documents based on the SMM XMI schema capturing measurements from the existing model of the tool.
- The capability to import measurements via representations based on the SMM XMI schema and to map the measurements into the existing model of the tool.

# 3 Normative References

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

- UML 2. Infrastructure Specification
- MOF 2.0 Specification

# 4 Terms and Definitions

We assume the following definitions:

**Measure**: A method assigning comparable numerical or symbolic values to entities in order to characterize an attribute of the entities.

**Measurement**: A numerical or symbolic value assigned to an entity by a measure.

**Measurand**: An entity quantified by a measurement.

**Unit of Measure:** A quantity in terms of which the magnitudes of other quantities within the same total order can be stated.

**Dimension**: A totally ordered range of values which can be stated as orders of magnitude relative to one another or to an archetypal member.

**Measurement Accuracy:** The measurement by which another measurement may be wrong.

**Measurement Scope**: The domain (set of entities) to which a given measure may be applied.

**Measurement Range**: The range (set of comparable values) assignable by a given measure.

# 5 Symbols

There are no symbols/abbreviations.

# 6 Additional Information

## 6.1 Changes to Adopted OMG Specifications

There are no changes to other OMG specifications.

## 6.2 How to Read this Specification

The rest of this document contains the technical content of this specification.

Although the chapters are organized in a logical manner and can be read sequentially, this reference specification is intended to be read in a non-sequential manner. Consequently, extensive cross-references are provided to facilitate browsing and search.

## 6.3 Acknowledgements

The following companies submitted and/or supported parts of this specification:

- EDS
- KDM Analytics
- Software Revolution
- Tactical Strategy Group
- NIST
- Benchmark Consulting
- eCube Systems

The following persons were members of the core team that designed and wrote this specification: Kevin Barnes, Djenana Campara, Larry Hines, Nikolai Mansurov, Alain Picard, John Salasin, Michael Smith, and William Ulrich.

# 7 SMM

Measurements provide data for disciplined software engineering in that engineers and their managers rely on these comparable evaluations in assessing the static and operational qualities of software systems.

Software measurement methods produce comparable evaluations of software or application artifacts. Counts such as number of screens, lines of code, and number of methods quantify the size of artifacts along a single dimension. These evaluations readily distinguish larger artifacts from smaller ones, likewise complexity metrics such as Halstead and Cyclomatic separate the simpler artifacts from the more complex. Comparable evaluations form mappings of artifacts of a given type into a single dimension.

Such is also the case for architecture measures (coupling and cohesion); functional measures (functions defined in system, persistent data as a percentage of all data, functions in current system that map to functions in target architecture); quality measures (failures per unit time, meantime to failure, meantime between repair); performance measures (average batch window clock time, average online response time); software assurance measures; and cost measures.

Predictive metrics provide a basis for continual system-level in contrast to fixed milestone-based assessments. These metrics may indicate at some future development stage the probability that the system will or will not meet its requirements.

This specification defines a meta-model for representing measurement related to existing software assets and their operational environments referred to as the Software Metrics Meta-model (SMM).

The SMM promotes a common interchange format that will allow interoperability between existing modernization tools, commercial services providers and their respective models. This common interchange format applies equally well to development and maintainence tools, services and models. SMM complements a common repository structure and so facilitates the exchange of data currently contained within individual tool models that represent existing software assets. Given that the repository's meta-model represents the physical and logical software assets at various levels of abstraction as entities and relations, SMM represent the measurements of these assets.

The main goal for the SMM is to provide an extendable meta-model establishing a standard for the interchange of software-related measurements over the entities modeled by ADM Roadmap[1] meta-models or other OMG meta-models. By software-related, we mean measurements derived from the existing software artifacts (including source, design and linkage from source to target architectures) or technical measurements concerning deployment. Source artifacts include program code, runtime traces, scheduling specifications, screen layouts and UML models. It may also include grid-service infrastructure descriptions and SOA adoption specification of multiple organization units in an enterprise.

SMM contains meta-model classes and associations to model measurements, measures and observations. We present and explain diagrams depicting measures, then measurements and finally observations.

SMM supports the meta-models of the ADM roadmap by providing quantifiable and specific indicators, in the form of counts, measures and computational results, about existing systems and the relationship of those systems to target architecture. The meta-model provides for and is extendable to measurements of entities modeled by other OMG meta-models where those measurements are software-related.

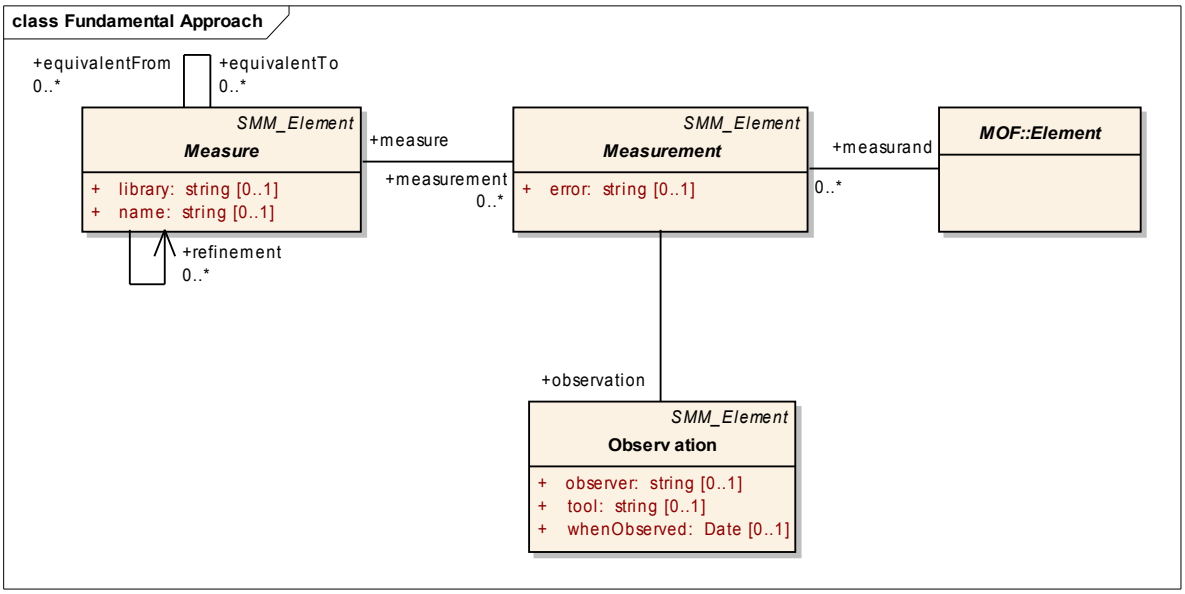


**Figure 1 - Fundamental Approach**

SMM avoids duplicating features of the measured artifact as features of the measurement. Consider as an example a log of bug reports. Possible measures are total bug count in the log, total time logged in the log and bugs per time-period. The unit of measures are a bug, a unit of time and bugs per time interval, respectively. SMM does not provide representations for bug, start time and end time. Their representations must be provided elsewhere[2].

[1] See OMG document **admtf/04-09-02: Architecture-Driven Modernization Roadmap.**

[2] For example, the General Ledger Specification v1.0 provides representations for start_date and end_date.

A measurement result is precisely identified only if its measure is identified. To understand the meaning of 1000 lines we need to know that it is the result of measuring a program's length in lines. The measured entity must be identified. That is, 1000 lines is for a particular program. Contextual information may also be needed. For example, function point counts of a program may vary depending upon the expert applying the measure.

Figure 1 presents the fundamental approach of this specification. Measurement has a value conveying the measurement results. The measurement may be of any MOF element as related by the measurand association. In this way, measurement is applicable to elements of other OMG meta-models including the Knowledge Discovery Meta-model and the Abstract Syntax Tree Meta-model. The measured entity may represent any software artifact or an aspect of an artifact.

The SMM associates an evaluation process, a measure, to each the measurement. Measures signify functions from the domain of software artifacts and aspects thereof to sets of ordered values.

Contextual information is related by Observation, such as who, where, and when. Observation may serve to distinguish distinct utilizations of a given measure on a given measurand.

# 8 Core Classes

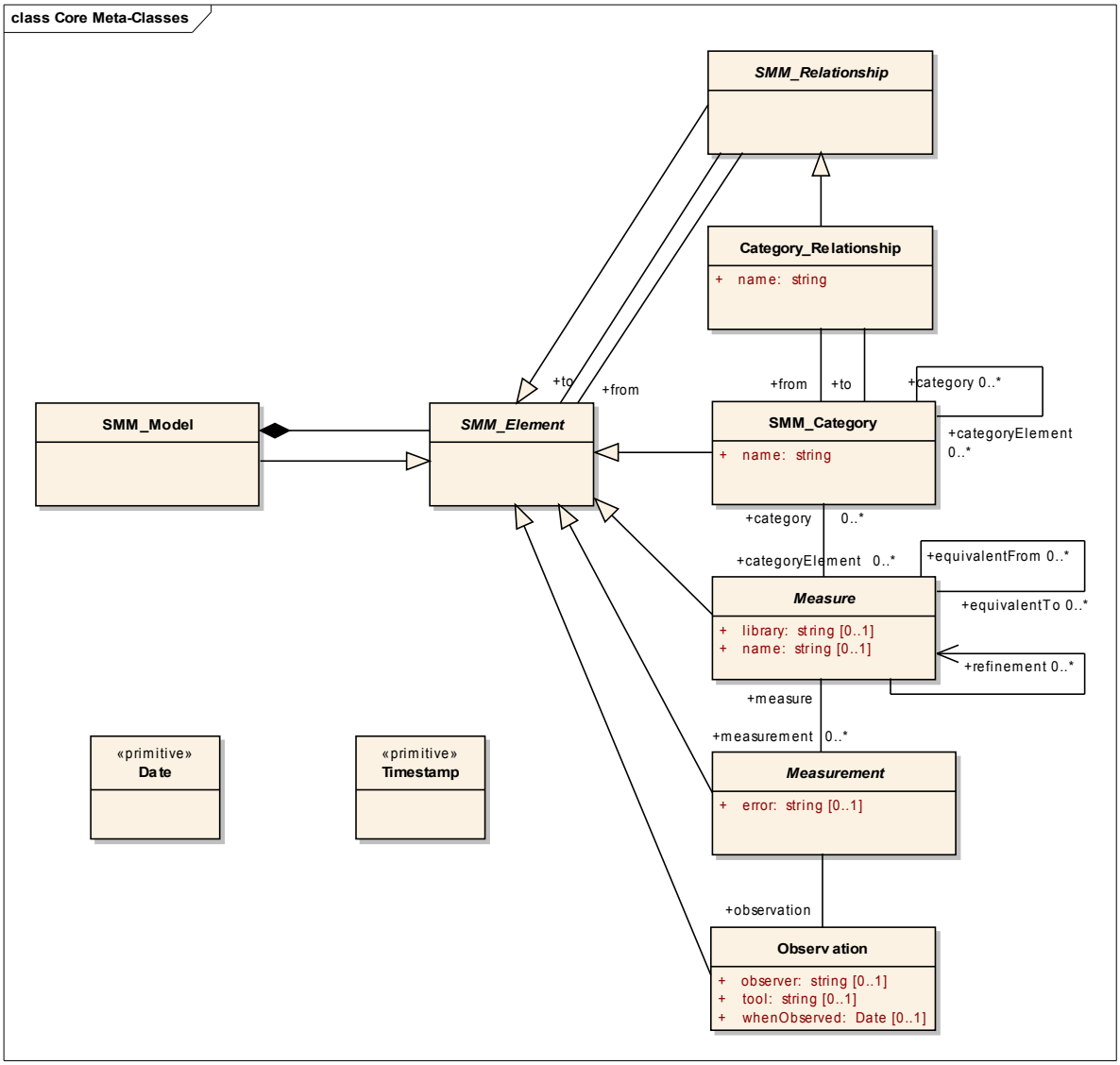


**Figure 2 - Core Classes Diagram**

## 8.1 SMM_Element Class (Abstract)

An SMM element constitutes an atomic constituent of a model. In the meta-model, SMM_Element is the top class in the hierarchy. SMM_Element is an abstract class.

### Attributes

| | |
|---|---|
| name: String | Specifies the name of the SMM element. |
| short_description: String | A short description for the element (optional). |
| description: String | A detailed description for the element (optional). |

## 8.2 SMM_Model Class

This class represents the aggregation of all the elements of the SMM.

## 8.3 SMM_Relationship (abstract)

This class is a model element that represents semantic association between SMM elements.

#### SuperClass

SMM_Element

#### Associations

| | |
|---|---|
| from:SMM_Element | The *origin* element (also referred to as the from-endpoint of the relationship). |
| to:SMM_Element | The *target* element (also referred to as the to-endpoint of the relationship). |

## 8.4 SMM_Category Class

This class represents categories of measures. A category has measures and other categories as its elements.

A category representsthe measures directly associated with an 'element' and the measures of each sub-category likewise associated withan 'element.'

A measure may appear in multiple categories. A category can be a subcategory of other categories indicating only that its measures also are measures of these other categories.

This class may be used to represent a family of similar measures which apply to different scopes such as lines of code in a file, lines of code in a method and lines of code in program. It may also represent a category of measures which are associated with a given software field or engineering task. For instance we speak often of Quality Assurance Metrics and Software Maintainability Metrics. The category of a metric may indicate the kind of purpose for which the metric is used.

- Environmental Metrics (e.g., number of screens, programs, lines of code, etc.)
- Data Definition Metrics (e.g., number of data groups, overlapping data groups, unused data elements, etc.)
- Program Process Metrics (e.g., Halstead, McCabe, etc.)
- Architecture Metrics (e.g., average call nesting level, deepest call nesting level, etc.)
- Functional Metrics (e.g., functions defined in system, business data as a percentage of all data, functions in current system that map to functions in target architecture, etc.)
- Quality Metrics (e.g., failures per day, meantime to failure, meantime to repair, etc.)
- Performance Metrics (e.g., average batch window clock time, average online response time, etc.)
- Software Assurance Metrics

Metric categorization has other uses as well. For example, measures may be categorized by tool support.

#### Associations

| | |
|---|---|
| categoryElement:Category | Indicates that categoryElement is a subcategory of this category. |
| categoryElement:Measure | Indicates that measure is in this category. |
| parameter:Category_Relationship[0..*] | Associates parameters or features of the category. |

## 8.5 Category_Relationship

This class is a model element that represents semantic or named association between SMM catogories and other SMM elements. For example, a modeler may choose to create a "gold standard" measure for a selected category. To do so, the modeler can use a category relationship named "gold standard" to associate the measure to the category. See Figure 15.

#### SuperClass

SMM_Element

#### Associations

| | |
|---|---|
| from:SMM_Category | Indicates the category which has relation. |
| to:SMM_Element | Indicates the SMM element related to the category. |

#### Semantics

Category_Relationship represents a named association between a category and an element (SMM_Element) such as a measure.

## 8.6 Date

This represents dates. In a language binding it should be mapped to a type that allows ordered comparison. For XMI it is mapped to the XML Schema **date** type.

## 8.7 Timestamp

This represents a point in time: for example, a combination of a date and a time within the day. For XMI it is mapped to the XML **dateTime** type.

# 9 Measures

Measures are evaluation processes that assign comparable numeric or symbolic values to entities in order to characterize selected qualities or traits of the entities. Counting the lines of program code in a software application is one such evaluation.

There may be many measures that characterize a trait with differing dimensions, resolutions, accuracy, and so forth. Moreover, trait or characteristic may be generalize or specialized. For example, line length is a specialization of length which is a specialization of size.

Each measure has a scope, the set of entities to which it is applicable; a range, the set of possible measurement results; and the measurable property or trait which the measure characterizes. For example, the aforementioned line counting has software applications as one of its scope with line length as one of its measurable trait. Explicitly representing the scope and the measurable trait allows for the consideration of different measures which characterize the same attribute for the same set of entities. Each measurable trait may have multiple, identifiably distinct measures.

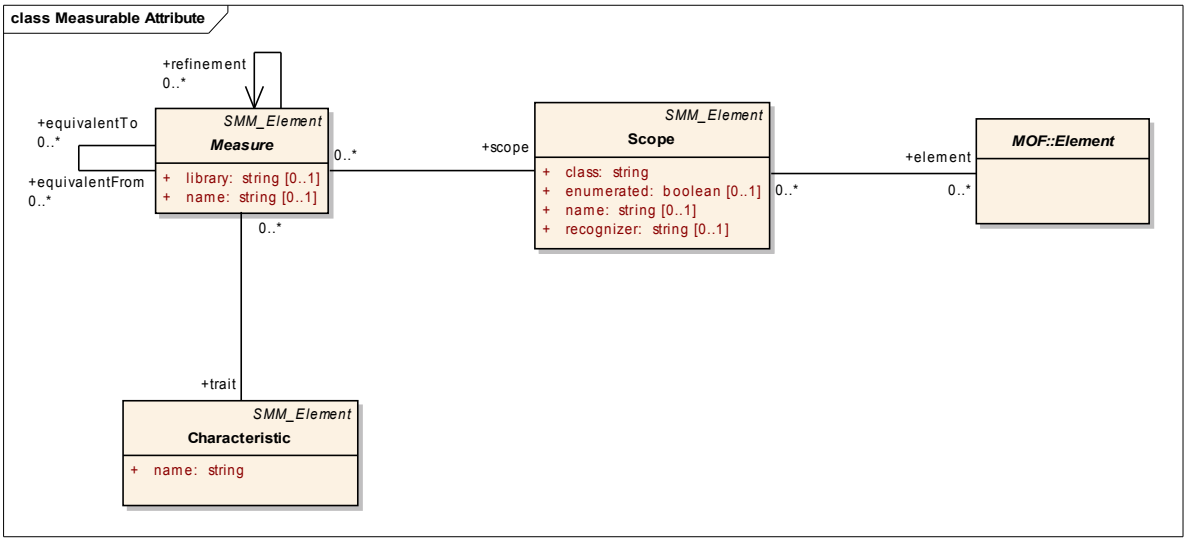


**Figure 3 Measurable Characteristic and Scope**

The evaluation process may assign numeric values which can be ordered by magnitude relative to one another. These measures are modeled by the DimensionalMeasure class.

The evaluation process may alternatively assign numeric values which are percentages or, more generically, ratios of two base measurements. These measures are modeled by the Ratio class. The percentage of comment lines in an application exemplifies this type of measure.

The evaluation process may also assign symbolic values demonstrating a ranking which preserve the ordering of underlying base measures. These measures are modeled by the Ranking class. Cyclomatic reliable/unreliable criterion illustrates one such ranking. Reliable is comparably better than unreliable. Comparability is essential here because ranking is not intended to model every possible assignment of measurands.

The documentations of measures should stand by themselves so that an interchange of measurements may simply reference such documentation and not duplicate it.

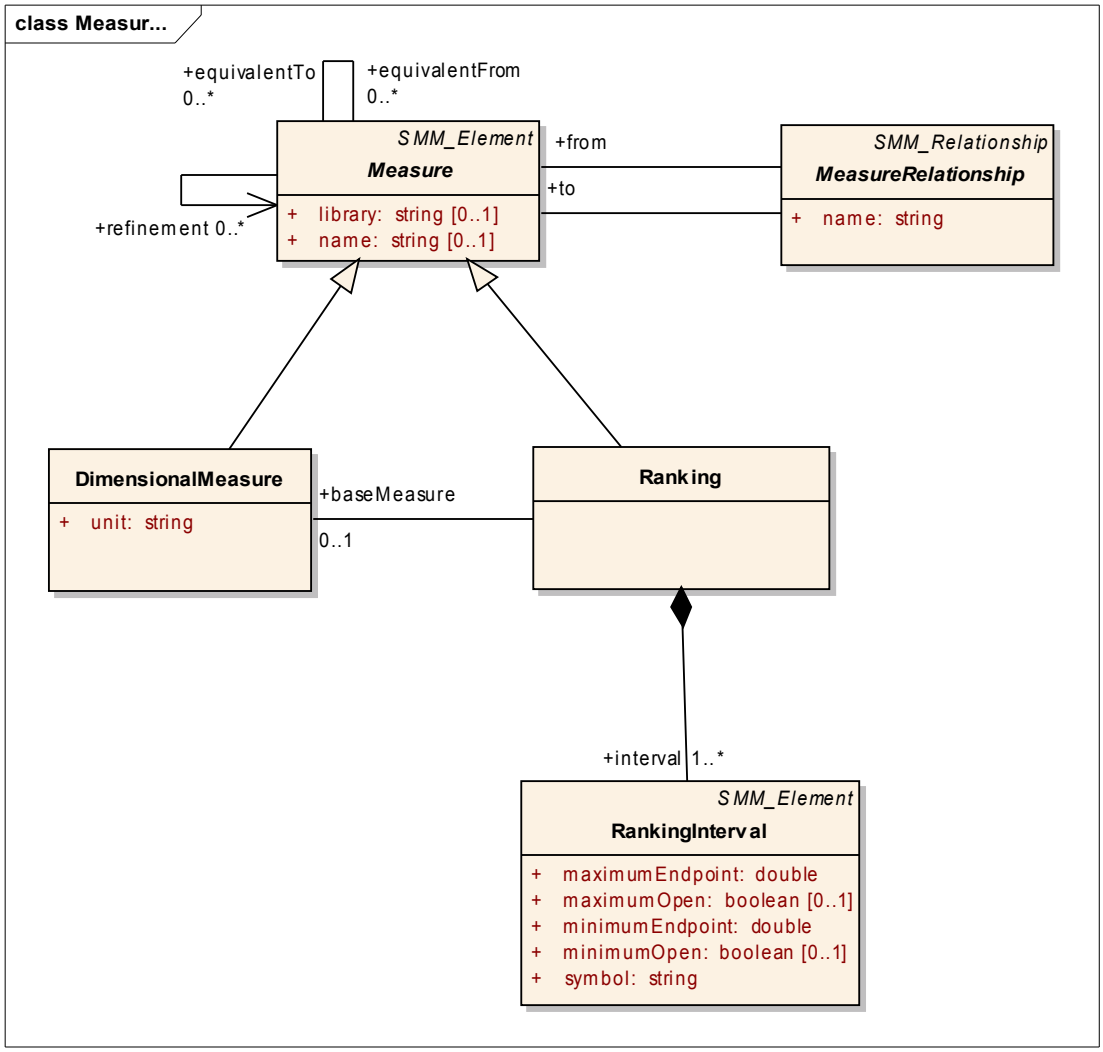


**Figure 4 - Measure Class Diagram**

## 9.1 Characteristic Class

This class represents a property or trait of the members in its scope, a set of MOF Elements, which may be characterized by applying a measure to those members. By specifying a characteristic a modeler is indicating what aspect, trait or property the measure purports to measure.

Note that Characteristic provides for a representaion of a hierarchy of measures based upon the abstraction of measured trait. For example, a length characteristic may be the parent of the fileLength and programLength characteristics. programLength could be the parent of programLinesOfCodeLength.

### SuperClass

SMM_Element

### Attributes

| | |
|---|---|
| name: String | Specifies the name of the SMM element. (inherited) |

### Associations

| | |
|---|---|
| parent:Characteristic[0..1] | Specifies the generalization of this characterization. |

## 9.2 Scope Class

This class represents sets of MOF::Elements as domains for measures. The domain is a subset instances of a class specified by the class attribute. If the subset does not include all instances of the given class then a restriction is specified either by enumerating the element or by specifying a recognizer for the subset elements.

