Counting Rules for Software Size Measure v2

This document is used as part of a research project to determine whether a new Software Size Measure [1] will produce better estimates of work effort when compared to existing effort prediction models.

July 10, 2012

David P. Voorhees
Le Moyne College
1419 Salt Springs Road
Syracuse, New York 13214

Contents

1. Introduction

This document describes rules associated with the software size measure version 2 (SSMv2) [1]. These rules describe how to identify software artifacts, entities, and attributes, and then how to count software attributes.

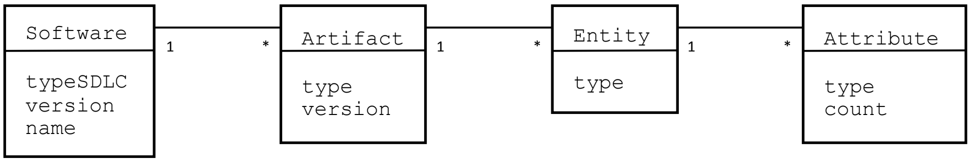
1.1 Glossary

The following terms are defined and used within this document.

Term	Definition
software artifact	A product that is created/updated as part of a software development project and results from implementing a software development process. For purposes of the SSMv2, a product is a *software artifact* only if (1) it meets the above definition and (2) the work effort in producing the product is known (recorded).
software attribute	A characteristic or aspect of a product or sub-product that is created/updated as part of a software development project, results from implementing a software development process, and is contained within a *software entity*. For purposes of the SSMv2, a characteristic or aspect is a *software attribute* only if (1) it meets the above definition and (2) the instances of the characteristic or aspect can be counted using the counting rules described within this document.
software entity	A product or sub-product that is created/updated as part of a software development project, results from implementing a software development process, and is contained within a *software artifact*. For purposes of the SSMv2, a product or sub-product is a *software entity* only if it meets the above definition. Note that the work effort in producing the *software entity* may be known (recorded). When the work effort is known, it shall be used in SSMv2.

1.2 Structure of Software Knowledge

The artifact, entity, and attribute terms represent a hierarchy of knowledge as expressed in a software product. The logical data model below illustrates the relationships between these terms.



1.3 Structure of Document

Section 2 describes the rules for identifying software artifacts, software entities, and software attributes. Section 3 then describes the rules for counting software attributes. Appendices A and B provide more details on modeling techniques, appendix C provides a document history, and appendix D lists references.

2. Identify Software Artifacts, Entities, & Attributes

Before we can count instances of software attributes, rules must be established for identifying software artifacts, software entities, and software attributes.

2.1 Rules for Identifying Software Artifacts

A software artifact is a software deliverable that is created and/or updated as part of a software development project. It represents a set of related knowledge packaged into a software product or deliverable and typically results from implementing a software development phase or activity. In order for a software artifact to help describe the size of software, and thus be used to estimate effort, the work effort expended to produce the artifact must be known (i.e., recorded). Note that the work effort is the cumulative effort of all tasks, regardless of role(s) performing each task, that are directly associated with gathering and storing knowledge contained within the artifact.

A *software artifact type* identifies a type of software knowledge. The types of software knowledge used with the SSMv2 are as follows.

- Feasibility and Planning e.g., feasibility study or project plan.
- Requirements Analysis e.g., requirements document.
- Design e.g., high-level design document, detailed design document.
- Code e.g., program code, html/xml code.
- Test e.g., test plan, test cases.
- Implement e.g., implementation/installation plan.

A software artifact instance may contain knowledge from one or more *software artifact types*. For example, one artifact instance may contain both Requirements Analysis and Design knowledge, or may contain both Code and Test knowledge.

2.2 Rules for Identifying Software Entities

A software entity is a subset of a software artifact that is created and/or updated as part of a software development project. (Note that one software artifact may contain one or more software entities.) It represents a subset of all the knowledge stored in the software artifact and typically results from using methods and techniques associated with a software development phase or activity. A software entity may include the work effort expended to produce the entity, but this is not required.

Knowledge is represented in a software entity using diagrams, text, and/or code.

2.2.1 Identifying Diagram-based Software Entities

When a software artifact contains one or more diagrams, each diagram is identified as a software entity and is measured (i.e., its attributes counted) as follows:

1. When a reference exists which describes the notation and semantics of the underlying modeling technique, use this reference to identify and count the software attributes. (See Appendix B for a description of some structured design modeling techniques.)
2. Otherwise, when the diagram itself includes a description of the notation and semantics being used, use this information to identify and count the software attributes.
3. Otherwise, the diagram shall be measured (counted) as a generic diagram with only one (generic) attribute instance.

