Hierarchical Software Size Metrics

Petri Vesterinen

Nokia Research Center
P.O.Box 422, FIN-00045 NOKIA GROUP, FINLAND
tel: +358 0 43761, fax: +358 0 4376 6855
e-mail: petri.vesterinen@research.nokia.com

Abstract:

Dozens of different software size metrics have been proposed and used. The most known ones are Lines of Code and Function Point metrics. Former one depicts the physical size and latter the functional size of software.

It is known that one metric does not suit for all purposes. For example, Lines of Code may not be a suitable metric for early estimation, and Function Points is a too high-level metric for some purposes.

The most common approach is to use only one metric despite of problems. Another common approach is to use different metrics for different purposes. It is shown here that this approach is not a satisfying one either.

There is a need for a hierarchical software size metric that 'grows up' or 'matures' along with software projects. Concerning this kind of metric, the lower levels in the hierarchy can use more detailed information available to decide the values of the metric.

We at Nokia Research Center have developed a hierarchical, three-level software sizing metric. It uses the same sizing unit at all levels. The most abstract (the highest) level can be used in early phases and/or for large software projects. The lowest level is detailed enough to be used in accurate effort estimation.
INTRODUCTION

Software size metric is perhaps the most important and most used single software metric. Size is used within several planning and tracking activities in software projects. Software size is also used in several important derived metrics like productivity (size/unit of effort) and quality (defects/size). Importance of software size metric is closely argued in e.g. [Huy96, Fen97, Ves97].

Dozens of different software size metrics have been proposed and used. Size is typically a direct count of selected characteristics to describe the volume of a software item. The different size metrics can be loosely categorized into three different categories: the physical, the functional and the logical size.

The physical size is the count of architectural (and physical) items at a selected architectural level. Some examples of this kind of metrics are:

- Lines of Code (LOC) [Par92, Zus98]
- Number of modules (or other similar objects) [Gil77]
- Number of classes [Lor94]
- Number of bytes used by the program

The functional size tries to depict the amount of meaningful functions / services to the user of the software. The functions are counted typically at some abstract level, the size thus being independent of the implementation. The most known metric in this category is Function Point Analysis (FPA) as defined by the International Function Point User's Group (IFPUG) [IFP94]. Capers Jones has listed dozens of variants of this original method. Object Points method [Ban91, Rei97] can be counted in this category also.

The logical size tries to combine logical complexity together with physical or functional size. Halstead's software science metrics [Hal77] and Function Bang [DeM84] can be regarded as such solutions.

The most known and used software size metrics are Lines of Code (LOC) and Function Point (FP) metrics. The discussion in the rest of the paper is mostly restricted to these and to the proposed new size metric.

PROBLEMS WITH LOC AND FP METRICS

LOC and FP are good metrics when they are used in right context. The LOC metric is usually easy to count and counting can be automated easily. The meaning of LOC is self-evident and clear (if the counting rules are well documented). LOC metric is easy to use, and it is used in many software estimation models and tools.
The main strengths of FP metric are early availability and independence of implementation. The functional view is also more meaningful to the user of the software.

However, there are many problems. LOC is not suitable for software estimation when the number of LOCs cannot be estimated accurately enough. It is often the case in early estimation, and when new technology is going to be used.

LOC is also unsuitable for derived metrics like productivity or quality. It can only be used within very narrow scope (e.g. when only one programming language is used, and the coding style is homogenous).

FP metric is not suitable for early estimation when the function types cannot be identified. The counting may also be too laborious in early phases. FP may not be accurate enough during late project phases. During later phases LOC-based estimation is often more accurate.

Figure 1 shows the primary functional phases of FP and LOC metrics. FP’s primary phase is requirements specification, because (customer) requirements are the basic input of FPA by definition. Primary area for LOC metric is module implementation and phases thereafter, as the LOC can not be measured before the program code is ready.

Figure 1: Primary functional phases of FP and LOC

The figure 1 explains why FP metric is often seen as superior in estimation (because of its early availability). It also explains why LOC-based estimates are often more accurate (because LOC metric uses more detailed data).
AVAILABLE SOLUTIONS

The most common approach is to use only one metric despite of problems. Many organizations use only LOC, because it is an older metric, and they already have collected huge amounts of corresponding history data. It seems not to be very reasonable to waste this information. Many organizations also have automatic code counting tools in place. The problem of this approach is that the selected metric (LOC or FP) is good only for some purposes.