The scope of a measure identifies a set of objects as the domain of the measure. The object all exhibit to varying degrees the trait or property characterized by a measurement. SMM requires that the objects be instances of a single class. The set of objects may be further restricted by a recognizer function or by enumerating them explicitly. The recognizer and the enumeration are optional, but they cannot be used together.

The recognizer, if given, is a boolean function applicable to instances of the named class. The measure's scope is restricted to those instances for which the recognizer returns true. Alternatively if enumeration is set then the scope is the set of instances (of the named class) associated as elements to the Scope.

### SuperClass

SMM_Element

### Attributes

| | |
|---|---|
| class: String | Specifies the class for elements of the set. |
| name: String | Specifies the name of this entity set. (inherited) |
| enumerated: Boolean[0..1] | If given and true, indicates that the elements of the set are enumerated by the element association. |
| recognizer:String[0..1] | If given, provides a boolean operation applicable to instances of the class which returns true if and only if the instance is an element of the set. |

### Associations

| | |
|---|---|
| element:MOF::Element[0..*] | Specifies the elements of the set. Elements are specified if and only if enumerated is true. |

### Semantics

The class attribute may name a class within any OMG standard. The entities associated as elements of an Scope are restricted to members of specifies class.

## 9.3 Measure Class (abstract)

This class (see Figure 1) models the specification of measures either by name, by representing derivations of base measures, or by representing method operations directly applied to the measured object. The essential requirement for the measure class is that it meaningfully identifies the measure applied to produce a given measurement. For example, McCabe's cyclomatic complexity could be specified by its name, McCabe's cyclomatic complexity, by a direct measurement operation or by rescaling counts of either independent paths or choice points. A measure may alternatively be identified by citing a library of measure which includes the measure by name.

The scope of a measure identifies a set of objects as the domain of the measure. The objects all exhibit to varying degrees the trait or property characterized by a measurement. SMM requires that the objects be instances of a single class. The set of objects may be further restricted by a recognizer function or by enumerating them explicitly. The recognizer and the enumeration are optional, but they cannot be used together.

Scope need not be specified if the library and name are given. In that case, the scope can be found in the library.

A measure may be a refinement of another measure. The scope of the first measure is a subset of the second measure's scope. The characteristic of both measures must be identical.

### SuperClass

SMM_Element

### Attributes

| | |
|---|---|
| name: String[0..1] | Specifies the unique name of the measure. (inherited) |
| library: String[0..1] | Specifies library declaring measure. |

### Associations

| | |
|---|---|
| equivalentFrom:Measure[0..*] | Indicates that two measures are equivalent. |
| equivalentTo:Measure[0..*] | Indicates that two measures are equivalent. |
| scope:Scope | Specifies a set of elements measurable by this measure. |
| refinement:Measure[0..*] | Specifies measures whose scopes are subclasses of this measure's scope. |
| category:SMM_Category[0..*] | Specifies categories to which this measure belongs. |
| measurement: Measurement[0..*] | Indicates measurements obtained by this measure. |
| trait:Characteristic | Specifies the trait characterized by this measure. |

### Constraint

```
context Measure inv:
not library->isEmpty implies not name->isEmpty and
scope->isEmpty implies not library->isEmpty.
```

### Semantics

Assigning a measure to the equivalentTo role of another measure states that two measures are semantically indistinguishable. Any measurement result by one on a given entity under a given observation should equal a measurement by the other on the same entity and observation. The semantics of this association is symmetric, but only one direction needs to be given.

Throughout the remainder of this document we will say that a measure is a refinement of another measure if and only if the first is associated to the second as a refinement directly or transitively. This association implies that the class of the scope of a measure is a superclass of the class of the scope of any refinement measure. For any measure `m` and for any class `c` equal to `m.refinement.scope.class`, `c` is the class or is a subclass of the class `m.scope.class`.

The refinement association essentially establishes measures as methods of their scope's classes.

## 9.4 MeasureRelationship (abstract)

MeasureRelationship is an abstract class representing any relationship between two measurements. See Figure 4. The class provides as an extension point.

#### SuperClass

SMM_Relationship

#### Attributes

| | |
|---|---|
| name:String | Specifies the name of this measure relationship. (inherited) |

#### Associations

| | |
|---|---|
| from:Measure | Specifies the measure at the from endpoint of the relationship. |
| to:Measure | Specifies the measure at the to-endpoint of the relationship. |

## 9.5 DimensionalMeasure Class

This class models the specification of measures which assign numeric values that can be placed in order by magnitude. Dimensional measures have units of measures and their values span a dimension. See Figure 4.

The unit of measure is an archetypal or prototype element of the dimension. Every element of the dimension can be stated by a numerical multiple of the 'unit of measure' element.

The unit of measure does not distinguish between measures which share the same range. That distinction would be entirely within the purview of the measure identification. For examples, a height measure and a width measure may share the same unit of measure. That is to say, a measurement is not just a number and a unit of measure. The measured artifact must be indicated, the measure identified and contextual information retained as the observation.

#### SuperClass

Measure

#### Attributes

| | |
|---|---|
| unit:String | Identifies the unit of measure. |

## 9.6 Ranking Class

This class represents simple range-based gradings or classifications based upon already defined dimensional measures. See Figure 4.

Examples are:

- Small, medium, large
- Cold, warm, hot
- A, B, C, D or F
- Reliable / Unreliable

Collectively the ranking intervals may completely cover the base dimension or may leave gaps. A base measurement in such a gap is considered unranked and is not representable as a measurement of the ranking measure.

The intervals may overlap. A ranking resulting in a particular symbol means and only means that the base measure resulted in a value occurring a ranking's interval which mapped to that symbol. This does not exclude the possibility that the value might occur in another interval.

Ranking consists of mapping intervals to symbols where the intervals are parts of the underlying measure's dimension. For example, 100 to 90 points maps to "A," 80 up to 90 maps to "B," 70 up to 80 maps to "C," 60 up to 70 maps to "D," and below 60 maps to "F." The underlying dimension consists of grade points. The result is the usual A,B,C,D, and F style grade.

Ranking measure may represent a purely qualitative evaluation with no quantitative base measure. For example we could measure the non-standardness of the source language and evaluate it without quantification. It is identified as "2GL," "Unacceptable 3GL or 4GL," "Acceptable 3GL or 4GL," or "Ideal Strategic Language." The first two are judged equivalently non-standard. The third is more nearly standard and the last is standard.

#### SuperClass

Measure

#### Associations

| | |
|---|---|
| baseMeasure:DimensionalMeasure[0..1] | Identifies the base measure on which this ranking measure is based. |
| interval:RankingInterval[1..*] | Identifies intervals within the dimension of the base measure and the symbol to which each interval is mapped. |

## 9.7 RankingInterval Class

This class represents the mapping of an interval to a symbol which serves as a rank. The booleans, maximuOpen and minimumOpen, default to false. See Figure 4.

#### SuperClass

SMM_Element

### Attributes

| | |
|---|---|
| maximumOpen:Boolean | True if and only if interval include maximum endpoint. |
| minimumOpen:Boolean | True if and only if interval include minimum endpoint. |
| maximum:Number | Identifies interval's maximum endpoint. |
| minimum:Number | Identifies interval's minimum endpoint. |
| symbol:String | Base measurements within this interval are mapped by symbol. |

### Constraints

```
context RankingInterval inv:
maximum ≥ minimum and (maximumOpen or minimumOpen → maximum > minimum)
```

SMM_Unit

# 10 Collective Measures

This diagram represents measures which assess *container* entities by accumulating assessments of *contained* entities which are found by the base measure. See demonstration given in Figure 6.

Most software engineering measures are collective. We count up lines of code for each program block and sum these values to measure routines, programs and eventually applications. A similar process is followed to count operators, operands, operator and operand occurrences, independent paths, and branching points.

Other frequently used container measures are based upon finding the maximum measurement of the container's elements. Nesting depth in a program and class inheritance depth exemplify these collective measures.

The collective measure specifies the following measurement process:

1. Apply the base measure to each contained element to obtain a set of base measurements.
2. Apply the n-ary accumulator to the set of base measurements to obtain the measurement of the container.

Figure 6 demonstrates this process.

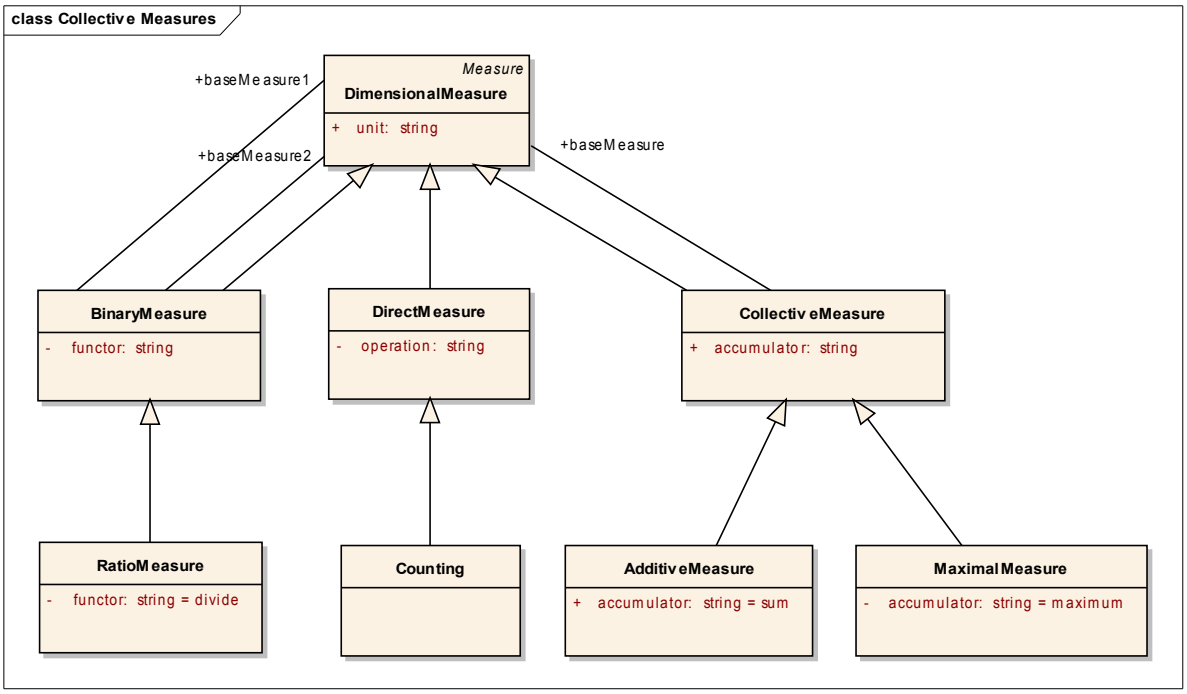


**Figure 5 - Collective Measures**

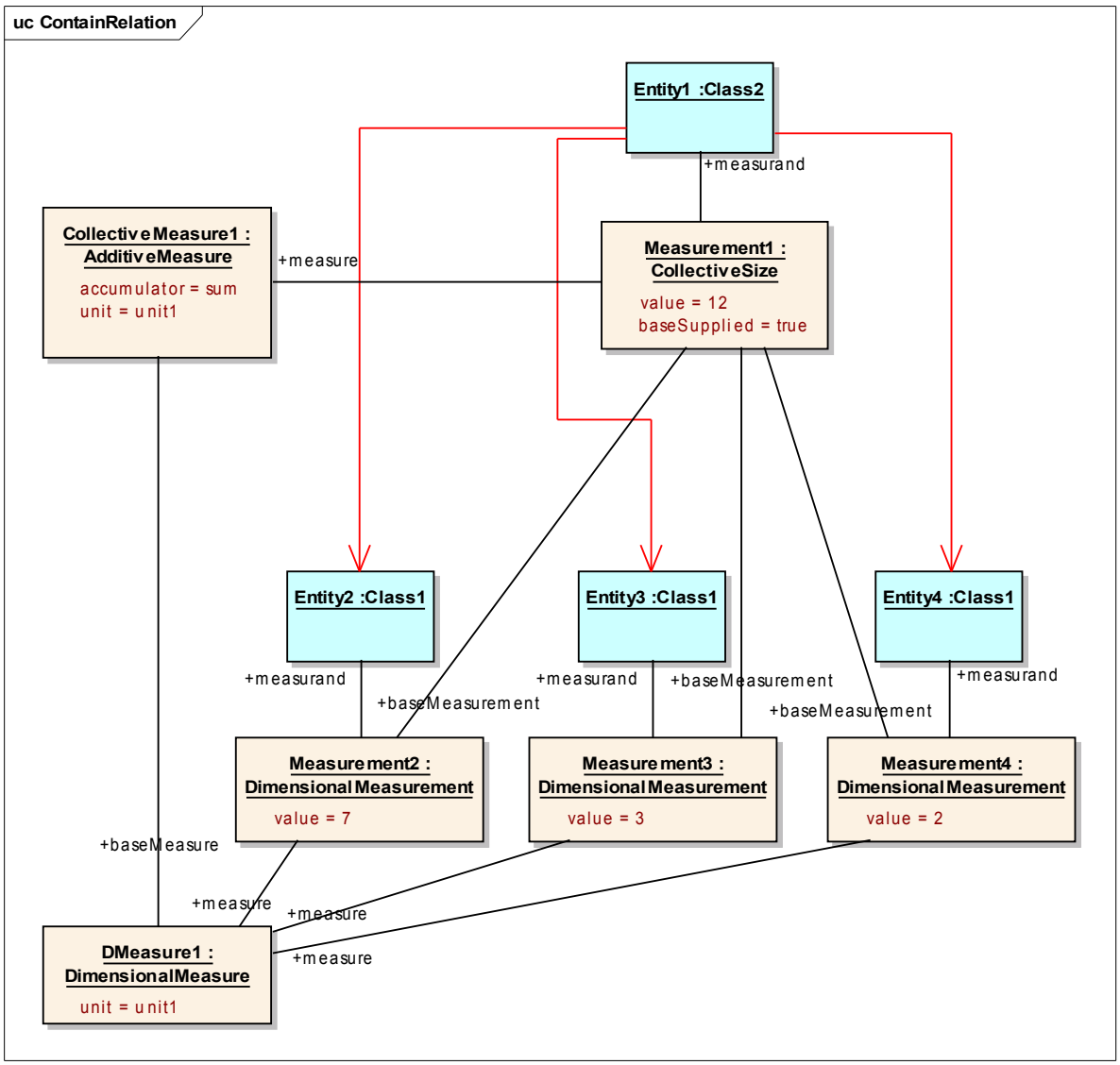


**Figure 6 - Collective Measure Demonstration**

## 10.1 CollectiveMeasure Class

The CollectiveMeasure class represents measures which when applied to a given entity accumulates measurements of entities simalarly related to the given entity. See Figure 5. For example, counts for container entities are often found by accumulating (adding) counts of the containers' contained entities. In fact, sizing measures generally accumulate to containers by adding the results of applying the appropriate size measure to the contained entities.

Maximum is another frequent accumulator.

The measurands of the base measurements need not be the same of the measurand of the collective measurement. Within SMM, the measurands are just arbitary MOF::Elements declared in another MOF model.

The SEI Maintainability Index is one such aggregation which does not change the unit of measure.

#### SuperClass

DimensionalMeasure

#### Attributes

| | |
|---|---|
| accumulator:String | Identifies the n-ary function which accumulates the base measurements. |

#### Associations

| | |
|---|---|
| baseMeasure:DimensionalMeasure | The base measurements are derived by applying the specified measure or refinements of it. |

## 10.2 AdditiveMeasure Class

AdditiveMeasure – a subclass of CollectiveMeasure which sums the measurements of the contained entities. See Figure 5.

#### SuperClass

CollectiveMeasure

#### Constraints

```
context MaximalMeasure inv:
accumulator = 'sum'
```

Accumulator is n-ary addition. If there are no contained entities then zero is returned by this measure.

## 10.3 MaximalMeasure Class

MaximalMeasure – a subclass of CollectiveMeasure which takes the maximum of the measurements of the contained entities. See Figure 5.

#### SuperClass

CollectiveMeasure

#### Constraints

```
context MaximalMeasure inv:
accumulator = 'maximum'.
```

## 10.4 DirectMeasure Class

DirectMeasure – a subclass of DimensionalMeasure which applies a given operation to the measured entity. See Figure 5.

### SuperClass

DimensionalMeasure

### Attributes

| | |
|---|---|
| operation:Operation | Specifies the measurement operation of this measure. It is applicable to elements of the class and returns numeric values interpretable with respect to the unit of measure. |

## 10.5 Counting Class

Counting is a subclass of DirectMeasure where the given operation returns 0 or 1 based upon recognizing the measured entity. See Figure 5.

### SuperClass

DirectMeasure

### Constraints

```
context Counting::self.operation(…):int
post: result = 0 or result = 1
```

The operation is a recognizer which selects some subset of the elements of the measure's scope found by self.scope. The recognizers returns 1 for the elements of the subset and returns 0 otherwise. Self.unit need not be an element of the subset.

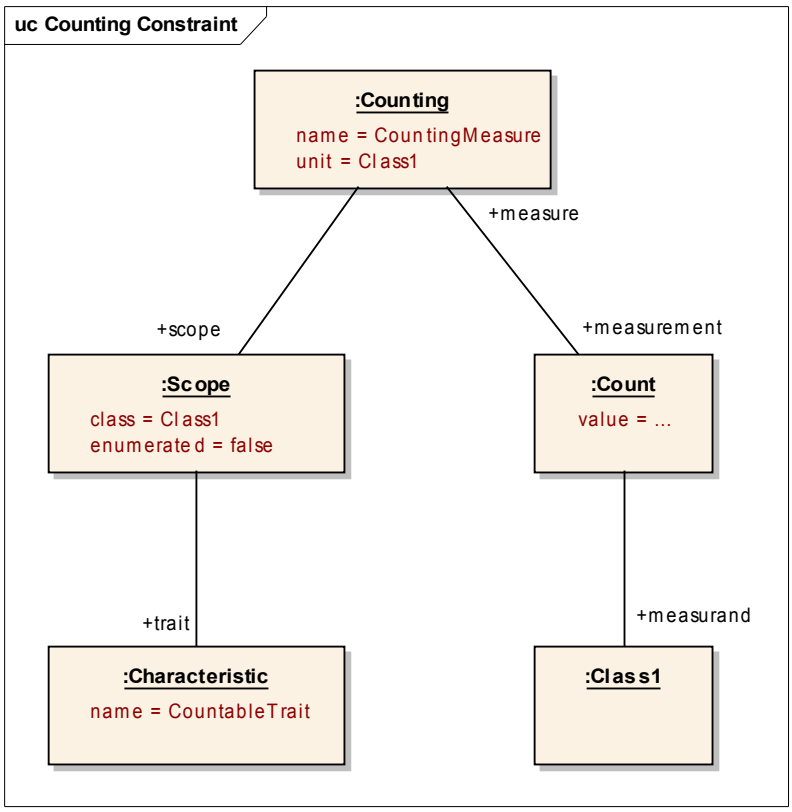


**Figure 7 Counting Unit of Measure Constraint**

## 10.6 BinaryMeasure Class

The BinaryMeasure class represents measures which when applied to a given entity accumulates measurements of two entities related to the given entity. See Figure 5. For example, areas for two dimensional entities are often found by accumulating (multipling) lengths.

The measurands of the base measurements need not be the same as the measurand of the collective measurement.

### SuperClass

DimensionalMeasure

### Attributes

| | |
|---|---|
| functor:String | Identifies the binary function which combines two base measurements. |

### Associations

| | |
|---|---|
| baseMeasure1:DimensionalMeasure | The first base measurement is derived by applying the specified measure or a refinement of it. |
| baseMeasure2:DimensionalMeasure | The second base measurement is derived by applying the specified measure or a refinement of it. |

### Semantics

The usual semantics of algrebra would require that the unit of a binary measure equals applying the accumulator to the units of the base measures. While conforming to this requirement would ensure more easily understood models, SMM does not enforce this requirement.

## 10.7 Ratio Class

This class represents those measures which are ratios of two base measures. See Figure 5. Examples include:

- Average lines of code per module,
- Failures per day,
- Uptime percentage – Uptime divided by total time,
- Business data percentage of all data,
- Halstead level = Halstead volume divided by potential volume,
- Halstead effort = Halstead level divided by volume.

A ratio measure and its two base measurements frequently characterize three different traits of the same entity. If the dividend characterized the total code length of an application and the divisor characterized the number of program in the application then the ratio characterizes the average code length per program.

Ratios may also characterize traits of distinct entities. For example, a ratio may contrast the code length between a pair of programs.

### SuperClass

DimensionalMeasure

### Constraints

```
context MaximalMeasure inv:
functor = 'divide'
```

# 11 Other Measures

The following diagram presents three additional measures.

- Direct applications of named measurements. (One such named measure is Cyclomatic Complexity.)
- Simple algebraic change of scales of already defined numeric measures (e.g., the translation to 'choice points' from Cyclomatic complexity).

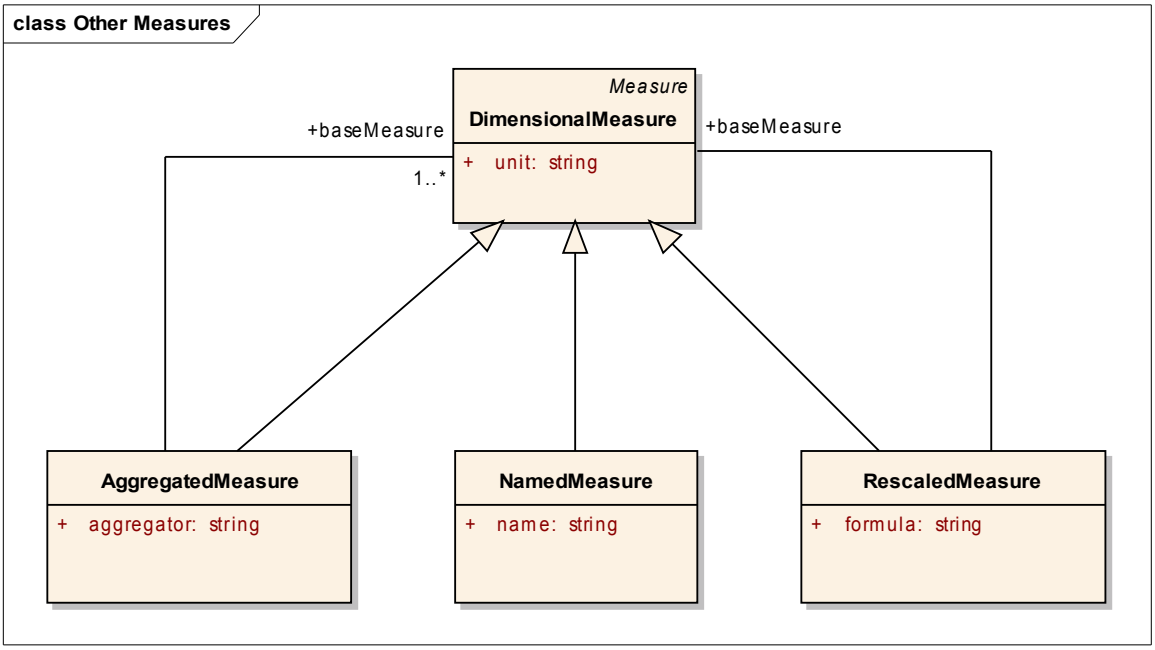


**Figure 8 - Other Measures**

## 11.1 NamedMeasure Class

The class allows for specifying measures which are well-known and can be specify simply by name. See Figure 8. For example, McCabe's cyclomatic complexity. The meaning of applying the named measure should be generally accepted.