2.2.2 Identifying Text-based Natural Language Software Entities

When a software artifact contains text, software entities will be identified and measured (i.e., its attributes counted) as follows:

1. When a software artifact contains document sections, each section containing text-based knowledge shall be identified as a text-based entity instance.
2. Otherwise, the entire software artifact instance is identified as one text-based entity instance.

2.2.3 Identifying Code-based Programming Language Software Entities

When a software artifact contains program code, the software entities will be identified and measured (i.e., its attributes counted) based on the type of programming language being used, as follows:

- For object-oriented program code, the software entity instances are:
  - Class definition
  - Files
  - Method definition
  - Source code
  - Tag code
- For structured (imperative) program code, the software entity instances are:
  - Files
  - Function definition
  - Source code
  - Tag code

2.3 Rules for Identifying Software Attributes

A software attribute is a characteristic or aspect of a product or sub-product that is created and/or updated as part of a software development project. It represents a subset of all knowledge stored in the entity and typically results from implementing a software development activity or task. A software attribute does not have work effort associated with it.

Rules for identifying software attributes are based on whether the knowledge is represented using diagrams, text, and/or program code.

2.3.1 Identifying Diagram-based Software Attributes

When a software entity instance is a diagram, attribute instances shall be identified as follows:

- When the semantics of the modeling technique defines each notation as conveying distinct knowledge, then each instance of this notation represents distinct knowledge. An example of this is a relationship between two entities in an entity-relationship diagram.
- When the notation of the modeling technique includes a label (text) to help identify distinct instances, then each instance with a unique label represents distinct knowledge. An example of this is the data flow diagram process notation, which includes a name to distinguish one process from another. The term *named attribute instance* refers to this labeling of diagram notation.

2.3.2 Identifying Text-based Natural Language Software Attributes

When a software entity instance is text-based, attribute instances of English text shall be identified using the paragraph structure. That is, a paragraph represents the only type of text-based software attribute that will be identified and counted.

2.3.3 Identifying Code-based Programming Language Software Attributes

Identifying code-based programming language software attributes is described in section *3.1.3 Counting Code-based Software Attributes*.

3. Count Software Attributes

Once a software attribute has been identified within a software entity, it shall be counted.

3.1 Rules for Counting Software Attributes

Rules for counting software attributes are based on whether the knowledge is represented using diagrams, text, and/or program code.

3.1.1 Counting Diagram-based Software Attributes

When a software entity instance is a diagram, attribute instances shall be measured (counted) as follows:

- When a notation (symbol) is not labeled (i.e., it is not a *named attribute instance*), each instance of this unlabeled notation shall be counted.
- When a notation (symbol) is labeled (i.e., it is a *named attribute instance*), each instance of this labeled notation shall be counted as follows:
  - When a *named attribute instance* occurs **more than once in one diagram**, the duplicated notation is counted only when the duplication adds knowledge to the diagram. When the duplicated notation is done for purely aesthetic reasons (e.g., same data store appears twice on a DFD to avoid crisscrossing of lines) then the duplication shall not be counted.
  - When a *named attribute instance* occurs **more than once within a collection of like-diagrams**, the duplicated notation is counted only when the duplication adds knowledge to the entity or artifact. For example, a collection of sequence diagrams likely includes the same object on many diagrams. This duplication adds new knowledge to the software entity and shall be counted.

Appendix A contains tables that identify software modeling techniques that may be found within a software entity. The table in this appendix identifies the rules for how to count a duplicate *named attribute instance*.

Appendix B shows the notations associated with some of the modeling technique listed in Appendix A.

3.1.2 Counting Text-based Software Attributes

When a software artifact contains English text, each paragraph shall be counted based on the software entity identification rules stated in section 2.2.2. When text is written as a list of bullets, then the following shall be applied:

- When each bullet contains a single sentence or phrase, the entire list of bullets shall be counted as one paragraph.
- When each bullet contains multiple sentences or phrases, each bullet shall be counted as one paragraph.

For example, the two bullets in this section would be counted as one paragraph. The two sub-bullets in the section 3.1.1 (which describe how to count *named attribute instances*) would be counted as two paragraphs.

3.1.3 Counting Code-based Software Attributes

When a software entity is program code, each attribute is counted based on the specific programming language being used.

