A Size Metric For UML

Lee Fischman
COCOMO / SCM 14
October 1999

Why A Metric For UML?

Software metrics. Are a cornerstone of software estimating. Numerous metrics are highly correlated with outcomes for effort and duration. Object-oriented design and languages. Are gaining mainstream prominence in software development. They seem to better model the problem domain, while promising more maintainable systems. Existing metrics. Are not necessarily suited to estimating object-oriented systems. Lines of code are meaningless in more automated development environments, function points do not map into objects but into delivered functionality. UML. Is becoming the de facto specification language in OOA/D. A metric that is specific to UML would directly support the needs of many modern developers.
Comparing CMPs To Function Points

Normal counting procedure for function points (and FBS):

Find function points using counting rules

Rank by data attribute count

# of functions # of files

Adjust total count using gross adjustment factors

Adjusted Count

Counting procedure for class-method points:

Find CMPs using counting rules

Rank by data attribute count

# of method pts # of class pts

CMPS fit directly into effort & duration estimating formulae. Other parameters capture gross project factors

Alternatives To CMP Discrimination With FBS Counting Rules

Methods for determining class and method points:

FBS Rules Analog Cyclomatic Complexity simple counting

Mitigates trivial methods and empty classes**

Inherently calibrated *

Early use sort of

No training required

© Galorath Incorporated 1998
G A SEER Technologies
Quick Intro To The UML

A synthesis. Of previous object-oriented specification languages, sharing with these a graphical view of system specification... a series of charts.

Uses several types of charts. Offering different levels of detail, perspective, and also different information.

Requires a varying number of charts. Depending on the size and complexity of a system; more than one of each type of chart may be drawn. Not all chart types are always required.

Relies on a core set of charts. “Use cases” for basic layout of system scenarios, “static structure charts” for basic system architecting.

UML is an abstraction above the language layer. Systems that are defined in UML are actually realized in an OO language (C++, Java, Smalltalk, Ada...).

UML Essentials... Fundamental Charts

Use Case

Class Diagram

An alarm system...
Survey of Software Size Metrics

Lines of code. Most wide-spread metric due to ease to use, best-suited to traditional (non-"visual") development. In its time, the best indicator of project effort and duration.

Function Points. Second generation metric, best for capturing size in terms of functionality delivered. Notable for its neutrality with respect to underlying implementation. Studies indicate that it correlates with effort and duration.

Object metrics. Much research; little agreement. Inviting realm in which to apply a metric due to the discipline of the OO approach, enabling potentially strong mappings. A metric for OO systems should if possible be designed around standard OO artifacts, rendering it easy to use.

OO Metrics To-Date - A Survey

Semantic Object Model. Developed by Graham (1995), uses a core sizing metric that is similar to function points.

3-D Function Points. Developed by Whitmire (1996), extends function points with additional OO-specific metric elements.

System Meter. Developed by Moser (1996), develops a metric based on objects.

Function Points for OOA/OOD. Developed by IFPUG (1996?), maps function points into object-oriented artifacts.
A "Good Metric" Checklist

✓ Simple to deploy (KISS). The metric should be understood by metrics practitioners and developers with little advance training. It should be appropriate for most circumstances within the problem domain.

✓ Usable early on. It should be possible to quantify a project at an early stage using this metric. The metric-developed measure should become more precise as additional information becomes available.

✓ Unambiguous. There should be no question regarding proper uses of the metric. It should not be misunderstood within its intended community.

✓ Highly correlated. With effort, duration, defects, etc.
"Best of Class" in OO Metric Research

Weighted Methods Per Class


Definition: measure of complexity within a class. Can be defined as:
- The sum of the cyclomatic complexities for each method in the class (Kolewe (1993))
- Number of methods implemented within a class, easier to count but does not account for variations in method complexity (Rosenberg (1998)).

Weighted Methods Per Class

From the UML Class Diagram...

3 Classes...

Program

Current Config

Event Log...

2 methods per class...
6 methods in total
either way, information is equivalent
OO Metrics To-Date (continued)

Analogy. Deployed in multiple methods, including SEER-SSM. Actuals for numerous previous projects must be available for prediction, as must a standard characteristic for use as a basis of comparison.

Mapping of OO-Jacobson Into FPA. Developed by Fetcke, Abran and Nguyen (1998), uses a size metric that is similar to function points.

Predictive Object Points. Developed by Minkiewicz (1998), counts number of classes and methods.

Nesi (1998). Counts the number of classes, further classifying these into effort ranges depending on complexity; this is determined heuristically based on # of methods, etc.

Good OO Metric...
Also A Good UML Metric?

Paradigm ➔ Metrics

OO-A/D ➔ Metrics

UML encodes the object-oriented paradigm.

UML ➔ OO-A/D ➔ Metrics
...Into “Class-Method Points”

What criteria can be used to differentiate between trivial and substantial methods?

Raw Count of Methods

“Method Points”

Raw Count of Classes

“Class Points”

Function-Based Sizing (FBS): Equivalent to function points but easier to learn, with an extension for complex internal functions.

Effect of Adjusting Classes and Methods With FBS Rules

All substantial methods are counted. These will include:

- All methods (or collections of methods) that handle features seen at the user interface which leave the application in an equilibrium (consistent) state.
- All methods deemed substantial entities, though involving no user interaction. These methods should at least be equivalent in complexity to those selected using criteria in the bullet above.

All substantial classes are counted. Classes that carry significant internal data structures are strongly designed entities in themselves, requiring measurable effort simply to lay out attributes. Other classes are simply receptacles for methods; in this case, counting their methods alone is sufficient, anything more would be over-counting.
Can Only One UML Diagram Make An Adequate UML Metric?

**Concern:**
Despite the number of diagrams UML supports, weighted methods per class involve only the UML class diagram.

**Restating the Goal:**
To obtain a metric that gives a good indication of effort and duration, while being easy to gather.

**Response To Concern:**
The source chart in UML is the class diagram. Other charts may capture underlying complexity, but not additional size. The full dimension of the system is apparent from the static structure (class) diagram alone.

It is also easy to count methods and classes from this chart.

Further Refinement of WMCs...

**Problem: Trivial Methods**

**Statement:** A UML Class Diagram may contain many trivial methods that do not strongly affect effort but which are nonetheless counted.

**Treatments:**
- Sum cyclomatic complexities in each class, automatically adjusting for simple methods -- *Too much work for a "workhorse" metric.*

✓ Count only substantial methods -- *What criteria can be used to differentiate between trivial and substantial methods?*
Early Sizing: Use Case Points

When only the scenarios for the system are known, can you obtain a size estimate?

Research indicates:

1 noun in spec = 1 ILF = average number of FPs

Translated to UML (VERY ROUGH):

Use case scenarios → # of internal data structures

# of CMPS

References

Graham, Ian: Migrating to Object Technology (1995) Addison-Wesley
IFPUG: Function Points for Object-Oriented Analysis and Design - Case 3
Whitemore, Scott: 3-D Function Points: Applications for Object-Oriented Software in Applications of Software Management Conference (1996)