SMM is for the exchange of measurement results. To convery such results for well known measures, it suffices to identify the measure solely by name.

### SuperClass

DimensionalMeasure

### Attributes

| | |
|---|---|
| name: String | Specifies the name of the SMM element. This attribute is inherited from the Element class where it is optional. Here it is required. |

### Constraints

```
context NamedMeasure inv:
not self.name->isEmpty
```

## 11.2 RescaledMeasure Class

The measure specifies a process which re-scales a measurement on an entity with one unit of measure to obtain a second measurement of the same entity with an different unit of measure. See Figure 8.

### SuperClass

DimensionalMeasure

### Attributes

| | |
|---|---|
| formula:String | Specifies the algebraic formula which re-scales a result from the base measure's dimension to obtain a value expressed in a different unit of measure with respect to this measure's unit of measure |

### Associations

| | |
|---|---|
| baseMeasure:DimensionalMeasure | Identifies the measure applied to each "contained" entity to determine base measurements. |

# 12 Measurements

Measurement results are values from ordered sets. Such a set may be nominal (e.g. Poor, Fair, Good, Excellent) as long as there is an underlying order. A set may instead define a dimension where its values may be stated in orders of magnitude with respect to archetypal member. SMM allows for dimensional measurements. The magnitude is the measure's unit of measure.

SMM also allows for dimensionless measurements derived by ratios and ranking schemes. In the former the ratio is derived from two measurements of the same dimension; whereas, in the latter measurements from a dimension are mapped to symbolic representations (e.g., 100-90 becomes "A," 89-80 becomes "B").

The modeling of measurements mirrors the modeling of measure.

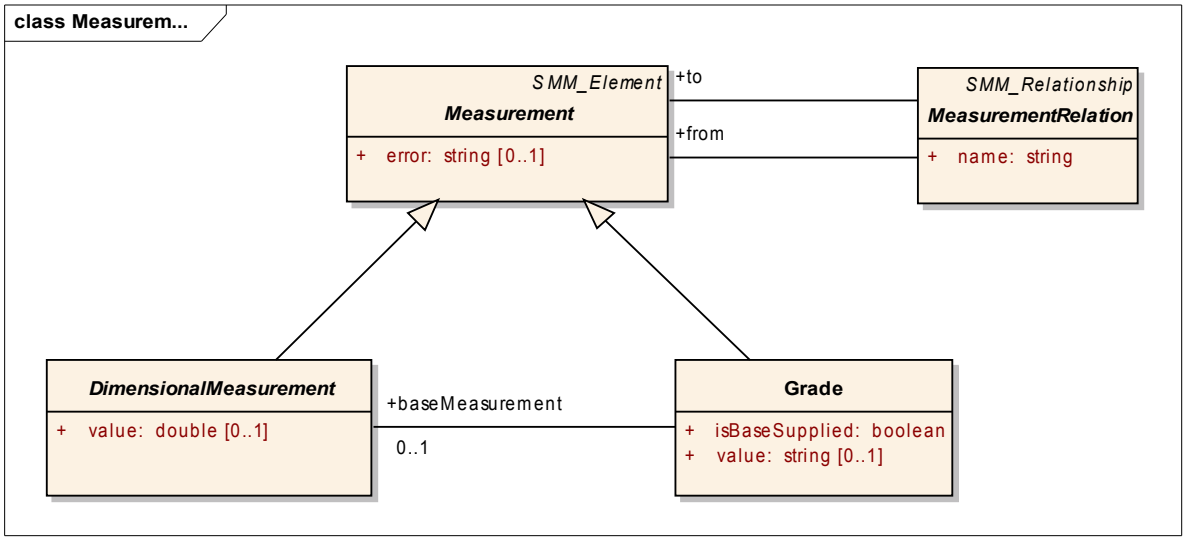


**Figure 9 - Measurements**

## 12.1 Measurement Class (abstract)

The Measurement class represents the results of applying the associated Measure to the associated Measurand. See . Two measurements of the same measurand by the same measure can be distinguished by observation information provided by the associated Observation.

Measurand is in the scope of the measure.

The value of a measurement is an element of an ordered set. It may be a number where the ordering is the usual standard. The DimensionalMeasurement and Percentage subclasses of Measurement defined below have numeric values. The value may also be a symbol that we can map to a numeric interval. The Grade subclass has a symbolic value.

Measure is a process and, hence, may fail. The error attribute of measurement allows such failures to noted. A measurement either has a value or an error is recorded.

### SuperClass

SMM_Element

### Attributes

| | |
|---|---|
| error:String[0..1] | If an error occurred in the measurement process, this field contains a code representing the error. |

### Associations

| | |
|---|---|
| measure:Measure | Identifies the process by which the measurement was determined. |
| measurand:MOF::Element | Identifies the object measured. |
| observation:Observation | Provides contextual information which may distinguish this measurement from other assessments by the same measure on the same measurand. |

### Semantics

Measurand must be in the scope of measure. Specifically, measurand must be an instance of the class named in measure.characterizes.scope.class. If class named in measure.characterizes.scope.enumerated is true then measurand is associated as an element to class named in measure.characterizes.scope. Otherwise, if measure.characterizes.scope.recognizers is given then the recognizer applied to the measurand must return true.

If the measure is identified by name and library, then the measure's measurable trait need not appear when convey of measurement. In that case the definitive measure is given in the named library with the given name. The measurable trait is found in the library by following the associated characterizes role.

## 12.2 MeasurementRelation (abstract)

MeasurementRelation is an abstract class representing any relationship between two measurements. See Figure 9 .

### SuperClass

SMM_Relationship

## 12.3 DimensionalMeasurement Class

The DimensionalMeasurement class represents the results of applying a dimensional measure to an entity. The result is given in terms of the measure's unit. See Figure 9.

### SuperClass

Measurement

### Attributes

| | |
|---|---|
| value:Number[0..1] | Represents the measurement result as a magnitude with respect to the unit of measure. |

### Constraints

```
context DimensionalMeasurement inv:
measure.oclIsTypeOf(DimensionalMeasure) and
error->isEmpty <> value->isEmpty
```

## 12.4 Grade Class

The Grade class represents the grade found by Ranking measure. Its ranking scheme mapped the grade's underlying base measurement to the grade's symbol. Once again, the base measurements shares its measurand with this derived gradingis. See Figure 9.

### SuperClass

Measurement

### Attributes

| | |
|---|---|
| value:String[0..1] | Identifies rank as a measurement derived from the base measurement. |
| isBaseSupplied:Boolean | True if baseMeasurement is supplied. |

### Associations

| | |
|---|---|
| baseMeasurement:DimensionalMeasurement[0..1] | Identifies the measurement from which the rank was derived. |

### Constraints

```
context Grade inv:
measure.oclIsTypeOf(Ranking) and
error->isEmpty <> value->isEmpty and
isBaseSupplied →(measurand = baseMeasurement.measurand and
baseMeasurement.measure = measure.baseMeasure)
```

### Semantics

If isBaseSupplied holds, then value is one of the symbols found by measure.interval where baseMeasurement.value is in the interval. A numeric value is in the interval if and only if the it is less than the maximumEndPoint when maximumOpen is false, less than or equal to maximumEndPoint when maximumOpen is true, greater than minimumEndPoint when minimumOpen is false, and greater than or equal to minimumEndPoint when minimumOpen is true.

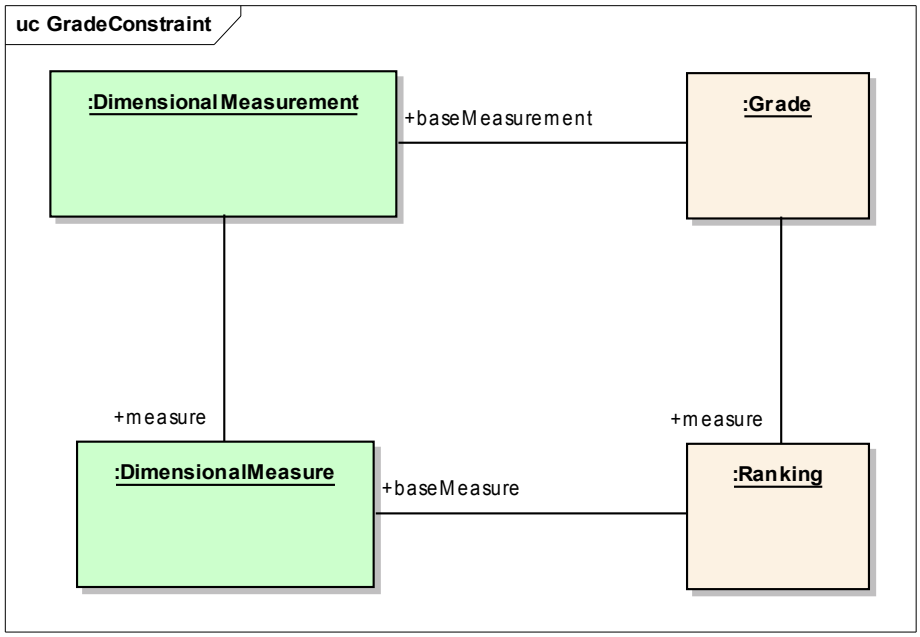


**Figure 10 - Grade Constraint**

# 13 Collective Measurements

This class represents measurements found by accumulating a set of base measurements. For example, the number lines of code in application can be determines by accumulating the number lines in its programs.

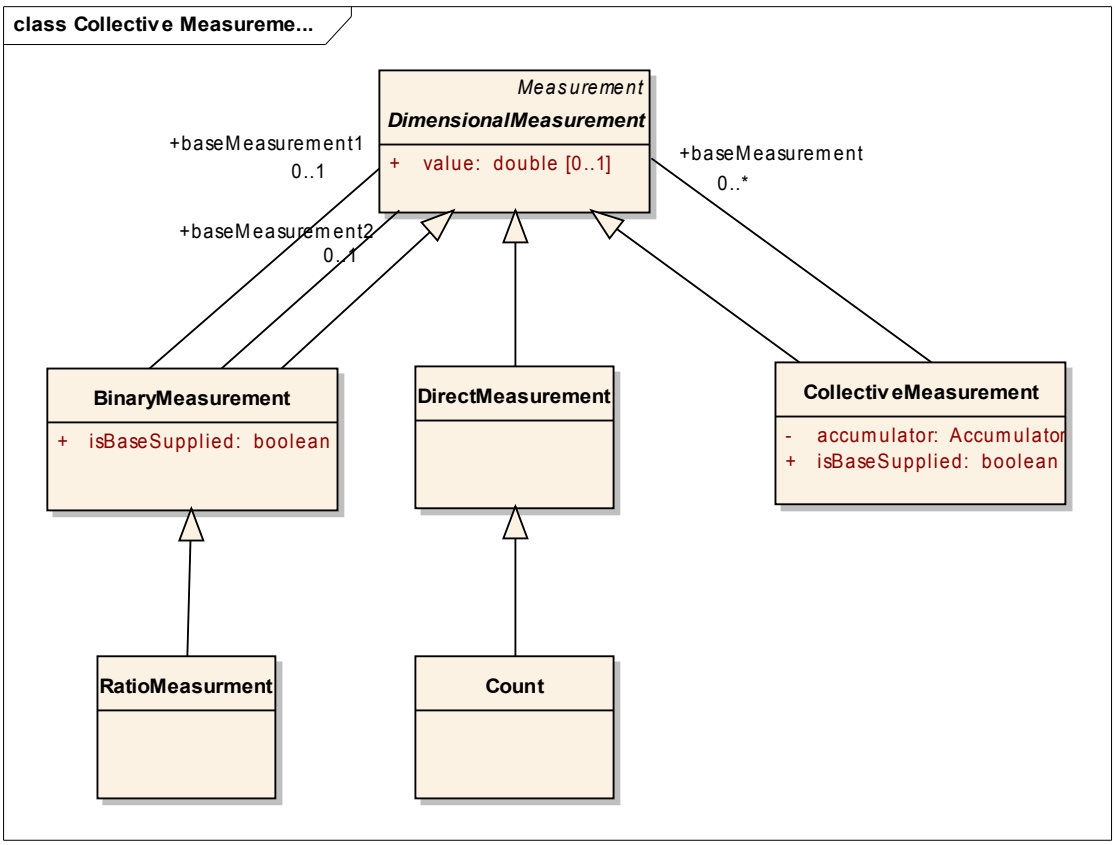


**Figure 11 - Collective Measurements**

## 13.1 CollectiveMeasurement Class

The CollectiveMeasurement class represents the results of applying its CollectiveMeasure measure to an entity. See Figure 11. In this case, applying the measure is as follows:

1. Apply the base measure to each contained element to obtain a set of base measurements.
2. Apply the n-ary accumulator to the set of base measurements to obtain the measurement of the container.

The results of step 1 are the DimensionalMeasurements associated by base measurement.

### SuperClass

DimensionalMeasurement

### Attributes

| | |
|---|---|
| isBaseSupplied:Boolean | True if baseMeasurements are supplied. All are supplied or none is assumed. |
| accumulator: Accumulator | Enumerated value indicating the type collective measure |

#### Associations

| | |
|---|---|
| baseMeasurement:DimensionalMeasurement[0..*] | Identifies the measurements from which this collective measurement was derived. |

#### Constraints

```
context CollectiveMeasurement inv:
measure.oclIsTypeOf(CollectiveMeasure) and
isBaseSupplied →
(not baseMeasurement->isEmpty and baseMeasurement.measure=measure.baseMeasure)
```

#### Semantics

If isBaseSupplied holds, then value equals the result of applying measure.accumulator the set of values given by baseMeasurement.value.

## 13.2 DirectMeasurement Class

The DirectMeasurement class represents the measurement results found by of applying the measure's specified operation directly to the measurand. See Figure 11.

#### SuperClass

DimensionalMeasurement

#### Constraints

```
context DirectMeasurement inv:
measure.oclIsTypeOf (DirectMeasure)
```

## 13.3 Count Class

Counting forms the basis for multiple software metrics. This class consists of a particular subclass of directMeasurement which is very useful in counting. See Figure 11. Its associated measure is a CountingMeasure where the specified operation is a recognizer operation. Therefore, the value of any instance of this class is 1 or 0 depending upon whether or not the measurand is recognized.

#### SuperClass

DirectMeasurement

#### Constraints

```
context Count inv:
measure.oclIsTypeOf (CountingMeasure)
```

## 13.4 BinaryMeasurement Class

### SuperClass

DimensionalMeasurement

### Attributes

| | |
|---|---|
| isBaseSupplied:Boolean | True if both base measurements are supplied. |

### Associations

| | |
|---|---|
| baseMeasurement1:DimensionalMeasurement[0..1] | Identifies the first base measurement. |
| baseMeasurement2:DimensionalMeasurement[0..1] | Identifies the second measurement. |

### Constraints

```
context RatioMeasurement inv:
measure.oclIsTypeOf(BinaryMeasure) and
isBaseSupplied →
(not baseMeasurement1.isEmpty and not baseMeasurement2.isEmpty) and
not baseMeasurement1.isEmpty →
(baseMeasurement1.measure = measure. baseMeasurement1) and
not baseMeasurement2.isEmpty →
(baseMeasurement2.measure = measure. baseMeasure2)
```

### Semantics

If isBaseSupplied holds, then value equals the result of applying measure.functor to baseMeasurement1.value and baseMeasurement2.value.

## 13.5 RatioMeasurement Class

The RatioMeasurement class affords evaluations of a ratio measure of two evaluations of different dimensional measures. See Figure 11. The measure associated with the dividend has its unit of measure in common with the measure associated with the divisor.

### SuperClass

BinaryMeasurement

### Constraints

```
context RatioMeasurement inv:
measure.oclIsTypeOf(RatioMeasure)  and
isBaseSupplied → (value = baseMeasurement1.value / baseMeasurement2.value)
```

# 14 Named and ReScaled Measurements

Measurement is in terms of its unit of measure as specified under its associated DimensionalMeasure. That is, the measurement is a multiple of its unit of measure where value determines the multiple.

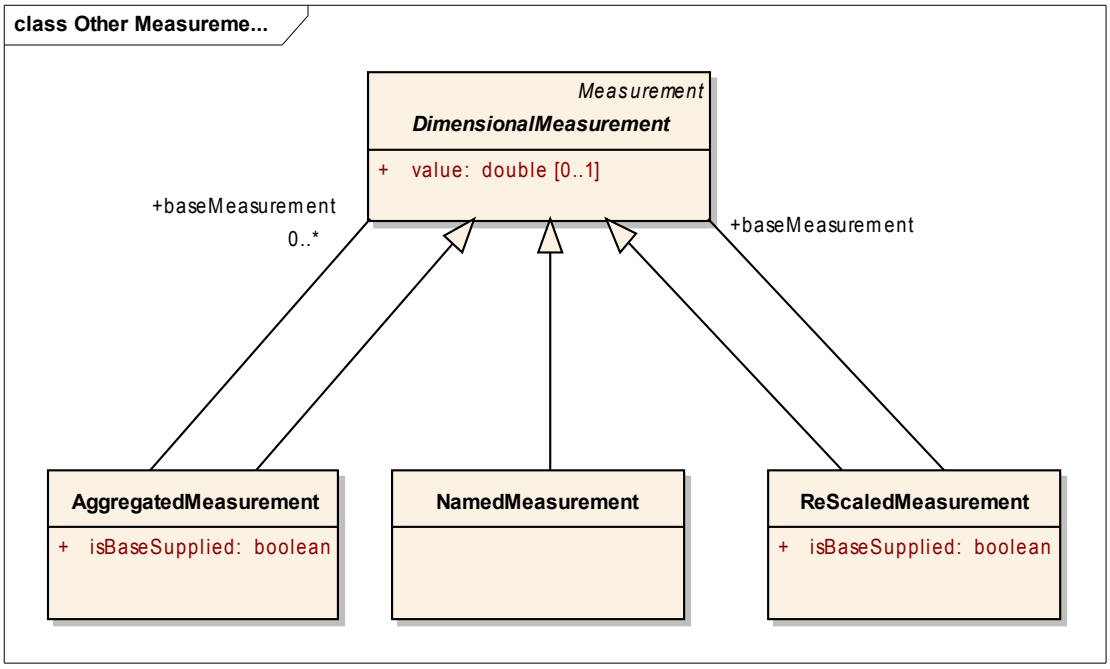


**Figure 12 - Named and ReScaled Measurements**

## 14.1 AggregatedMeasurement Class

The AggregatedMeasurement class represents the measurement results of applying the operation specified by the measure to the base measurements. See Figure 12. Its measurand and the measurand of its base measurement are identical. That is, this is not a measurement of a container as represented by the CollectiveMeasurement. Instead, AggregatedMeasurement combines different measurements of a given entity to create a new measurement for that entity. The SEI Maintainability index demonstrates this process.

$$171 - 5.2(\ln(aveV)) - 0.23(aveV(g')) - 16.2(\ln(aveLOC)) + 50(\sin(\sqrt{2.4(perCM)}))$$

### SuperClass

DimensionalMeasure

### Attributes

| | |
|---|---|
| isBaseSupplied:Boolean | True if base measurements are supplied. All are supplied or none is assumed. |

### Associations

| | |
|---|---|
| baseMeasurement:DimensionalMeasurement[0..*] | Identifies the measurements from which this aggregated measurement was derived. |

### Constraints

```
context AggregatedMeasurement inv:
measure.oclIsTypeOf(AggregatedMeasure) and
isBaseSupplied → (not baseMeasurement->isEmpty) and
forAll(b:baseMeasurement | b.measure = measure.baseMeasure)
```

### Semantics

If isBaseSupplied holds, then value equals the result of applying measure.accumulator the set of values given by baseMeasurement.value.

## 14.2 NamedMeasurement Class

The NamedMeasurement class represents the measurement results of applying to the Measurand measurement processes which are generally known and identifiable by name. See Figure 12.

### SuperClass

DimensionalMeasure

### Constraints

```
context NamedMeasurement inv:
measure.oclIsTypeOf(NamedMeasure).
```

## 14.3 ReScaledMeasurement Class

The ReScaledMeasurement class represents the measurement results of applying to the base measurement the operation specified by the Measure to rescale the measurement. That is, given a one measurement of the measurand with respect to one unit of measure, we obtain a second measurement of the measurand with respect to a different unit of measure. See Figure 12.

Measure is a RescaledMeasure.

### SuperClass

DimensionalMeasure

### Attributes

| | |
|---|---|
| isBaseSupplied:Boolean | True if the base measurement is supplied. |

### Associations

| | |
|---|---|
| baseMeasurement:DimensionalMeasurement[0..1] | Identifies the measurement from which this measurement was derived. |

### Constraints

```
context ReScaledMeasurement inv:
measure.oclIsTypeOf(RescaledMeasure) and
isBaseSupplied →
not baseMeasurement->isEmpty and baseMeasurement.measure = measure.baseMeasure
```

### Semantics

If isBaseSupplied is true then value equals result of applying measure.operation to the baseMeasurements' values.

# 15 Observations

Measurements are sometimes repeated. An old carpentry rule is measure twice, cut once.
To distinguish these multiple measurements, the observation class can respresent contextual information such as the time of the measurement and the identification of the measurement tool.

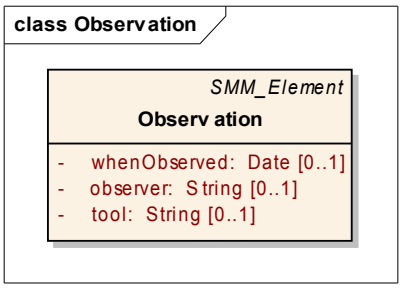


**Figure 13 - Observations**

## 15.1 Observation Class

This class represents some of the contextual information which may be unique to this measurement such as date, measurer and tool used. See Figure 13.

### SuperClass

SMM_Element

### Attributes

| | |
|---|---|
| whenObserved:date[0..1] | Identifies the "moment" when the measurementwas taken. |
| observer:String[0..1] | Identifies measurer. |
| tool:String[0..1] | Identifies tool used in measurement. |

# 16 Historic and Trend Data (Non-Normative)

SMM does not model tracking or trend data directly. Linking versions of objects through a software evolution poses a concern in modeling software evolution even if measures are never taken. When the measurand's model provides the linkage (e.g. an "EvolvesTo" relationship), then a measurement of an original artifact could be traced to its newer versions and to their measurements if available. The diagram below (Figure 14) is overly simplistic, but hopefully conveys the gist of such tracing. The beige filled instances indicates the metric representations augmenting the base model (green). The central point is that the evolves path is between instances of the base model. The measures of the evolving artifacts can be gathered or compared only if the linkage between the artifacts is captured and maintained through the modeling of the system development and modification.