For all types of programming languages:

Entity	Attribute	Counting Rule
Files	source	Number of files containing program source code.
	tag	Number of files containing tag-based (e.g., html) source code.
Source code	all statements	Number of statements in the source code files. When a programming language has a statement delimiter, this is used to count the statements.
	iteration	Number of iteration statements in the source code files.
	method/function calls	Number of methods or functions invoked in the source code files. Each method/function invocation is counted regardless of how many times it is called.
	selection	Number of selection statements in the source code files.
Tag code	all tags	Number of tags in the source code files.
	unique tag	Number of unique tags in the source code files.

For object-oriented languages:

Entity	Attribute	Counting Rule
Method definition	private	Number of private method definitions in the source code files.
	public	Number of public method definitions in the source code files.
	other	Number of method definitions that are not private and not public in the source code files.
Class definition	private	Number of private class definitions in the source code files.
	public	Number of public class definitions in the source code files.
	other	Number of class definitions that are not private and not public in the source code files.

For imperative languages:

Entity	Attribute	Counting Rule
Function definition	public	Number of public function definitions in the source code files.
	other	Number of non-public function definitions in the source code files.

Appendix A – Modeling Technique Duplicate Counting Rules

The table below identifies modeling techniques that may be found in a software artifact or software entity. Each modeling technique lists its software attributes and identifies how duplicate *named attribute instances* shall be counted.

When a duplicate *named attribute instance* is found:

- On **one diagram**, the duplicate instances are counted either **once** or ***each*** duplicate instance is counted.
- On a **set of like-diagrams**, the duplicate instances are counted either ***once*** or ***each*** duplicate instance is counted.

		Duplicate Named Attribute Instances	
Modeling Technique	**Attribute**	**One Diagram**	**Set of Like-diagrams**
Data flow diagram	data flow	each	each
	data store	once	each
	external entity	once	once
	process (operation)	once	each
Data model diagram	attribute	each	each
	entity	once	each
	relationship	each	each
Entity relationship diagram	entity	once	each
	relationship	each	each
Event table	action	each	each
	event	once	each
	state/condition/entity	each	each
Flowchart	decision	each	each
	flow	each	each
	input/output	each	each
	processing	each	each
	subroutine call	each	each
	terminal	each	each
Nassi-Schneiderman chart	iteration	each	each
	selection	each	each
	sequence	each	each
SADT activity diagram (ICOM)	activity	once	each
	control	each	each
	input	each	each
	mechanism	each	each
	output	each	each
State transition diagram	state	once	each
	state action	once	each
	transition	each	each
	transition action	each	each
	transition guard condition	each	each
State transition table	action	once	each
	state	once	each
	transition (event)	each	each
Structure chart	information flow	each	each
	invocation	each	each

Modeling Technique	Attribute	Duplicate Named Attribute Instances	
		One Diagram	Set of Like-diagrams
	predefined procedure	once	once
	procedure	once	each
Structure diagram (Jackson)	connection	each	each
	iteration	each	each
	selection	each	each
	sequence	each	each
UML Class diagram	aggregation ———◇	each	each
	association ———▶	each	each
	attribute	each	each
	class	once	each
	composition ———◆	each	each
	dependency - - - - - - -▶	each	each
	generalization ———▷	each	each
	interface (abstract class)	once	each
	operation	each	each
	package	once	each
	realization ———	each	each
UML Collaboration diagram	condition [in brackets]	each	each
	iteration * or for each	each	each
	message ———▶	each	each
	object	once	each
	return (assignment op)	each	each
UML Component diagram	component	once	each
	dependency - - - - - - -▶	each	each
	generalization ———▷	each	each
	non-OO component	once	each
	interface	once	each
UML Package diagram	component	once	each
	dependency - - - - - - -▶	each	each
	generalization ———▷	each	each
	non-OO component	once	each
	interface	once	each
UML Deployment diagram	component	once	each
	dependency - - - - - - -▶	each	each
	interface	once	each
	node	once	each
UML Object diagram	object	once	each
	link	each	each
UML Package diagram	dependency - - - - - - -▶	each	each
	generalization ———▷	each	each
	package	once	each
UML Sequence diagram	condition [in brackets]	each	each
	iteration * or for each	each	each
	message ———▶	each	each
	object	once	each
	return ◀- - - - - -	each	each

