IRM PRESS
701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.irm-press.com

ITB11782

This chapter appears in the book, *Skills for Managing Rapidly Changing IT Projects* by Fabrizio Fioravanti © 2006, Idea Group Inc.

Chapter XIV

Software Measurement

The problem of measurement in software engineering has been addressed by many authors, and one of the most common questions is, "Can we learn from measurement in physics and can we transform this to software engineering measurement?" (Zuse, 1997, 1998).

An engineering measure, from the physical point of view, is relevant if it can quantify the object under measurement, since qualitative measures usually are considered too coarse. The problem arises when we shift this concept in software engineering, where nearly all the possible measurements are qualitative ones. All of the characteristics defined by ISO9126 (1991) are qualitative and not directly related to precise *physical* or tangible phenomena.

On the other hand, a wide skepticism of using numerical values is diffused, because it can be hard to give the requested semantic to the number.

Metrics, that are the measures performed on code, can be split into two categories: metrics for software complexity/size measurement and effort estimation, and metrics for qualitative characteristics evaluation. In the first part of the chapter, the former type of metrics is discussed, while in the second part, a general overview of quality in use by metrics will be performed. Apart from this general distinction, a more accurate taxonomy of the software metrics is reported in order to classify them on the basis of the applicability field in which they can be adopted.

Metric Taxonomy

Metrics can be classified according to different criteria. A first classification can be performed on the basis of its capability to predict and/or to evaluate a posteriori system characteristics.

The main difference between these two classes of metrics is in the use or not of the functional code. In general, predictive metrics are evaluated only on the basis of the class interface (e.g., C or C++ Header files, Java class definition, without inline code, etc.), while a posteriori metrics also need class code (method implementation).

In general, a posteriori metrics consider all class aspects — attributes, methods interface, and method implementation (both locally defined and inherited). Predictive metrics also can be evaluated, if the implementation phase has not been performed yet, such as in the early phase of system development. These metrics also can be used to predict the adaptive maintenance effort by knowing only the interface that classes will have at the end of the system adaptation. This can be very useful for evaluating the adaptive maintenance effort needed during its planning phase.

A metric is either able to measure directly a software characteristic or not; thus, it can be classified as direct or indirect. Direct metrics should produce a direct measure of parameters under consideration; for example, the number of Lines of Code (LOC) for estimating the program length in terms of code written. Indirect metrics usually are related to high-level features; for example, the number of system classes can be supposed to be related to the system complexity by means of a mathematical relationship, while LOC (as an indirect metric) is related typically to development effort. Thus, the same measure can be considered as a direct and/or an indirect metric, depending on its adoption. Indirect metrics have to be validated for demonstrating their relationships with the corresponding high-level features. This process consists of (1) evaluating parameters of metrics (e.g., weights and coefficients) and (2) verifying the robustness of the identified model against real cases. The model can be linear or not, and it must be identified by using both mathematical and statistical techniques (Briand, 2000a, 2000b; Nesi, 1998; Rousseeuw, 1987; Schneidewind, 1992, 1994).

Metrics are frequently reclassified on the basis of the phase in or for which they can produce significant estimations; therefore, a distinction can be made among analysis, design, and code metrics. An exact classification of metrics, according