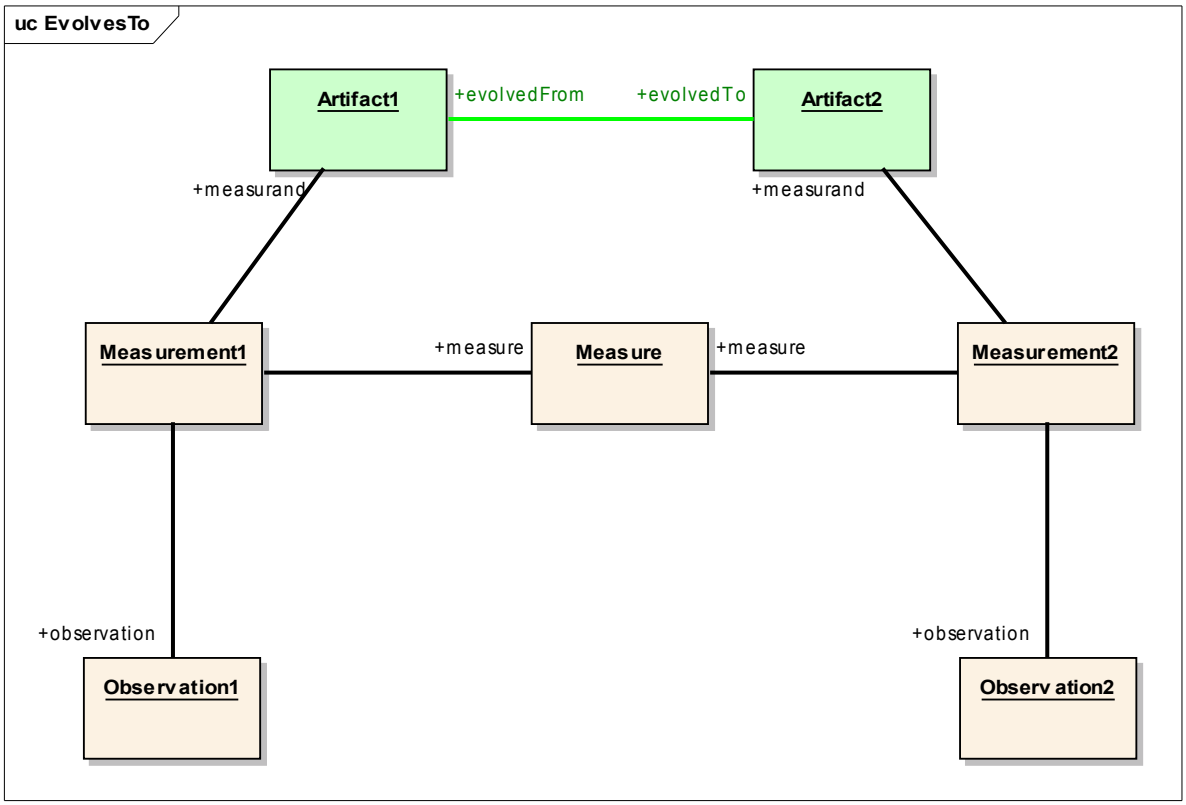


**Figure 14 - Tracking Measurements Across Versions**

# 17 Inaccuracy (Non-Normative)

Inaccuracy of a measurement is the amount by which the measurement is in error. That is, we may model inaccuracy as measure if we first model a measure which is assumed to be true. Inaccuracy of a measurement is then just the difference between the measurement and a "true" measurement of the same entity.

In SMM inaccuracy is representable by measures which characterizes inaccuracy. The measures are comparable elevation of measurements evaluated bythe difference between the measurement and the truest (at least accepted as such) measurement of that entity for that trait.

Given two measures which characterizes the same trait and share the same scope, then inaccuracy can be modeled as

In the demonstration below (Figure 15), a category collects measures which are applicable to ExampleClass1 and characterize ExampleTrait. The category identifies the "truest" measure by the goldStandard relationship and identifies an appropriate inaccuracy measure for Measure1 by the InaccuracyMeasure relationship.

A Characteristic may have a measure which is designated as the best or truest measure of the attribute. That measure may be associated as the attribute's gold standard. Such a designation allows for the representation of inaccuracy for each of the attribute's measures as the difference between the measure and the gold standard.

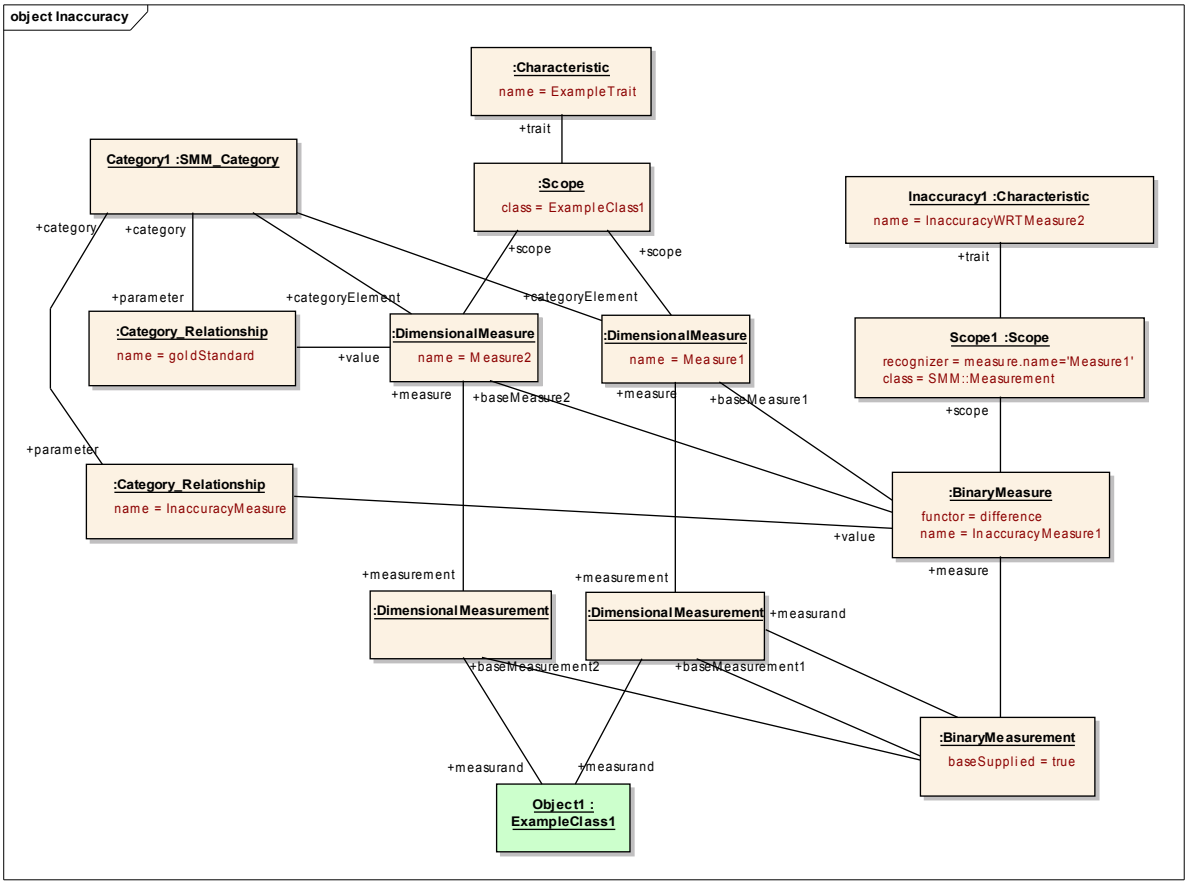


**Figure 15 - Inaccuracy Demonstration**

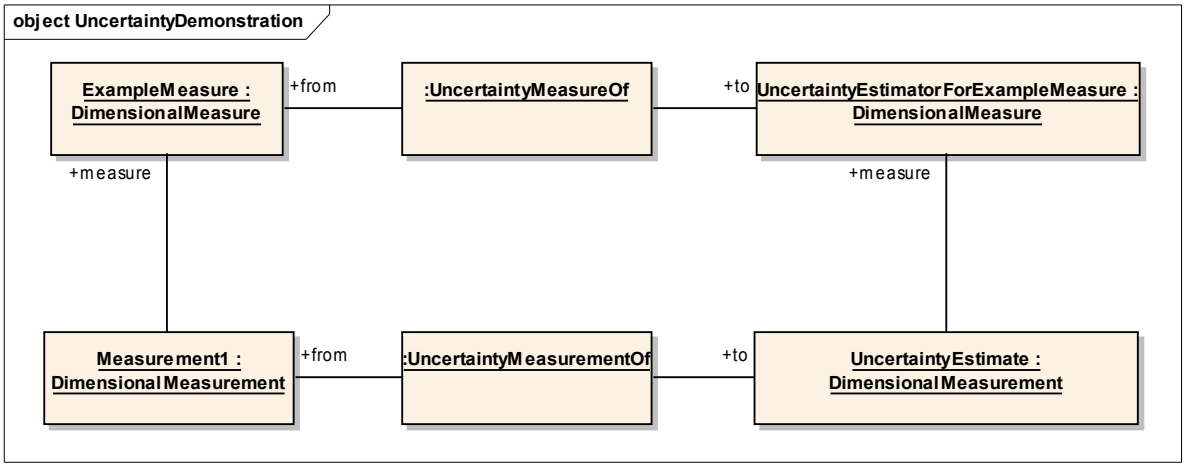


**Figure 16 - Uncertainty Demonstration**

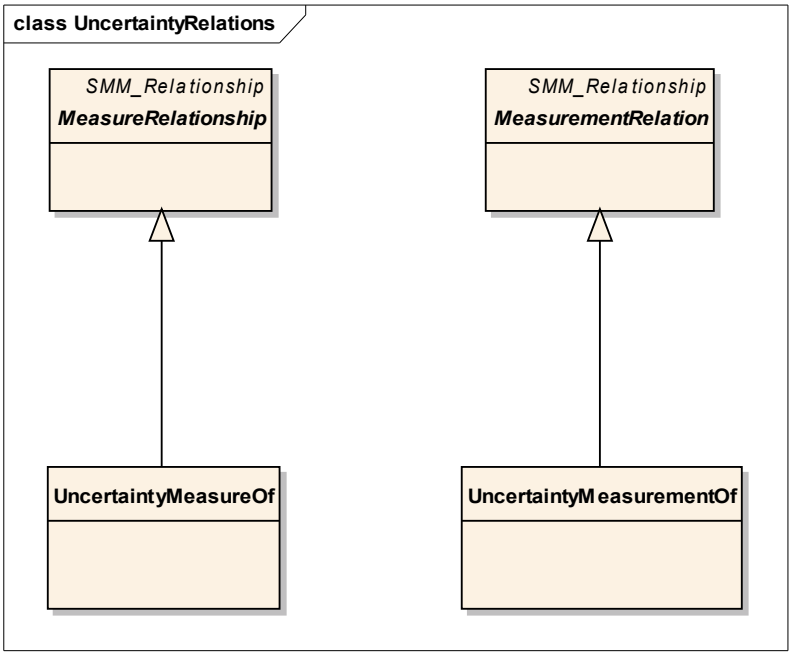


**Figure 17 - SMM Extension for Uncertainty**

# 18 Library of Measures (Non-Normative)

The following is a suggestive list of measurement classes along with their measure classes and measurand classes. Sources include:

- *Comsys Systems Redevelopment Methodology: www.comsysprojects.com/SystemTransformation/TMethodology.htm*
- "A Survey of Software Metrics" by F. Riguzzi, DEIS Technical Report no. DEIS-LIA-96-010, July 1996, Università degli Studi di Bologna.

Each measure is defined using the classes of the SMM. The referenced software artifacts are modeled using the Knowledge Discovery Metamodel (KDM) unless otherwise noted.

## 18.1 Various Counts

### 18.1.1 Module Count[3]

Module Count ≡ A count of the number of modules in a system.

Assume that the system is modeled by a KDM model. The KDM:AbstractCodeElement serves as a container of code parts as well as modeling the code parts themselves. The KDM:Module is an AbstractCodeElement subclass which models modules. See Figure 18.

Counting the modules in the code model requires summing the results of a recognizer for module across the model. The unit of measure is module. See Figure 19 for the library entry and see Figure 20 for a brief demonstration.

[3] See GAM 003 in Comsys Systems Redevelopment Methodology.

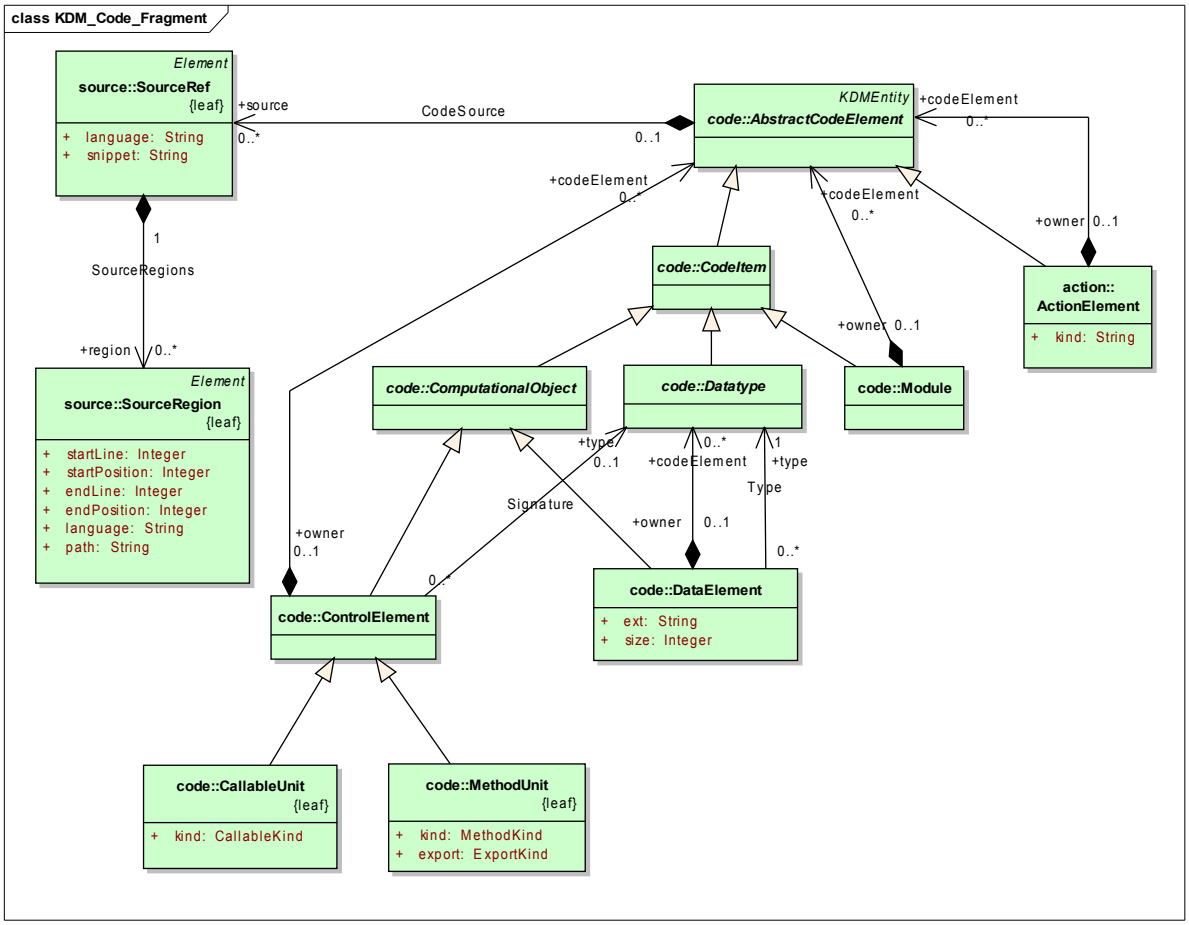


**Figure 18 - KDM Code Package Fragment**

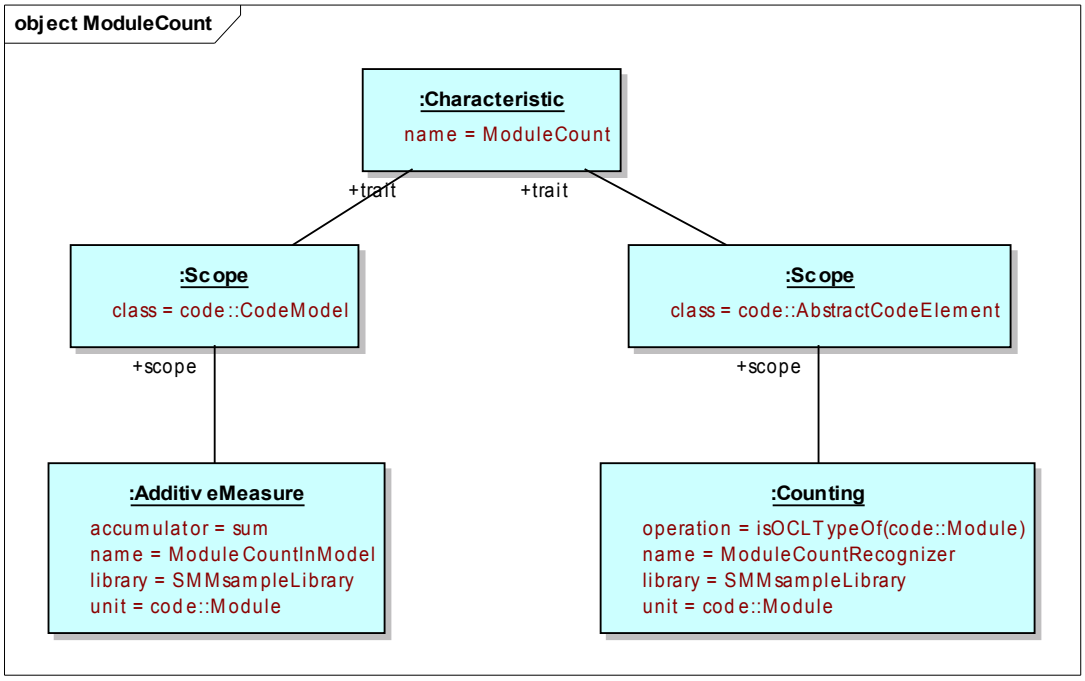


**Figure 19 - Library Entry for Module Count in Code Model**

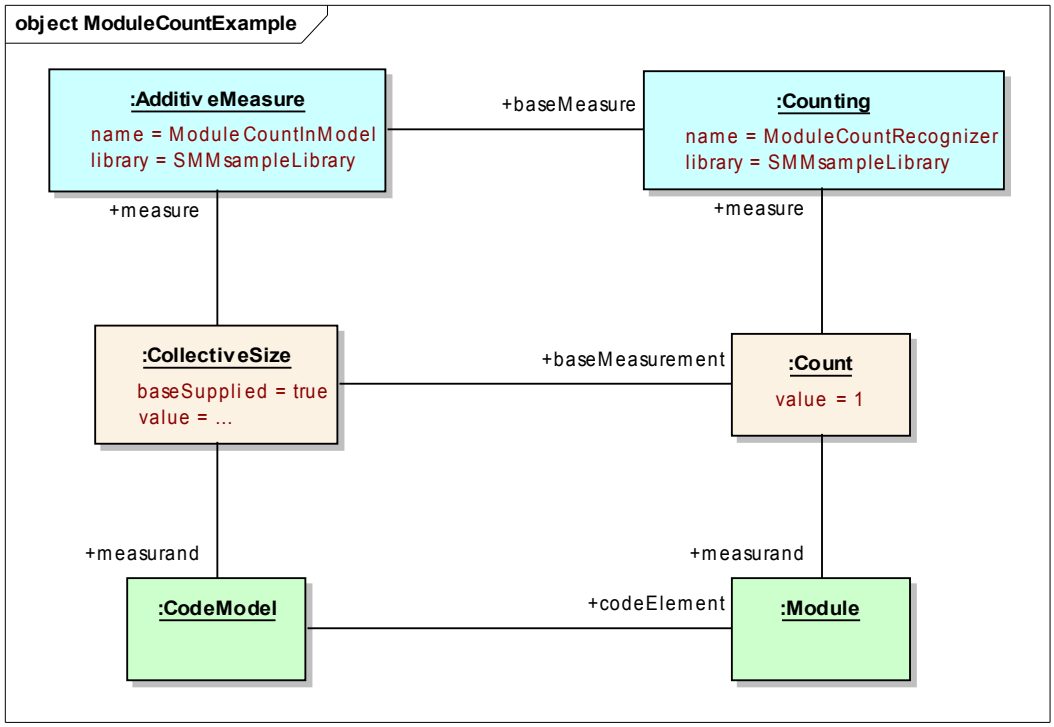


**Figure 20 Module Count in Model Demonstration**

Counting the modules in an abstract code element sums recursively the count up the code part heirarchy.

It requires noticing if the code element is a module and returning 1 as well as recursively counting the modules in all the contained code elements. This is a CollectiveMeasure which sums two base measures. The first is a CountingMeasure which recognizes modules. The second is a sum accumulator of the owner/codeElement association from CodeElement to CodeElement and its base measure is the above CollectiveMeasure. The unit of each of these measures is a module.

For the entire system, we count the modules in the CodeModel which owns the top-level code elements of the system. The counting is a CollectiveMeasure with a sum accumulator of the model/codeElement association from CodeModel to CodeElement and its base measure is the above counting of modules in a code element.

### 18.1.2 Screen Count[4]

Screen Count ≡ A count of the number of screens in a system.

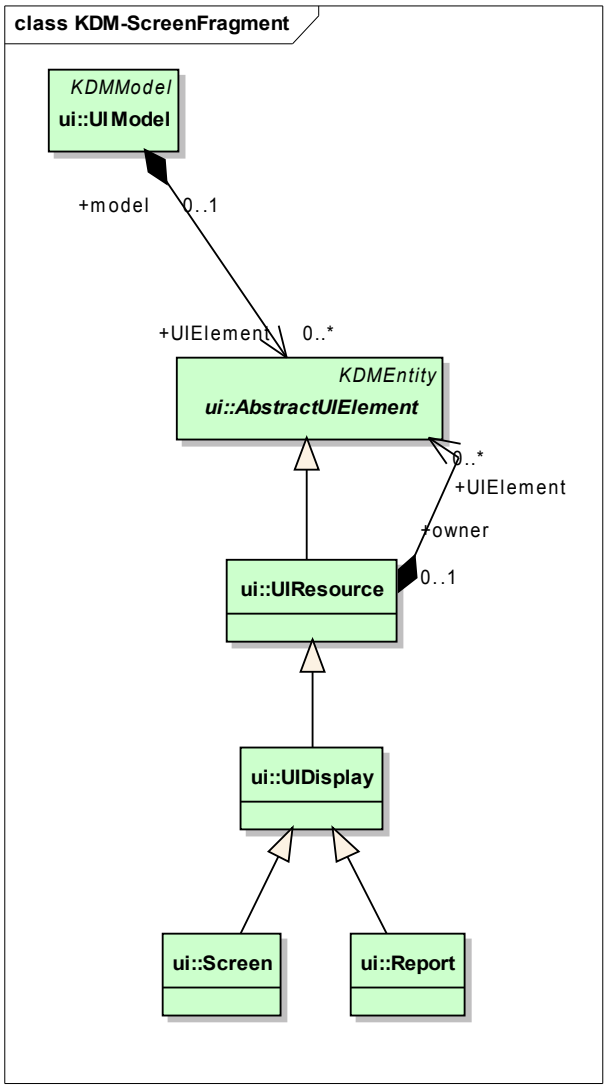


**Figure 21 - KDM Action Package Fragment**

[4] See TEM 153 in Comsys Systems Redevelopment Methodology.

