# Function Point Analysis and Executable UML

Lee Riemenschneider

June 9, 2011

**Abstract**

This article isn't intended to be a lesson in Function Point Analysis or Executable UML. It strictly provides some direction on how to combine the two technologies. A basis is given for the reader to pursue more thorough explanations of the technologies. It should be noted that this estimation method is valid, even if Executable UML modeling isn't used to implement the software.

# Part I
# Basis

The basis for estimation is taken from the Garmus and Herron function point analysis book[1]. The basis for the Executable UML terminology is defined in the seminal book,[2] and an overview can be found here.

# Part II
# Method

## 1 Initial Estimation

The preliminary analysis of the software system should be done per the Executable UML guidelines for defining domains, bridges, and classes.

- the unique subject matter domains within the systems should be identified.
- the dependencies between domains (i.e., bridges) should be identified.
- the classes within a domain should be identified. This is commonly referred to as an *object blitz*.

As this is a preliminary analysis, it isn't expected that everything is correct. The whole exercise shouldn't take more than one working day.

Domains define the application boundary in function point analysis terms. Domains that are already developed[1] usually don't need to be included as they are assumed to require no effort, but dependencies to and from these domains will need to be examined. If an existing domain is known to require rework, then the rework effort should be estimated as well.

Domains that are expected to not be modeled in Executable UML are to be included. Even though these domains might not be implemented as classes, they can be classified for use in the estimation. e.g., a complex math domain could have each unique algorithm identified as a class, which would have it's own complexity scaling.

The object blitz will usually come out close to the actual number of classes. As with any estimation method, after enough data is collected, a common modifier can be factored into the estimate. Care should be taken to identify any generalizations, as these are considered differently by the estimation method than a normal class.

## 2 Subsequent Estimation

As the software development work progresses, actual metrics can be determined and compared against the estimate to determine if any adjustments need to be made to the schedule. This activity should be scheduled such that it allows for timely adjustments. Again, once enough data is collected, the inaccuracy of the estimates will be able to be determined and corrected.

## 3 Estimation Method

The estimation should follow the recommendations for Object-Oriented Analysis in chapter 12 of the Garmus and Herron book.[1] The following list adds enhanced, Executable UML-based explanations to what already exists in Table 12-2 of the book.[1]

ILF — each class identified in the object blitz will be an ILF for the associated domain. Complexity should be based on the number of attributes for the class and any dynamic behavior associated with the class. Rules for this can be gleaned after enough estimation data is compiled.

EIF — objects contained in one domain and referenced in another are considered EIFs to the referencing domain. Executable UML supports this concept only through the *composite* data type, but some tools allow the user to define structured data types to mimic external objects. Per the Executable UML rules for usage of composite data, all EIFs should be considered of low complexity.

[1] Realized or existing modeled domains.

RET — generalizations are counted as RETs of the ILF, that represents the root class.

Externals — EIs, EOs, and EQs are identified via the bridge identification, so a bridge between domains should be analyzed for the number of EIs, EOs, and EQs within the bridge. Each end of the bridge will have its own EI, EO, and EQ numbers.

Once the numbers are determined, the standard function point analysis rules can be used for assigning the value adjustment factors for each domain.

## References


[1] Function Point Analysis: Measurement Practices for Successful Software Projects. David Garmus and David Herron. 2001. Addison-Wesley. ISBN: 0201699443

[2] Executable UML: A Foundation for Model-Driven Architecture. Stephen J. Mellor and Marc J. Balcer. 2002. Addison-Wesley Professional. ISBN: 9780201748048