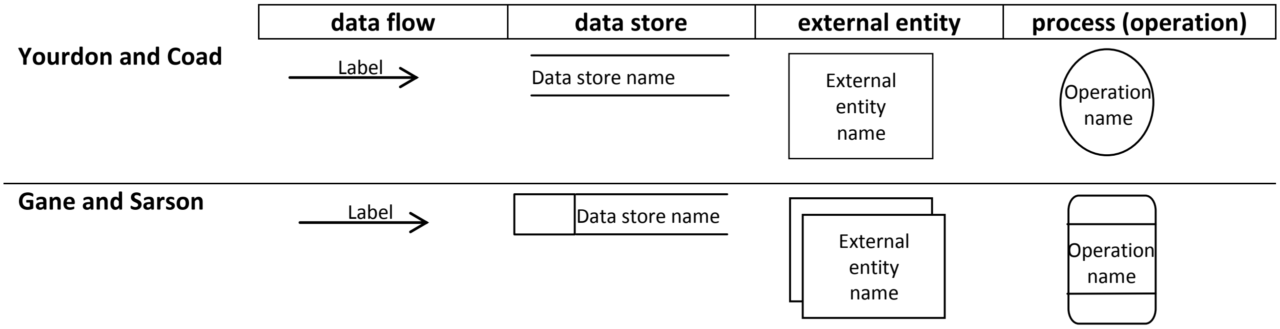
Modeling Technique	**Attribute**	**Duplicate Named Attribute Instances**	
		One Diagram	**Set of Like-diagrams**
UML State machine (statechart)	decision (branch/merge) ◇	each	each
	signal/event (accept) ⧐	once	once
	signal/event (generate) ⬠	once	once
	signal/event (timer) ⧗	once	once
	state/activity	once	each
	state action	each	each
	synchronize (fork/join) ▬	each	each
	transition ⟶	each	each
	transition action	each	each
	transition [guard condition]	each	each
UML Use case diagram	actor	once	each
	use case	once	each
	association ──	each	each
	generalization ──▷	Each	each
	dependency - - - - - - ▶	Each	each
	system boundary	Each	each

Appendix B – Modeling Technique Notations

The tables below show the notations used for many of the modeling techniques listed in Appendix A. In some cases, there is more than one popular notation for the modeling technique.

Data Flow Diagram

There are two popular notations for DFDs.

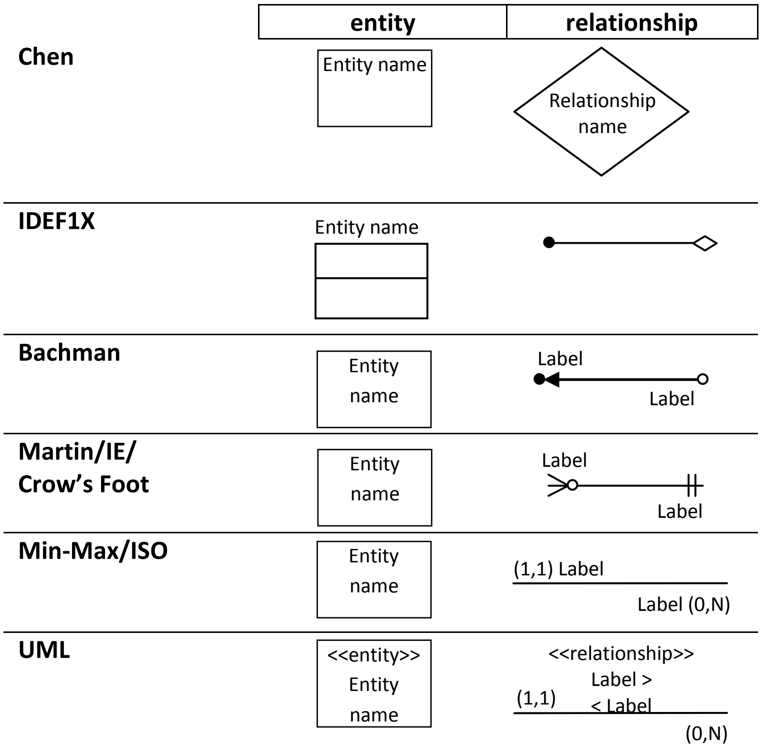


Data Model Diagram

The notation is similar to ERDs, except that attributes will be listed for each entity.

Entity Relationship Diagram

There are six popular notations for ERDs.



Note that any cardinality rules expressed in the ERD are not counted.