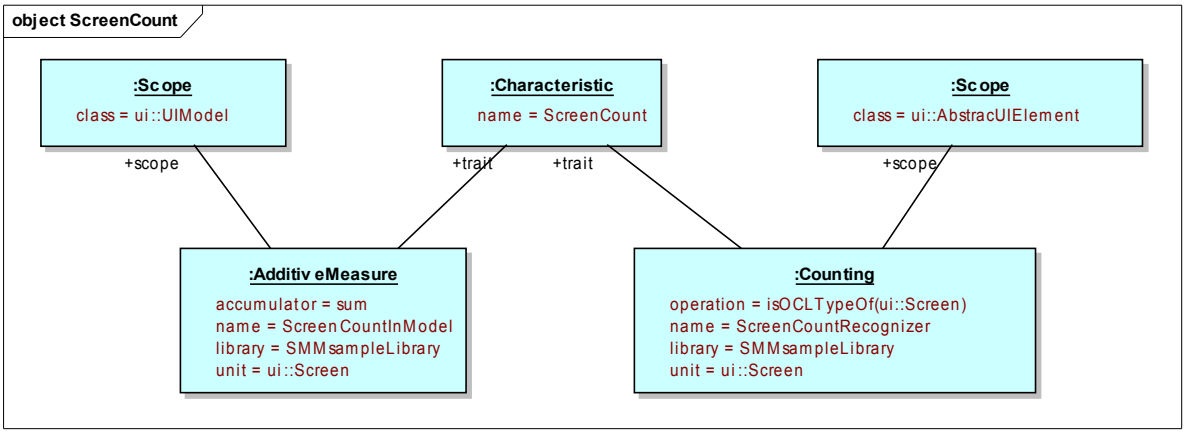


**Figure 22 Screen Count Library Entry**

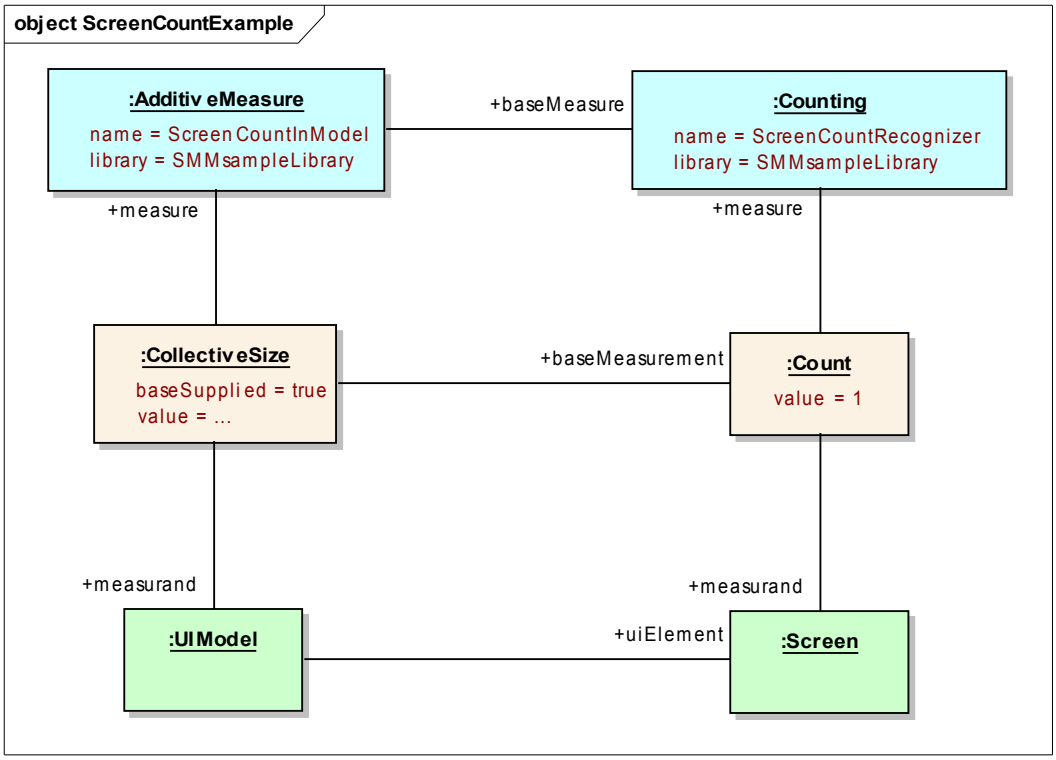


**Figure 23 Screen Count Demonstration**

Assume that the system is modeled by a KDM model. The KDM:UIElement serves as a container of user interface parts as well as modeling the user interface parts themselves. The KDM:Screen is a UIElement subclass which models screens.

Count the screens in a code element requires noticing if the user interface element is a screen and returning 1 as well as recursively counting the screens in all the contained user interface elements. This is a CollectiveMeasure which sums two base measures. The first is a CountingMeasure which recognizes screens. The second is a sum accumulator of the

owner/UIElement association from UIElement to UIElement and its base measure is the above CollectiveMeasure. The unit of each of these measures is a screen.

For the entire system, we count the screens in the UIModel which owns the top-level user interface elements of the system. The counting is a sum accumulator of the model/uiElement association from UIModel to UIElement and its base measure is the above counting of screens in a user interface element. The unit of measure is "each".

### 18.1.3 Method Count

Method Count ≡ A count of the number of methods in a system.

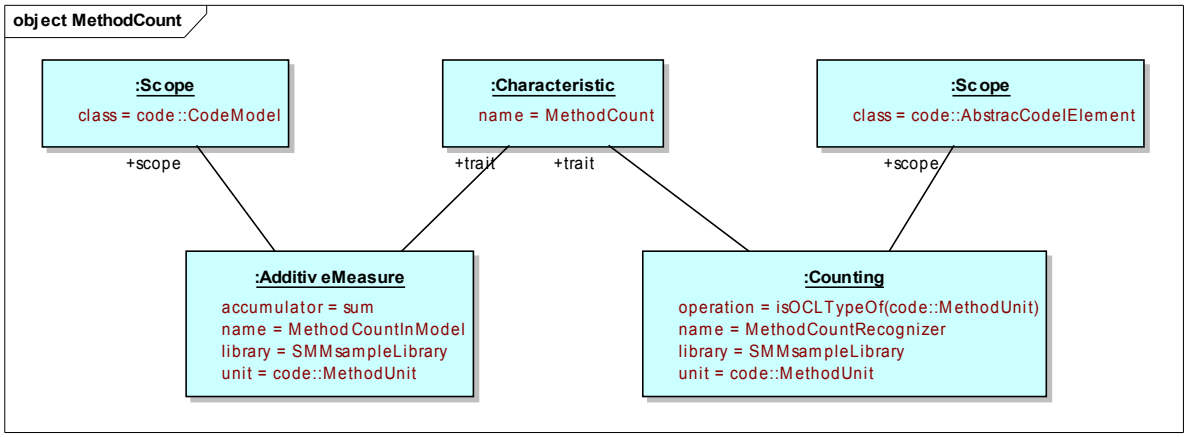


**Figure 24 Method Count Library Entry**

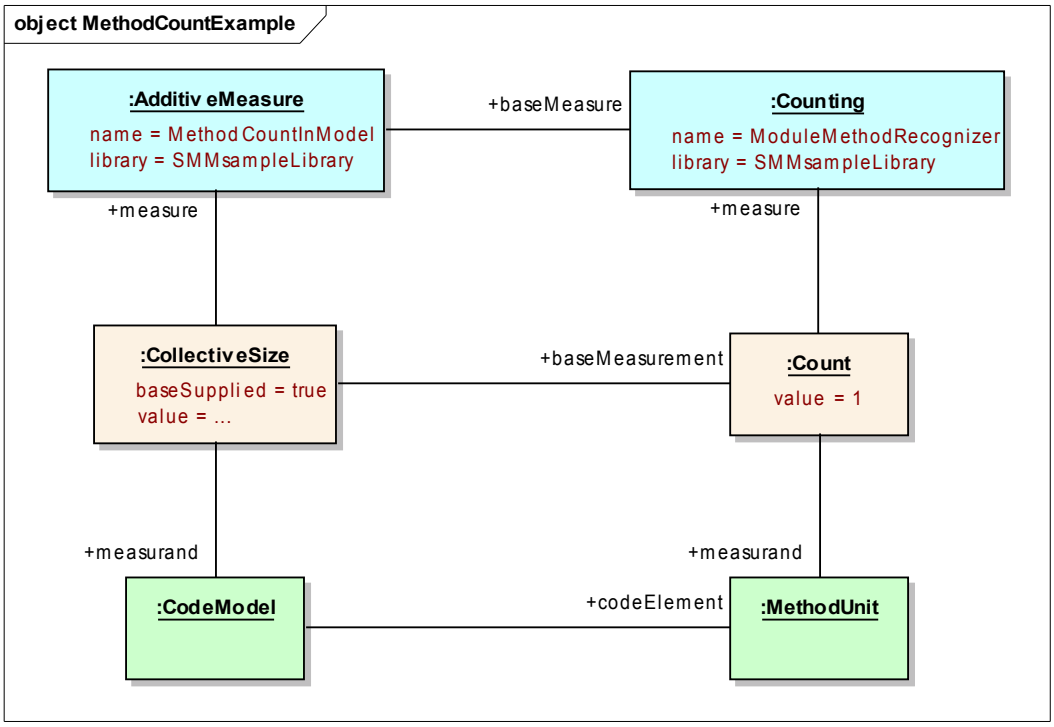


**Figure 25 - Method Count Demonstration**

Assume that the system is modeled by a KDM model. The KDM:MethodUnit is a CodeElement subclass which models methods. The counting of methods then is very similar to the counting of modules given above.

Counting the modules in a code element requires noticing if the code element is a method and returning 1 as well as recursively counting the methods in all the contained code elements. This is an CollectiveMeasure which sums two base measures. The first is a CountingMeasure which recognizes methods. The second is a sum accumulator of the owner/codeElement association from codeElement to codeElement and its base measure is the above CollectiveMeasure. The unit of each of these measures is a method.

For the entire system, we count the methods in the CodeModel which owns the top-level code elements of the system. The counting is a sum accumulator of the model/codeElement association from CodeModel to CodeElement and its base measure is the above counting of modules in a code element. The unit of measures is a method.

### 18.1.4 Lines of Code[5]

A line of code is any line of program text that is not a comment or a blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program headers, declarations, and executable and non-executable statements"[6] Lines of code here means fully expanded lines of code including copy books, includes and comments.

KDM does not directly model lines of source, code or otherwise. As a demonstration, let us assume that blank lines may be included. This allows us to use the KDM SourceRegion to measure lines of code. We will further assume source region do not overlap or even having one start on the line that another ends on. The problem here is that code snippets are the smallest pieces of source modeled in KDM. Lines by themselves are not which means counting them is indirect. We will sum of the line size of code snippets and call that counting lines of code.

#### Lines of SourceRegion and SourceRef

KDM specifies a code snippet with a SourceRegion element which have two attributes, startLine and endLine, that interest us here. The number of lines in the SourceRegion is endLine – StartLine + 1.

Our representation is a DirectMeasure with a class of SourceRegion and a function of endLine – startLine + 1.

SourceRef consists of multiple SourceRegions. Assuming no overlap as stated above, the determination the lines of code in a SourceRef is an AdditiveMeasure with the previous lines of SourceRegion as its base measure.

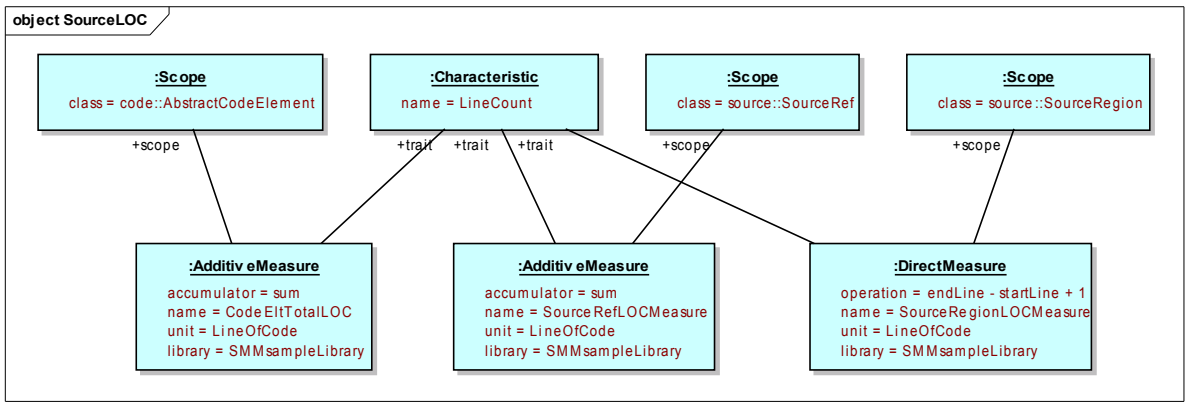


**Figure 26 Lines of Code Measures**

[5] See ERP 001 in Comsys Systems Redevelopment Methodology.

[6] See S. Conte, H. Dunsmore, V. Shen, *Software Engineering Metrics and Models*, Benjamin/Cummings, Menlo Park, CA.

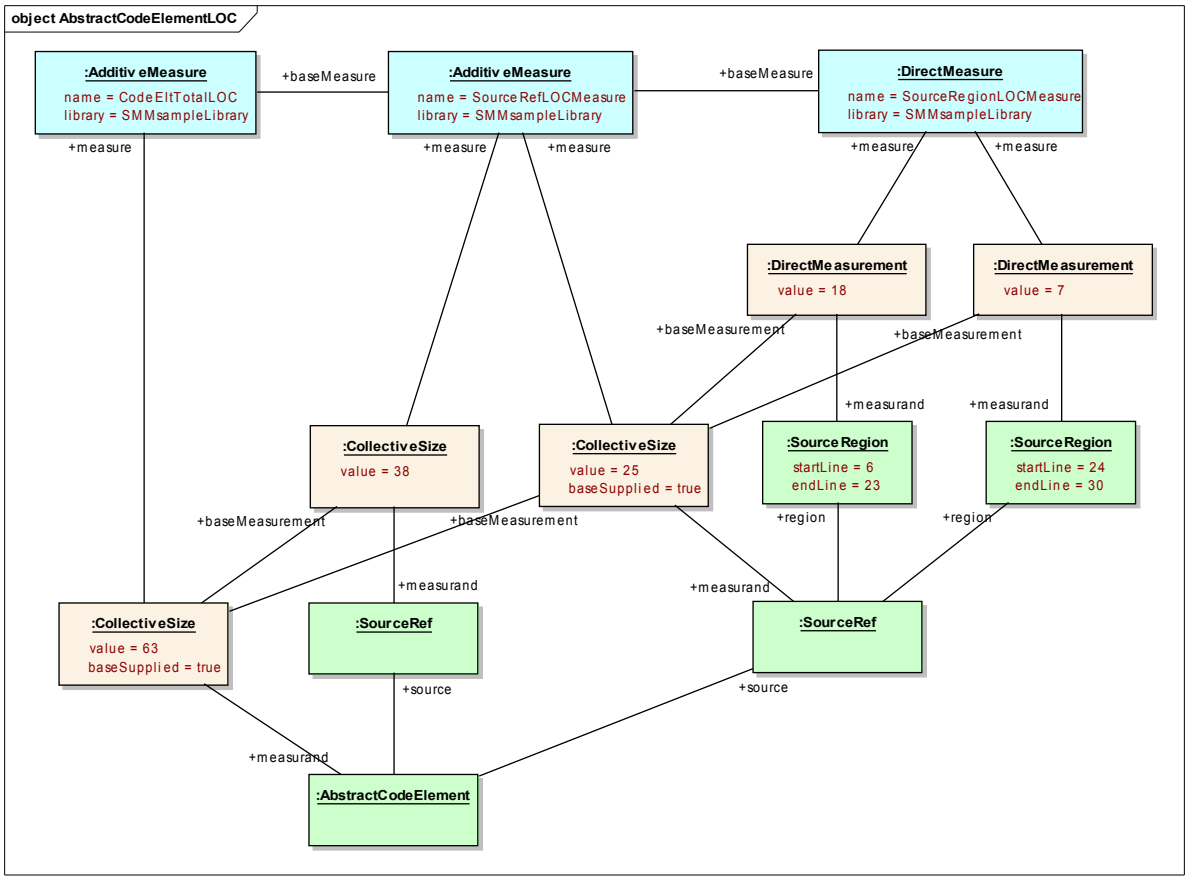


**Figure 27 - Lines of Code Demonstration**

## Lines of AbstractCodeElement

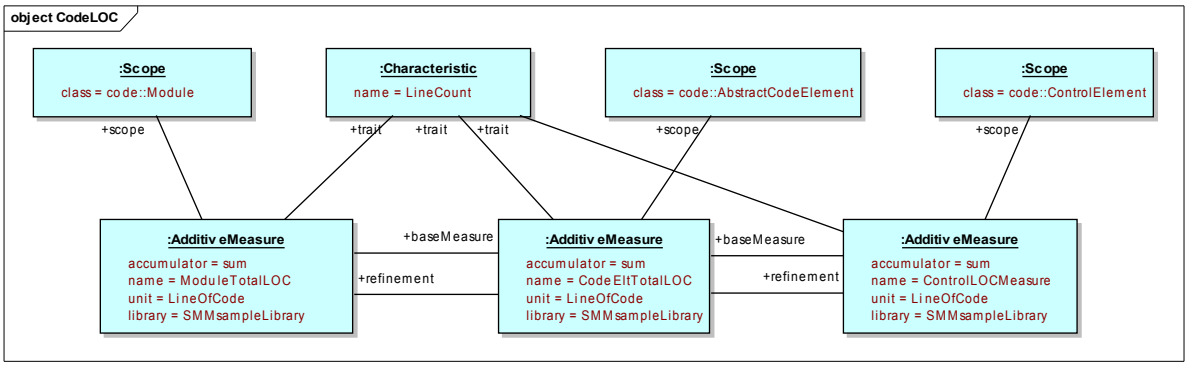


**Figure 28 - Additional Lines of Code Measures**

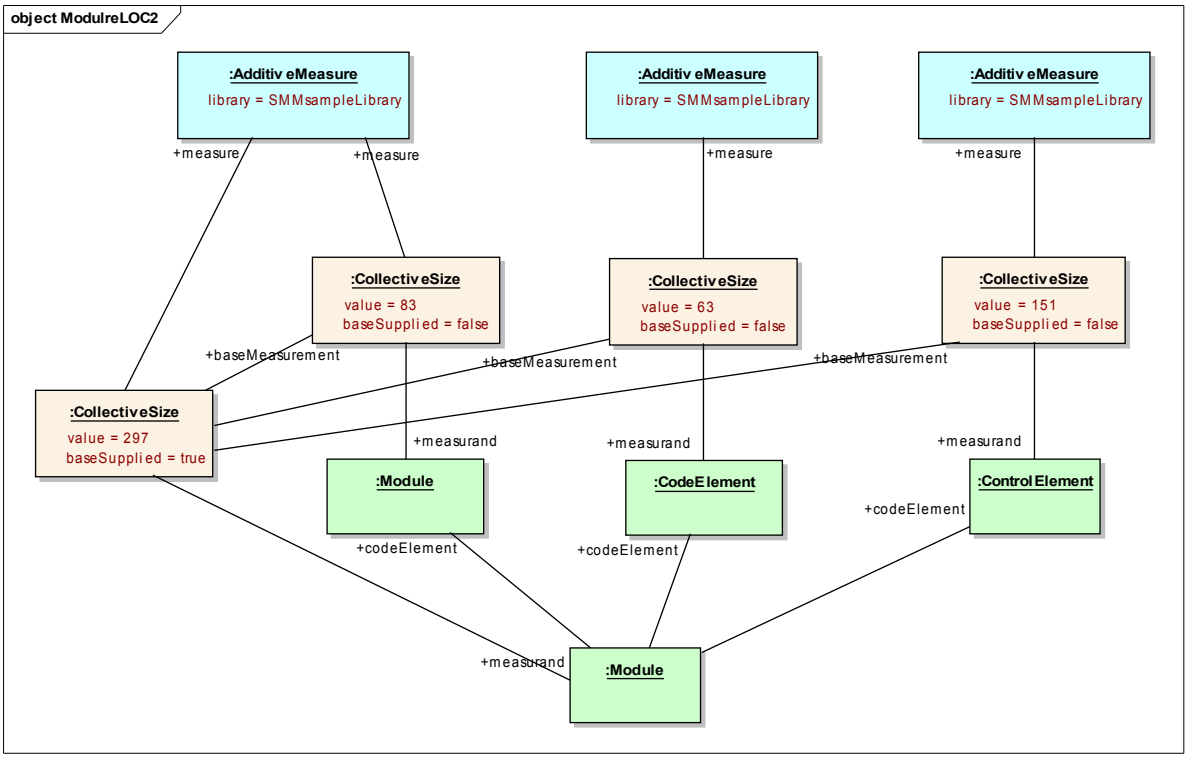


**Figure 29 - Module and Control Element LOC Demonstration**

### Refinement of Lines of ControlElement, CodeElement and Module

The source role for these elements is SourceRef. Determining the lines of code in each is an AdditiveMeasure where the base measure is the lines of SourceRef given above.