Another common approach is to use different metrics for different purposes. In many cases FP is used to agree on the project size with customer, and it is also used in early estimation. LOC may be used in late estimation, quality measurements and as a predictor of memory consumption, for example. COCOMO 2.0 uses Object Points, FP and LOC metrics during different life-cycle phases of software projects [Rei97].

Also the approach to use several metrics is not a fully satisfying one. Several metrics has to be taken, and this results to increased cost of measuring. Different metrics depict different views and are not comparable. It may be impossible to find reliable conversion factors between the metrics. Capers Jones maintains a list of conversion factors between LOCs and FPs for hundreds of programming languages. The conversion factors are only averages. Capers himself states that the range may be more than 50% on each side of the nominal average [Jon94].

THE NEED

We need a hierarchical software size metric that:

1. uses same metric units in all project phases. We can thus avoid conversion between different size units like FP and LOC.
2. is cost-effective to measure in all phases. This means practically that measuring should be easy and quick during early project phases.
3. is implementation independent during phases when implementation techniques are not known. It should act like FP in this sense.
4. is implementation dependent in phases when implementation techniques are known. Thus it practically would be more accurate than FP and should act like LOC in this sense.
5. takes use of the increased understanding of the nature of the work as specification and design work proceeds. It should thus 'grow up' or 'mature' along with software projects.
6. can be used in derived metrics like productivity and quality.
7. is easy to understand.
8. is based on a solid theory.
The figure 2 clarifies the need. It shows that current metrics are static metrics when the amount of information used is considered. We can see that during early phases some information has to be estimated and during late phases some information is not utilized when using this kind of metrics. The ideal metric should be capable to use all available size information in a software project.

Figure 2: Need for a new 'dynamic' software size metric

NOKIA SOLUTION

We at the Nokia Research Center have developed a solution where we have three fully integrated, hierarchical levels within our software size model. The levels may be called here as "Release", "Feature" and "Details" levels. All these levels produce size figures using the same unit. The unit may be called here as "Nokia Sizing Point" (NSP).

The Release level is meant for sizing software releases during early phases when the release contents are planned. The Feature level is meant for sizing features or subprojects. The Details level is the most detailed one. It is mainly used in software design phase and thereafter. Figure 3 shows the target areas.
The information that is needed in each of the levels in determining the size is shown in figure 4. The Release level is of course the most independent of the architecture and the Details level is architecture dependent. The Release level is somewhat subjective, but the Details level is totally objective. The Details level is described in more detail in [Ves96].

The Feature level weights are fixed. It is much the same than the fixed weights of function types in FPA. Weights for the Details and Release levels are to be calibrated. The Release level weights are organization dependent and the Details level weights are architecture-dependent. The idea of hierarchical calibration is shown in figure 5.
AVAILABLE SOLUTIONS FOR FP

There are also some available solutions in the context of FP that satisfy some of the wanted properties mentioned earlier. Some of these solutions are shown in figure 6. Early Function Points [San81] is really a hierarchical solution as it uses the same metric units as the IFPUG FPA. However, 3D-FP [Whig51] and Full Function Points (FFP) [May98] are not truly hierarchical solutions, as they do not use the same measurement units. They are more like extensions of the FPA.

Figure 6: Some available solutions in FP context
SUMMARY

It has been shown that current size metrics are suitable only for (more or less) restricted use. When multiple metrics are used it is hard to cope with different size units. Current metrics are static. The ideal metric should ‘grow up’ or ‘mature’ along with software projects. There really is a need for a hierarchical software size metric.

This paper has described a corresponding solution developed within Nokia. The Feature and Details levels have already been used in several software projects. The preliminary results suggest that the idea really works. The proposed metric has been easy to understand and use. The calibration at the Release and Details levels is the critical activity when taking the metric into use.

A future research interest is to form a true hierarchical size measurement framework within the FP context. Another interesting context would be within the object-oriented paradigm.

REFERENCES


[Rei97] Reifer, D., COCOMO 2.0 Tutorial. 12th International Forum on COCOMO and Software Cost Modeling, University of Southern California, Los Angeles, CA, USA, October 6-8, 1997.