Event Table

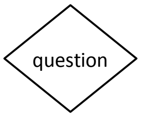
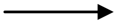
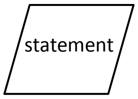
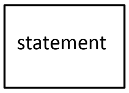
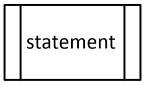
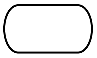
In the table below, **s/c/e** represents a state, condition, or entity.

	s/c/e/event 1	**s/c/e/event 2**	**…**	**s/c/e/event N**
s/c/e 1	actions and/or state change			
s/c/e 2		actions and/or state change		X
…				actions and/or state change
s/c/e M	X			

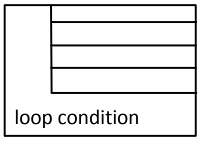
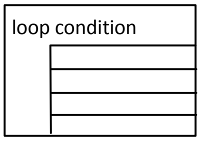
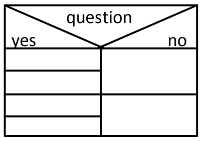
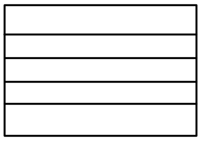
An "X" in a cell indicates that the state and condition combination is not possible.

An empty cell indicates that the state and condition results in no action and no state change.

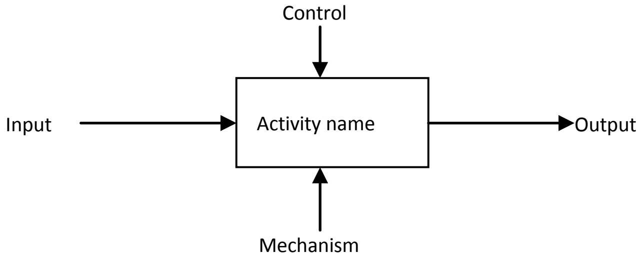
Flowchart

decision	**flow**	**input/output**	**processing**	**subroutine call**	**terminal**
					

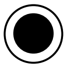
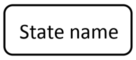
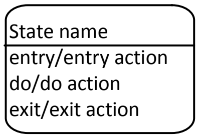
Nassi-Schneiderman Chart

iteration (test after)	**iteration (test before)**	**selection**	**Sequence**
			

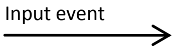
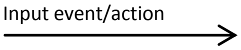
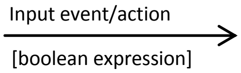
SADT Activity Diagram (ICOM)



State Transition Diagram

	initial	**final**	**no internal actions**	**w/ internal actions**
state				 State name entry/entry action do/do action exit/exit action

Each type of action specified in a state is counted as one action.

	no actions	**w/ action**	**w/ action & guard condition**
transition			

A [boolean expression] is counted as one guard condition regardless of the expression's complexity.

State Transition Table

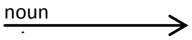
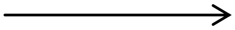
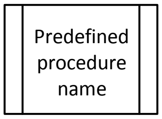
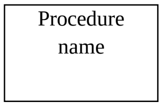
	event 1	**event 2**	**…**	**event N**
state 1	actions and/or state change			
state 2		actions and/or state change		
…				actions and/or state change
state M				

An empty cell in the table above indicates that the state and event is either not possible or results in no action and no state change.

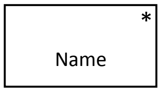
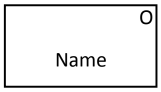
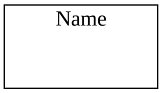
	next-state 1	**next-state 2**	**…**	**next-state M**
state 1	actions and/or event			
state 2		actions and/or event		
…				actions and/or event
state M				

An empty cell in the table above indicates that the state and next-state is either not possible or results in no action and is not triggered by any event.

Structure Chart

information flow	**invocation**	**predefined procedure**	**procedure**
			

Structure Diagram (Jackson)

connection	**iteration**	**selection (option)**	**sequence**
			

Appendix C - Document History

Date	Description
05/19/2009	Original document.
01/19/2010	Added rules for identifying and counting program code. Added notations to appendix A. Reformatted document.
05/18/2010	Added information about software artifact types in section 2.1. Modified rules for identifying and counting program code; to make it consistent with the source code counting tool (CntSrc).
06/10/2012	Converted from OpenOffice Document to Microsoft Word 2007. Made subtle wording changes to improve internal consistency of document.

Appendix D - References

[1] Voorhees, D.P., Mitropoulos, F.J. (2007). A Software Size Measure for Estimating Effort based on a Software Development Life Cycle. Proceedings of the 2007 International Conference on Software Engineering Theory and Practice, July 9-12, Orlando, Florida, USA.