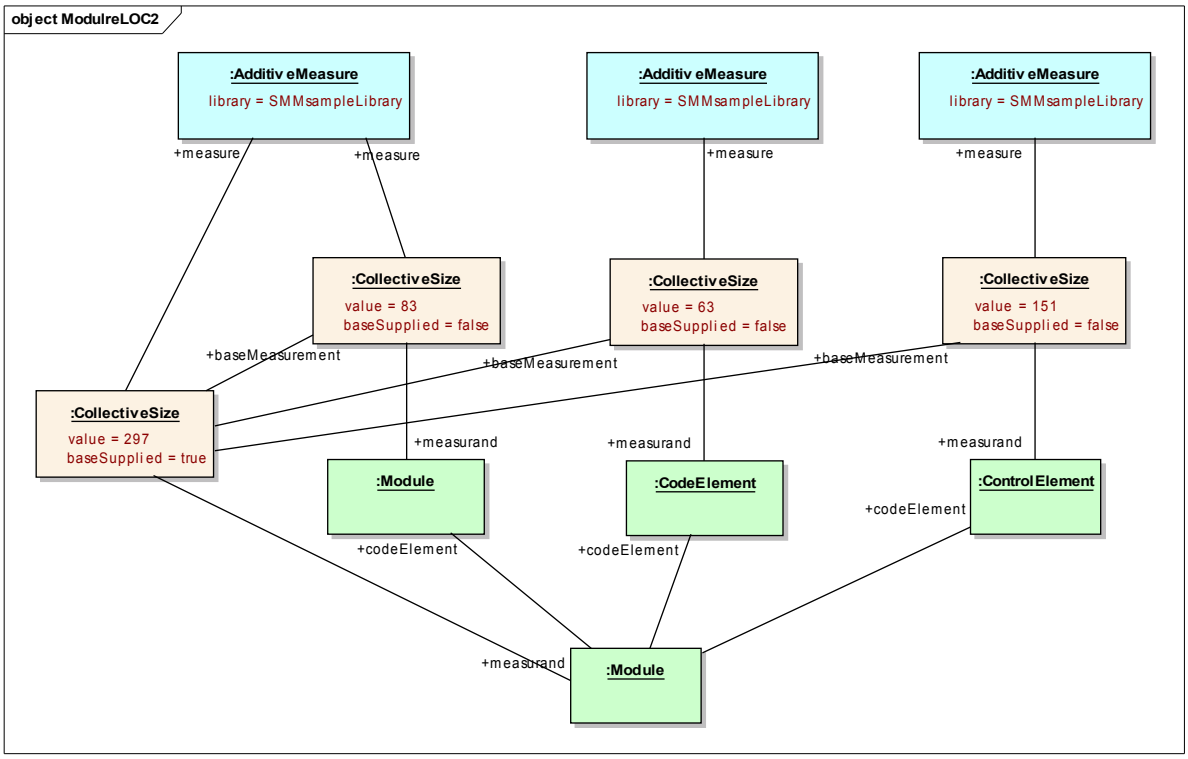


**Figure 30 - Module LOC Demonstration**

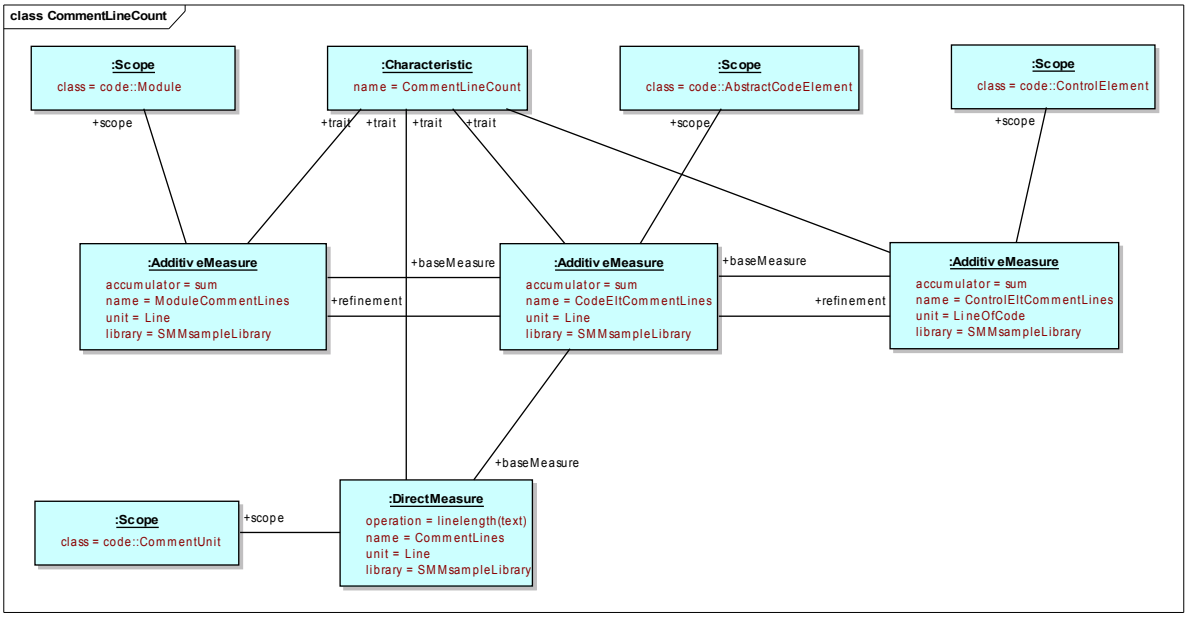


**Figure 31 - Comment Line Count**

### 18.1.5 Lines of Code for ASTM

The Abstract Syntax Tree Metamodel (ASTM) facilitates the interchange of programming language constructs parsed as abstract syntax trees. The Generic Abstract Tree Metamodel establishes a common core for modeling across a wide variety of programming languages. Each of these constructs may, of course, be measured by their lines of code.

GASTM does not directly model lines of source, code or otherwise. We will, consequently, make the same assumptions we made above for KDM. Blank lines are included and overlaps are ignored.

Figure 32 shows a fragment of the proposed ASTM covering the core syntax object, source location and source file. Figure 33 shows a possible SMM library entry to represent lines of code measure of GASTM syntax objects.

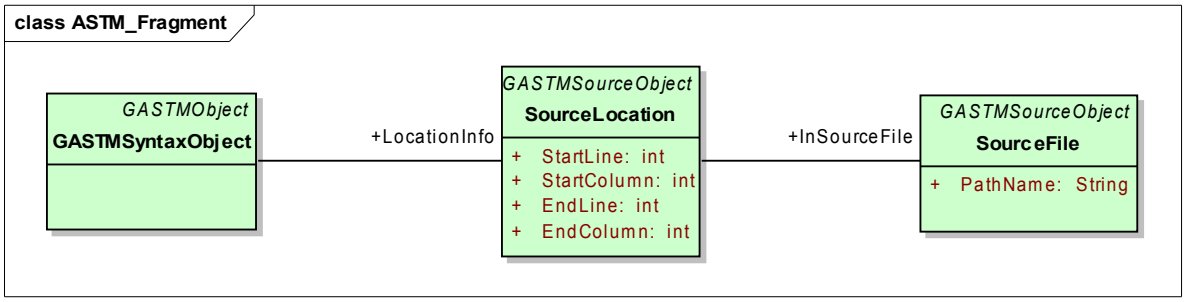


**Figure 32 - GASTM Fragment**

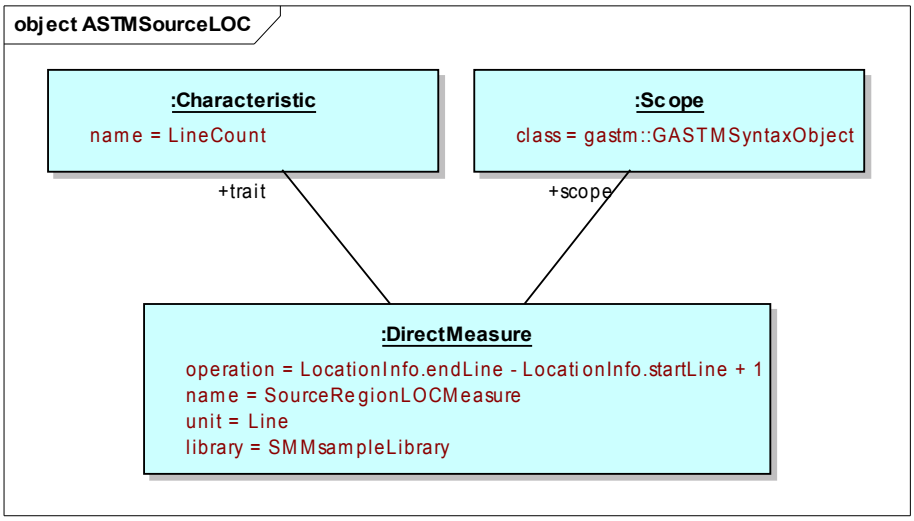


**Figure 33 - LOC Library Entry for GASTM**

## 18.2 McCabe

McCabe's cycolmatic complexity could modeled as a NamedMeasure. It is widely recognized. Alternatively, it could be a ReScaledMeasure from count of independent paths found by adding 2. Another representation would be as aReScaledMeasure from count of branching points found by adding 1. Each of these representations are present equivalent measures. We demonstrate below cyclomatic as a NamedMeasure and as a ReScaledMeasure from branching factor.

### 18.2.1 Branching Factor of ActionElements and Modules

Branching Factor is simply the difference between the number of nodes and edges in a module's control flow graph. KDM models the nodes as ActionElements, the edges as ControlFlow. Branching factor is then measured by subtracting the count of ControlFlow instances from the count of ActionElements.

\

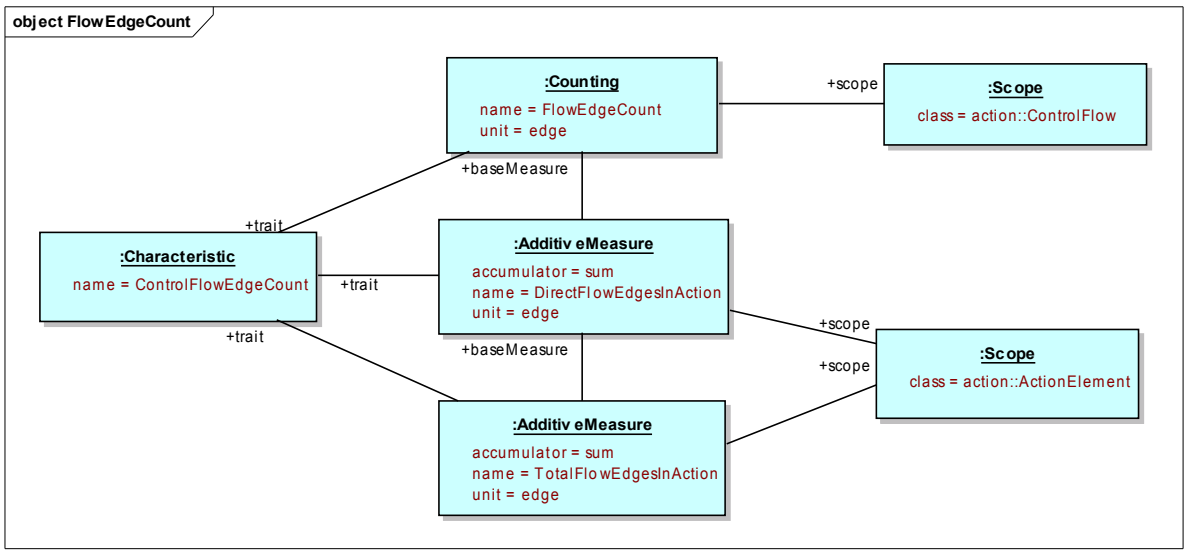


**Figure 34 - Control Flow Edge Count Library Entry**

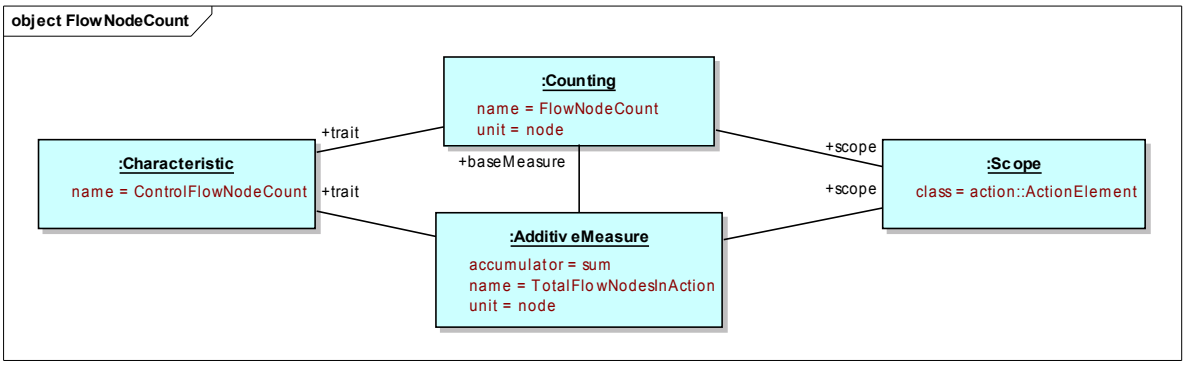


**Figure 35 - Control Flow Node Count Library Entry**

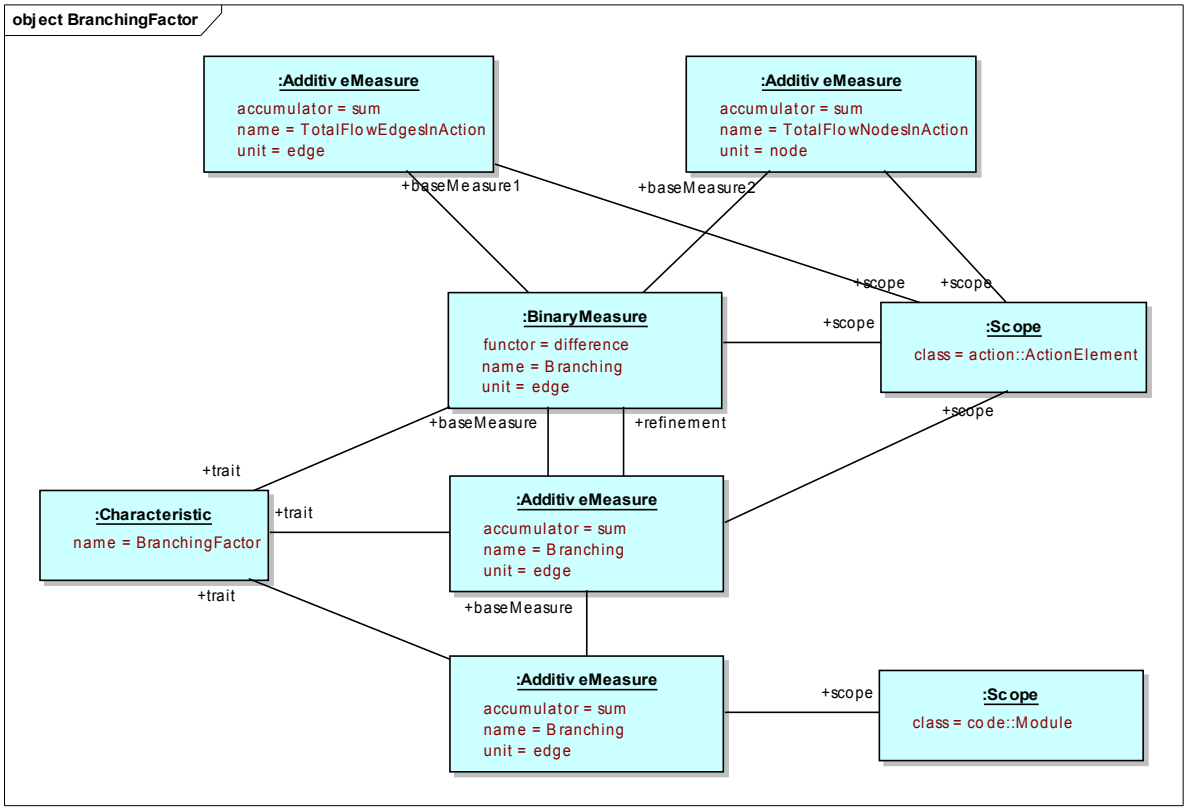


**Figure 36 - Control Flow Branching Factor Library Entry**

## 18.2.2 Cyclomatic Complexity of a Module[7]

> Cyclomatic complexity (CC) = $E - N + p$ where $E$ is the number of edges of the flow graph, $N$ is the number of nodes of the flow graph and $p$ is the number of connected components.

In this demonstration we assume that the control graph of each module is entirely connected. That is, $p$ is always 1. Cyclomatic is then simply the branching factor of a module plus one.

[7] See TPM 065 in Comsys Systems Redevelopment Methodology.

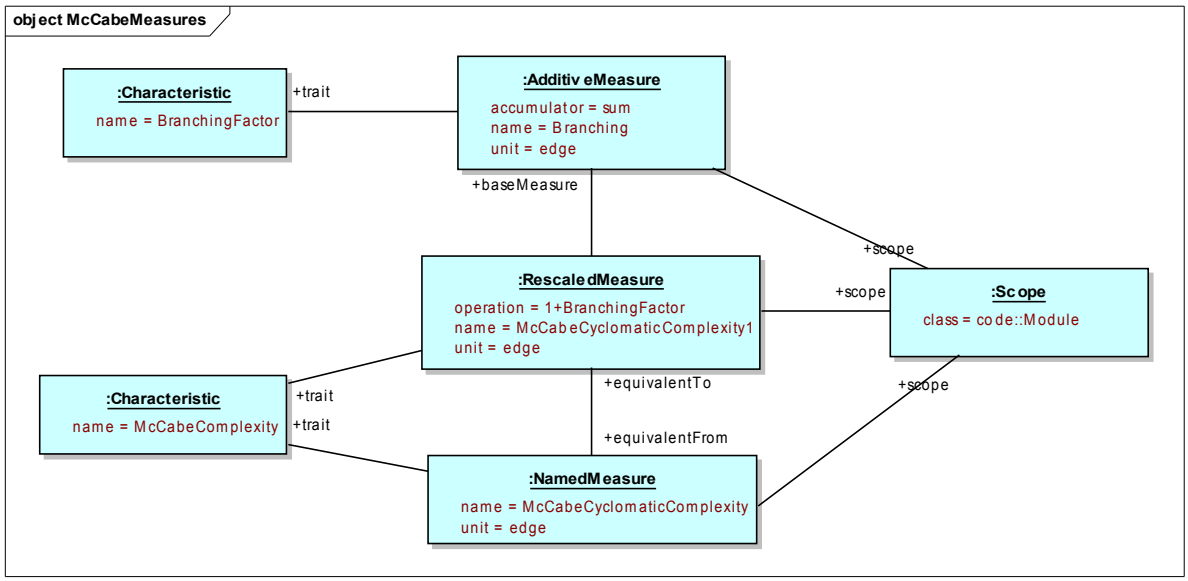


**Figure 37 - McCabe Cyclomatic Complexity Library Entry**

### 18.2.3 Extended Cyclomatic Complexity of a Module[8]

Extended cyclomatic is the count of predicates or atomic formula in the condition of branching statements. We demonstrate this count based upon ASTM modeling of an "if" statement. The condition of the "if" is an expression which can be navigated to find its atomic formulas.

### 18.2.4 Average Extended Cyclomatic Complexity of Modules in the System

Ratio of Additive ECC over Additive Counting of modules.

### 18.2.5 Counts of Operating Systems

The Application Management and System Monitoring for CMS Systems (ASMS) specification provides a PIM based upon commercial enterprise management called the DMTF Common Information Model (CIM). "CIM models a software or hardware system as a collection of component models connected via associations. A specific instance of a system is modeled as a collection of instances of component models and associations."[9]

We demonstrate the counting of operating systems installed and running on computer systems.

[8] See "An extension to the Cyclomatic measure of Program Complexity", Glenford Myers, SIGPLAN Notices, vol 12 no 10, 1977.
[9] See dtc/07-05-02.

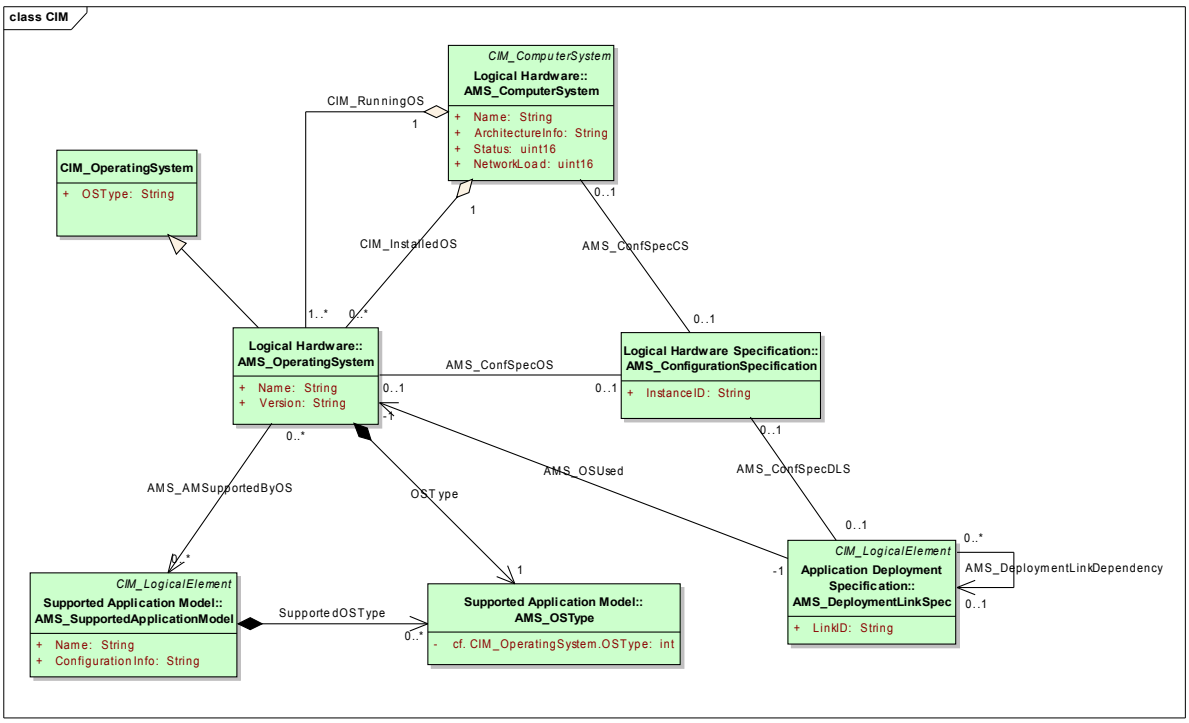


**Figure 38 - ASMS Fragment**

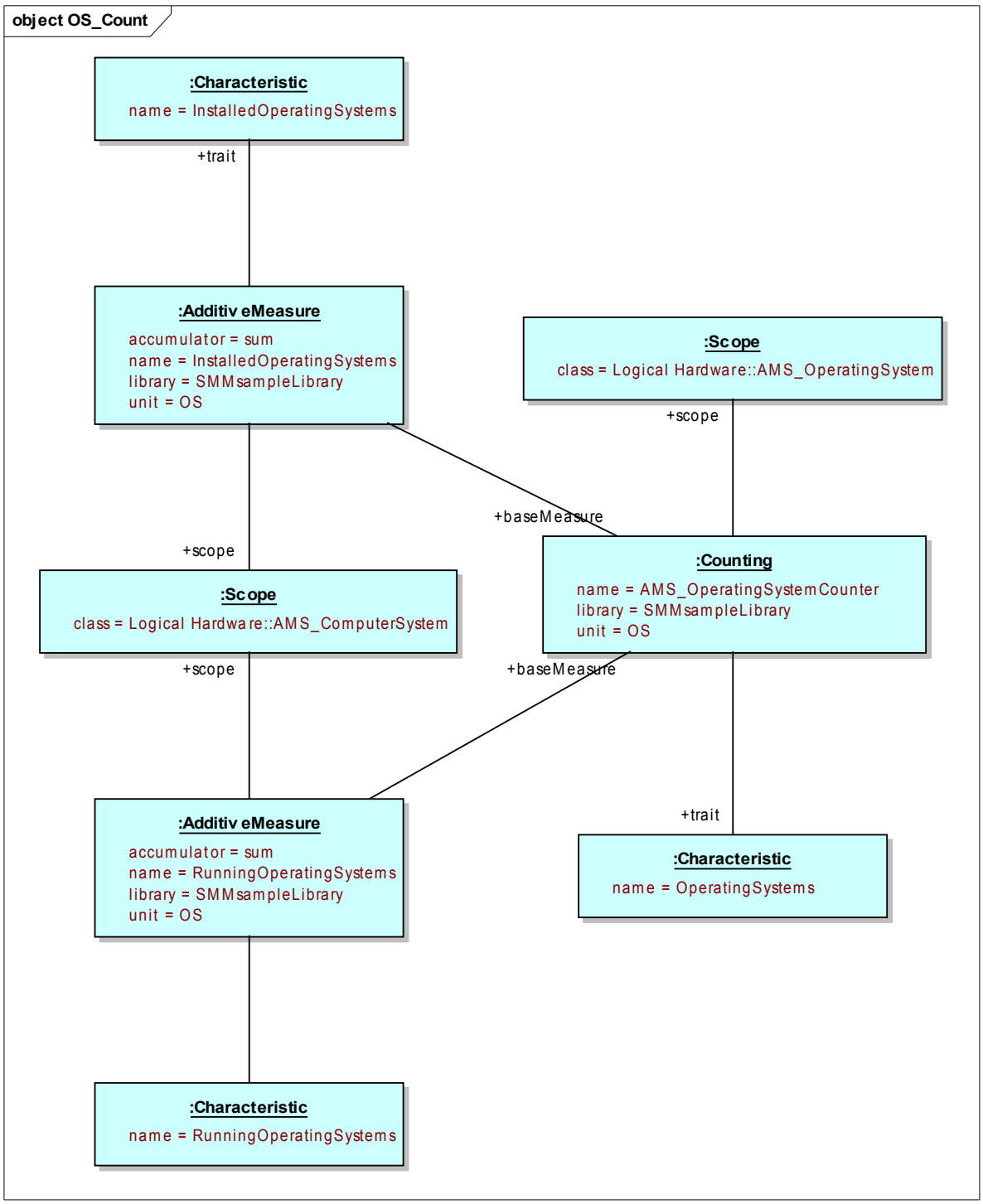


**Figure 39 - OS Counting Demonstration**

## 18.3 Halstead

### 18.3.1 Distinct Operator Count of a Module

$\acute{\eta}_1 \equiv$ A count of the number of distinct operators in a module.

Distinguishing operators invocations from calls to externally defined routines is not the type of higher level architectural concerns represented in the KDM. Counting the number of called, but not defined elements would get us close to this metric.

### 18.3.2 Distinct Operand Count of a Module

$\acute{\eta}_2 \equiv$ A count of the number of distinct operands in a module.

This is the data count shown above.

### 18.3.3 Operator Occurrence Count of a Module

$N_1 \equiv$ A count of the number of operator occurrences in a module.

This is a count of the calls to elements identified as operators.

### 18.3.4 Operand Occurrence Count of a Module

$N_2 \equiv$ A count of the number of operand occurrences in a module.

For KDM, this is a count StorableElements owned by ActionElements.

### 18.3.5 Halstead Length of a Module

$N=N_1+N_2$

This is an CollectiveMeasure where the aggregator is addition and the base measures are the occurrence counts given above.

### 18.3.6 Halstead Vocabulary of a Module

$\acute{\eta} = \acute{\eta}_1+\acute{\eta}_2$

This is an CollectiveMeasure where the aggregator is addition and the base measures are the counts given above.

### 18.3.7 Halstead Volume of a Module

$V=N \log_2 \acute{\eta}$

First $\log_2 \acute{\eta}$ is a ReScaledMeasure based upon the vocabulary metric given above. The volume is then an CollectiveMeasure of the length given above and the rescaled vocabulary with multiplication as the aggregator. The unit of measure for the rescaled vocabulary and for the volume is "required bits of representation."

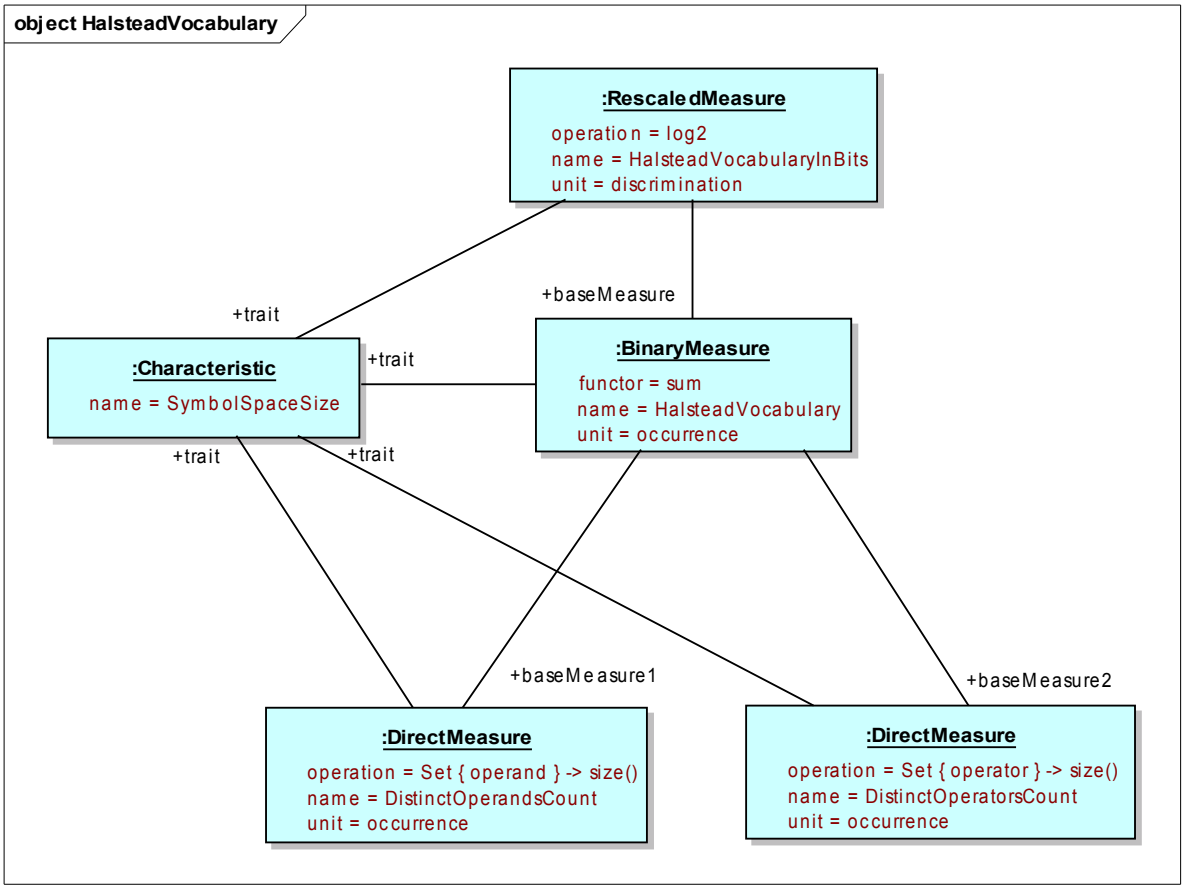


**Figure 40 - Halstead Vocabulary Library Entry**

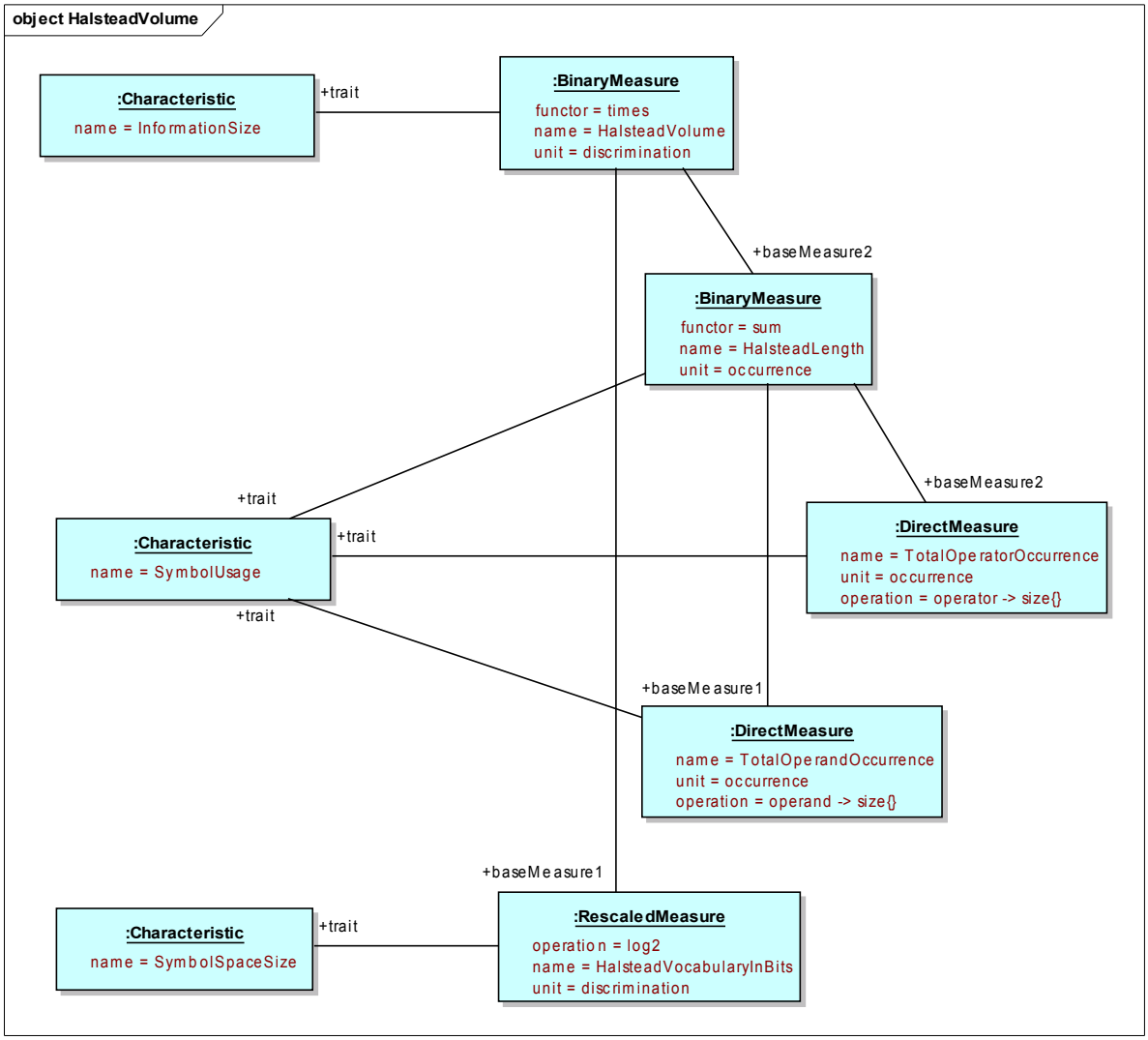


**Figure 41 - Halstead Volume Library Entry**

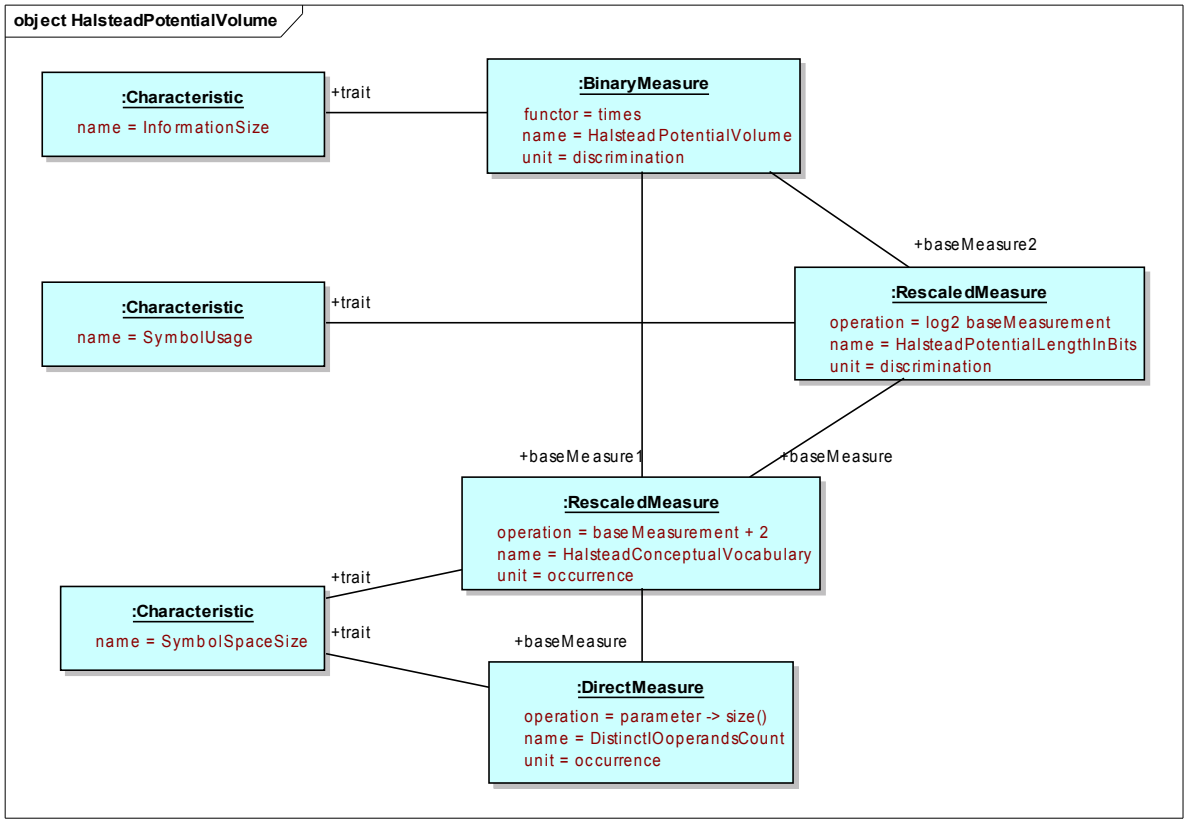


**Figure 42 - Halstead Potential Library Entry**

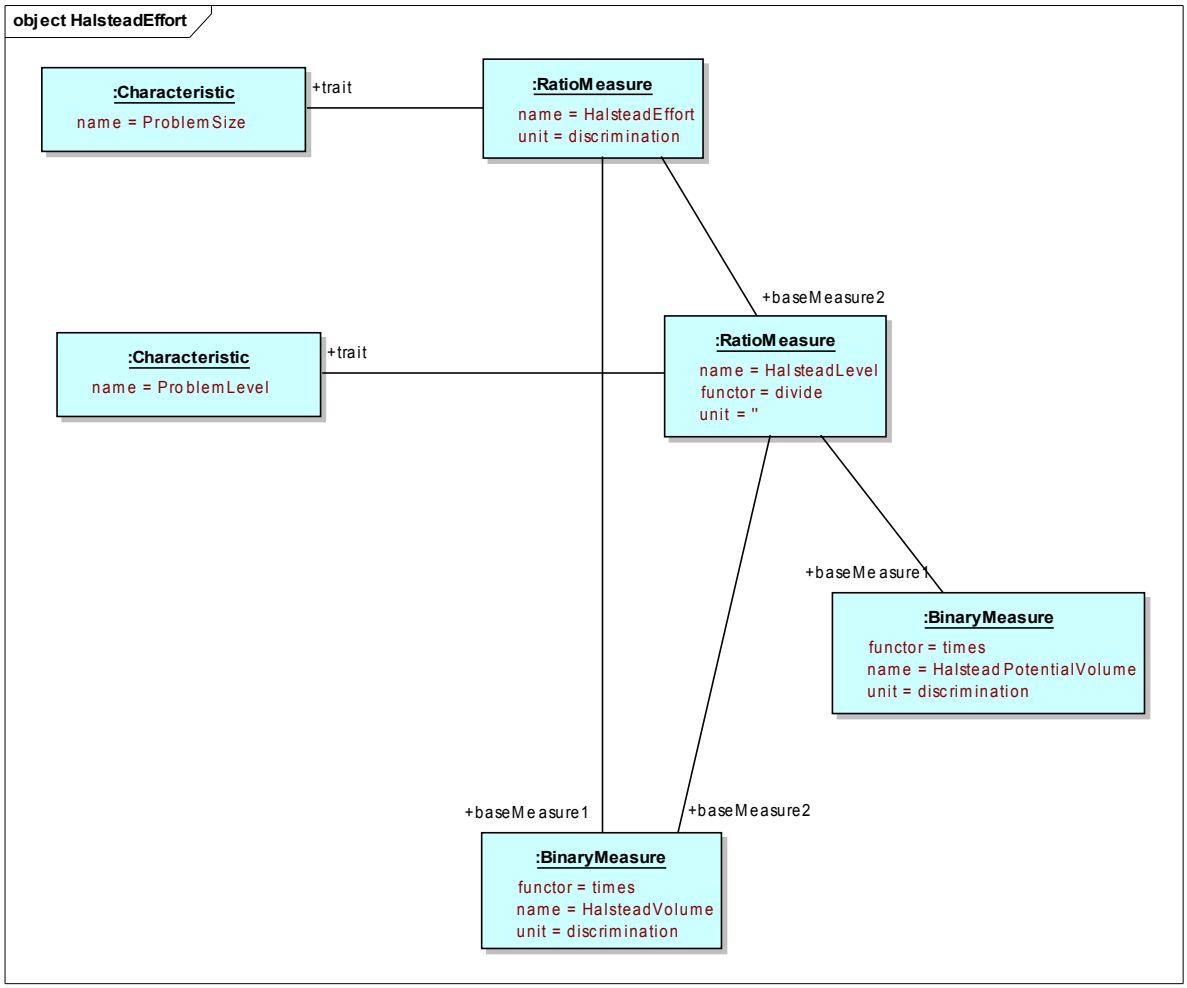


**Figure 43 - Halstead Effort Library Entry**

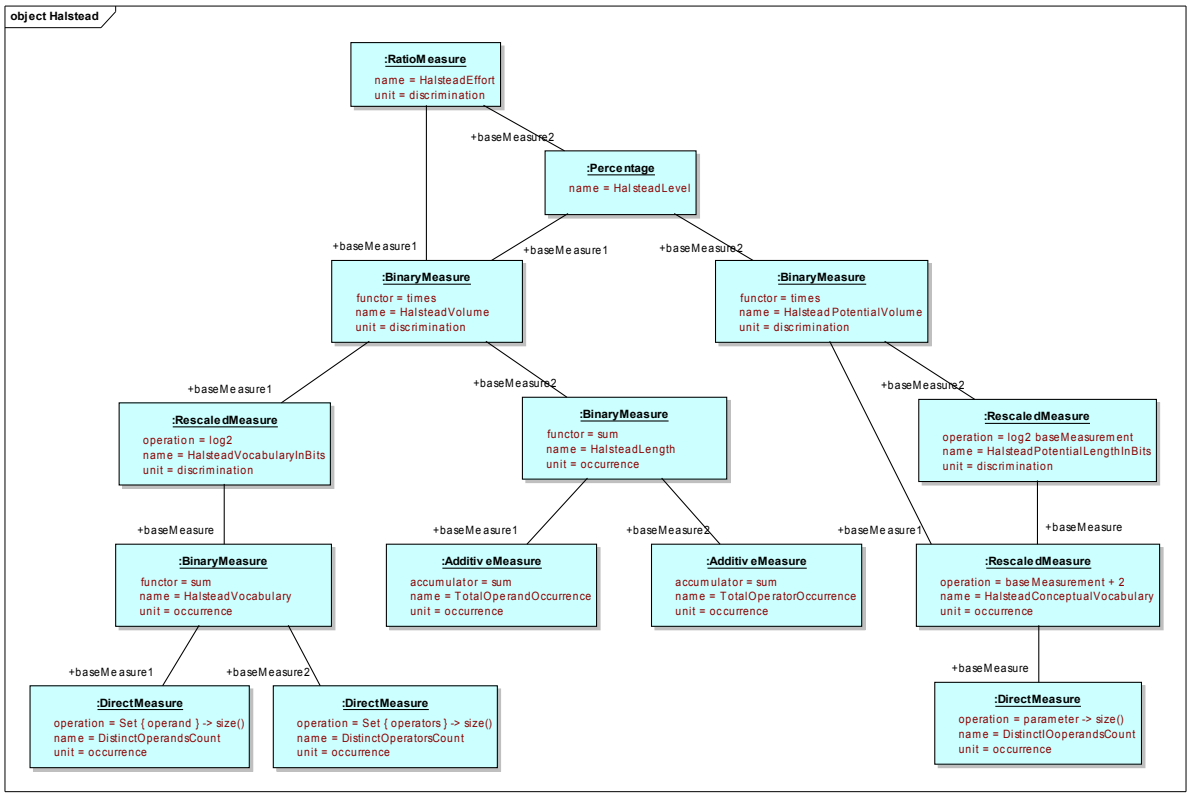


**Figure 44 - Halstead Measures Demonstration**

## 18.4 Software Engineering Institute (SEI) Maintainability Index

$$171 - 5.2(\ln(aveV)) - 0.23(aveV(g')) - 16.2(\ln(aveLOC)) + 50(\sin(\sqrt{2.4(perCM)}))$$

Each of the averages are RatioMeasures of their respective metric (V for Halstead volume, V(g') for extended Cyclomatic complexity and LOC of line of code) for modules over the count of modules. perCM, the percentage of comments in a module, is a PercentageMeasure of line count of comments over the total line count of a module.

Each resulting metric is rescaled to share the same unit of measure, namely maintainability index points.

| | |
|---|---|
| aveV rescaled | 50 – 5.2(ln(aveV) |
| aveV(g') rescaled | 50 – 0.23(aveV(g')) |
| aveLOC rescaled | 21 – ln(aveLOC) |
| perCM rescaled | 50(sin (sqrt(2.4(perCM)))) |

The SEI index is then an CollectiveMeasure for a module of the above four rescalings with addition as the aggregator.

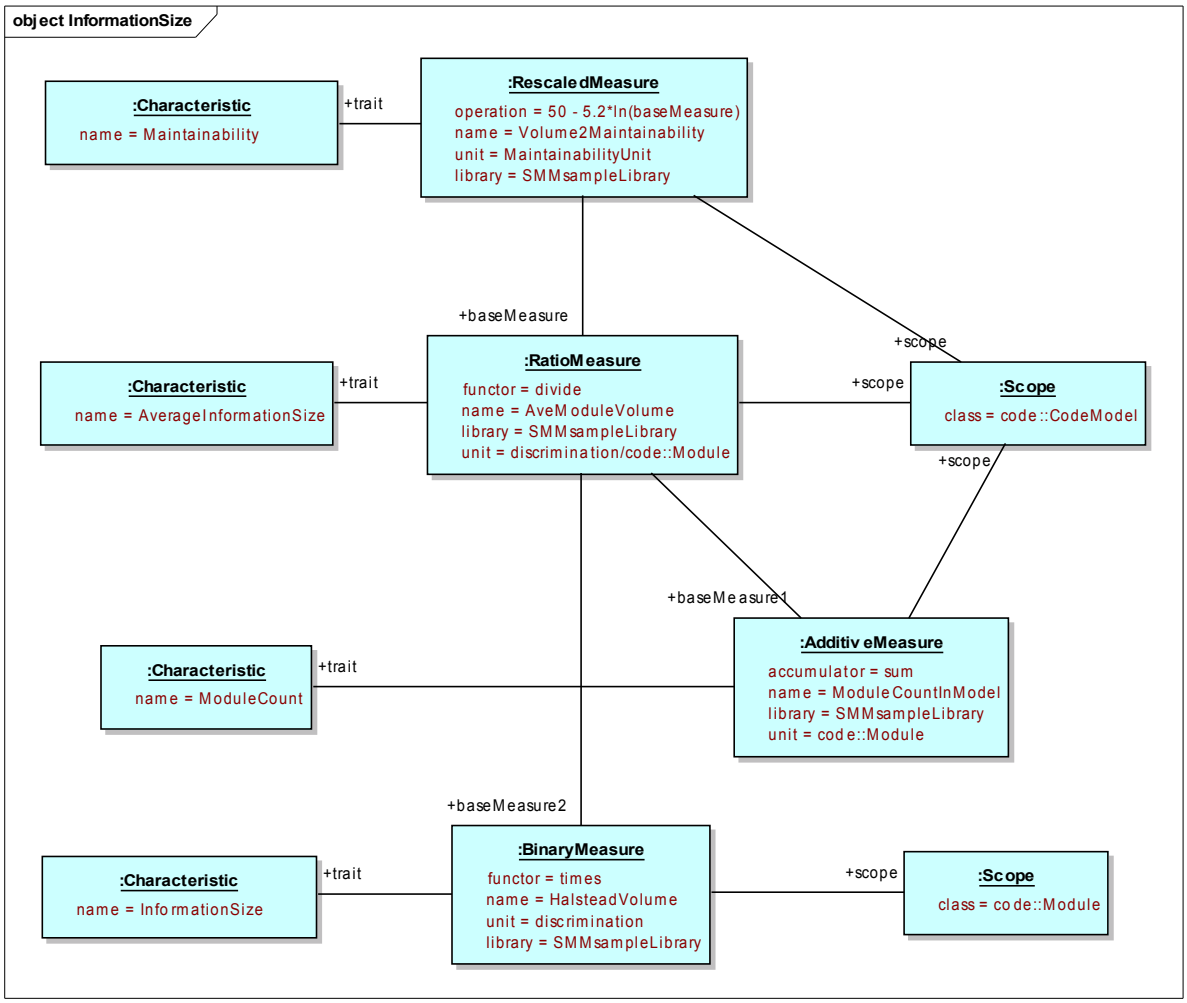


**Figure 45 - Conversion of Information Size to Maintainability**

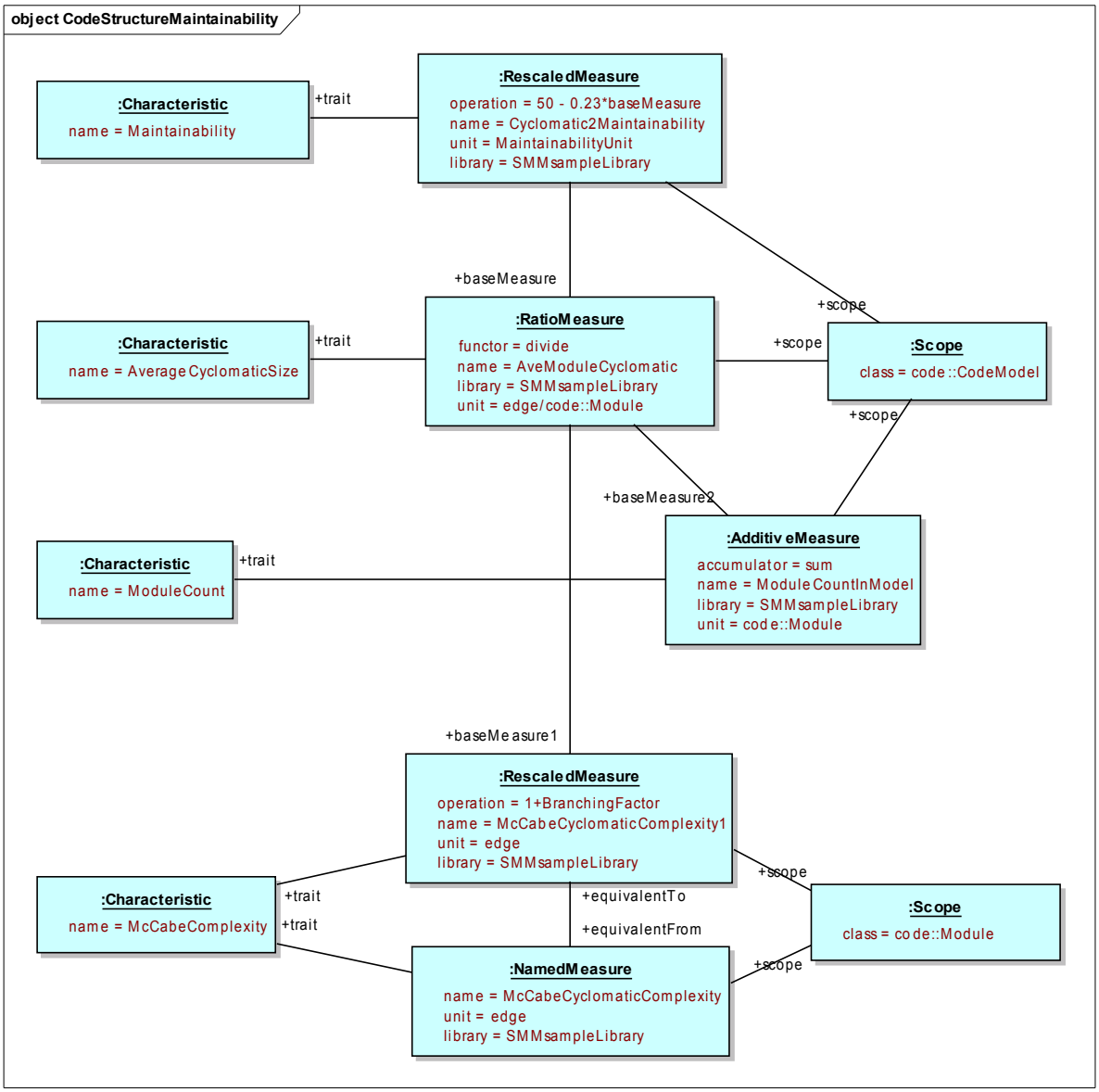


**Figure 46 - Conversion of McCabe Cyclomatic to Maintainability**

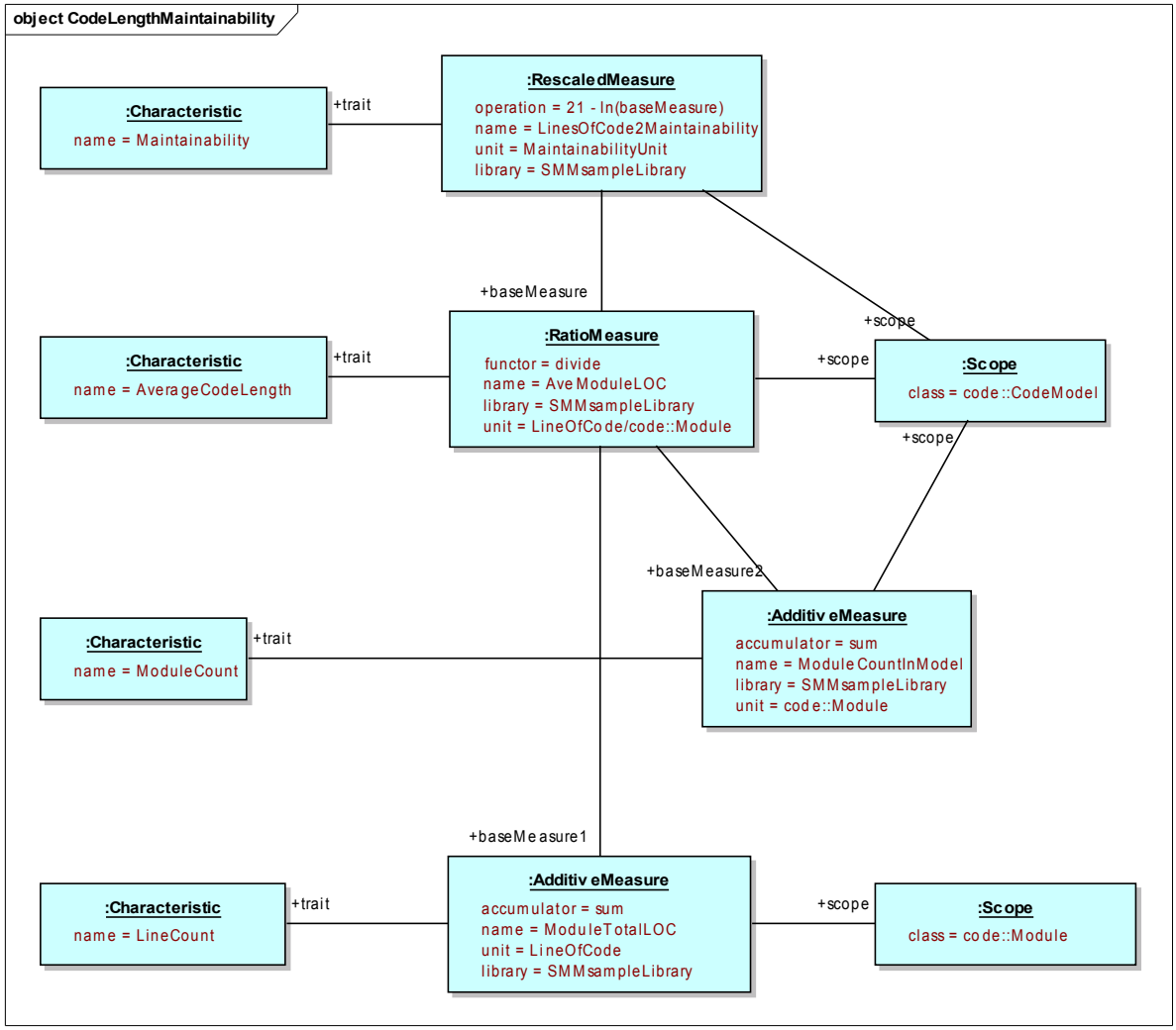


**Figure 47 - Conversion of LOC to Maintainability**

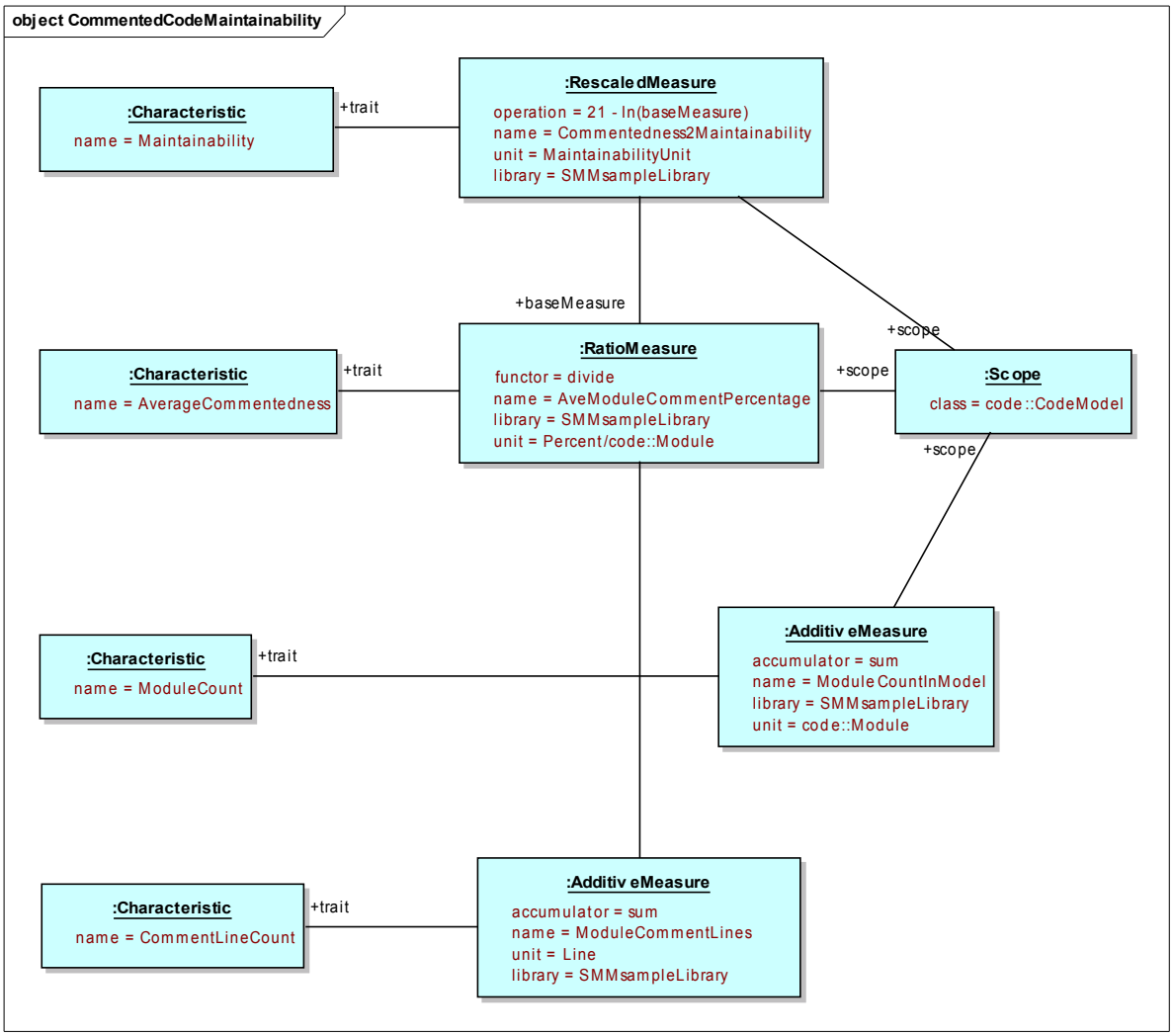


**Figure 48 - Conversion of Comment Count to Maintainability**

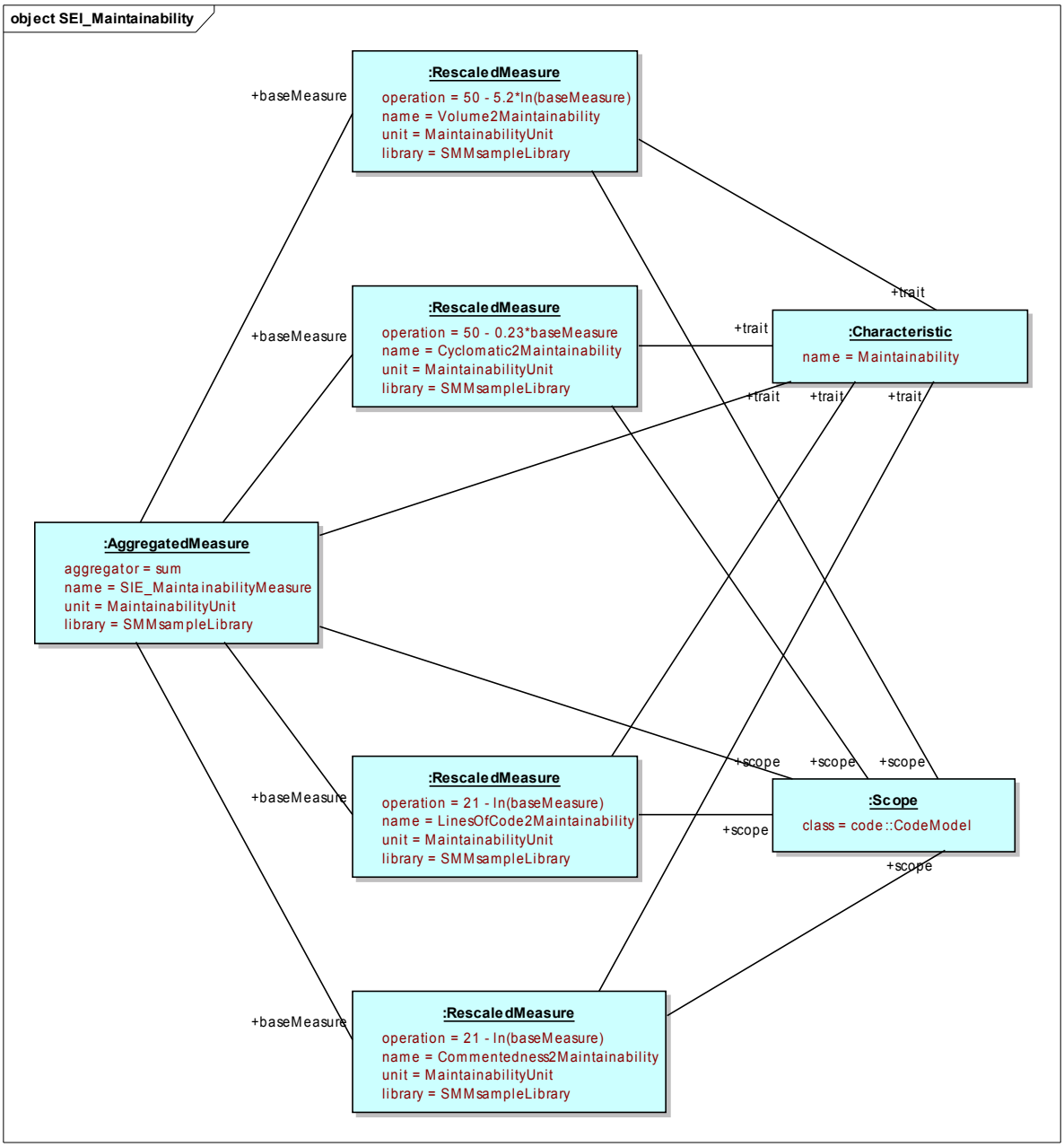


**Figure 49 - SEI Maintainability Demonstration**

# 18.5 Qualitative Example

## 18.5.1 Non-standard language usage score

Non-standard languages are defined by an organization's accepted technology standards. Assign the following scores where a 1 or 2 is low, a 3 is medium and a 5 is high:

1. 2GL or unacceptable 4GL assign 1 or 2
2. Acceptable 3GL or 4GL assign 3 or 4
3. Ideal strategic language assign 5

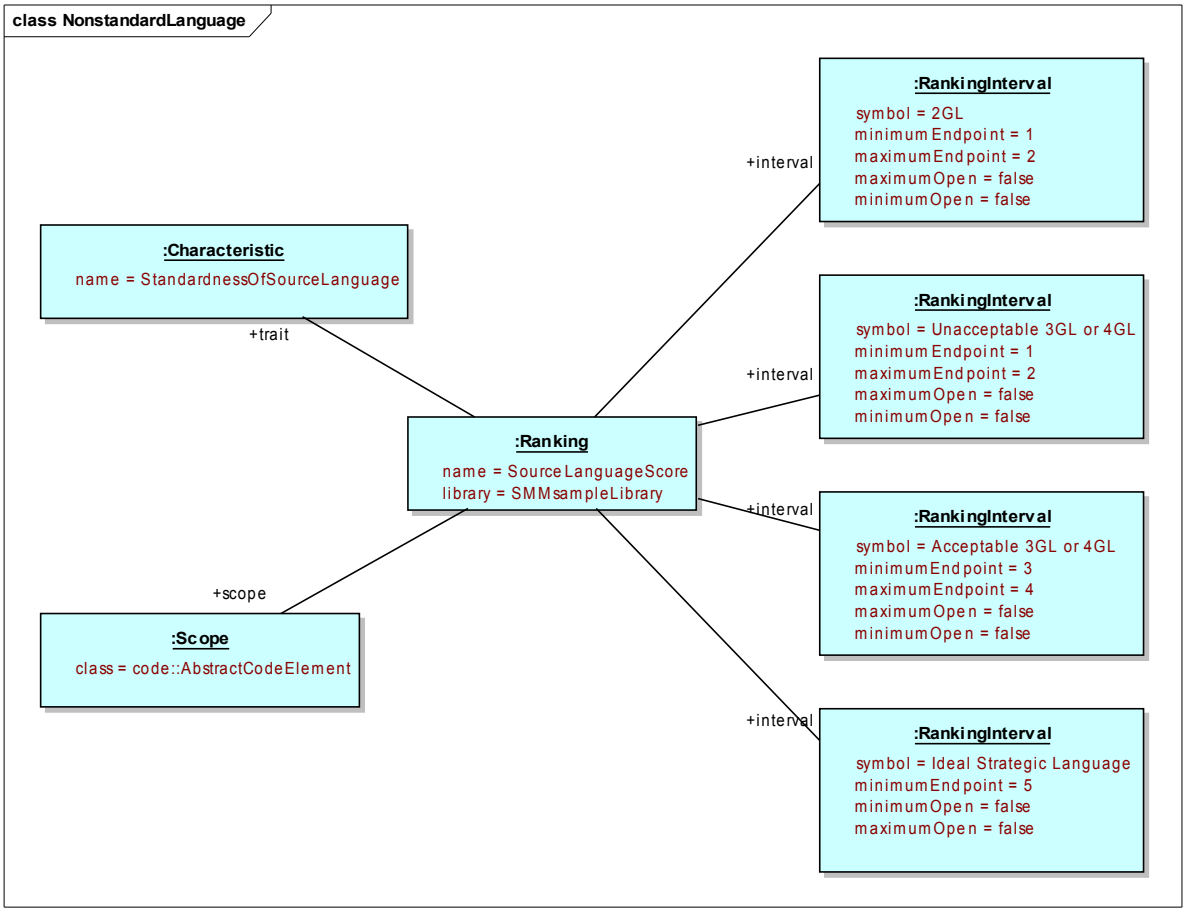


**Figure 50 - Qualitative Measure Demonstration**

# 19 Library of Categories

SMM does not establish a standard set of measurement categories which presents an organization of measures applicable to every software environment, every stage of software life cycle, every software platform, software language or every software engineering activity. SMM minimally establishes a demonstration library of metric categories. The library does not assert that the given categories are standards.These metric categories reflect a high-level summary of industry metrics that support some software engineering processes.

## 19.1 Environmental Metrics

number of screens, programs, lines of code, etc.

## 19.2 Data Definition Metrics

number of data groups, overlapping data groups, unused data elements, etc.

## 19.3 Program Process Metrics

Halstead, McCabe, etc.

## 19.4 Architecture Metrics

average call nesting level, deepest call nesting level, etc.

## 19.5 Functional Metrics

functions defined in system, business data as a percentage of all data, functions in current system that map to functions in target architecture, etc.

## 19.6 Quality / Reliability Metrics

failures per day, meantime to failure, meantime to repair, etc.

## 19.7 Performance Metrics

average batch window clock time, average online response time, etc.

## 19.8 Security / Vulnerability

breaches per day, vulnerability points, etc.