Software Metrics
An Overview
Version 1.0

Simon Alexandre
CETIC asbl - University of Namur
Software Quality Lab
Belgium

July 2002

Contents

Chapter 1

History and Definitions

1.1 A brief history of the metrics

Software metrics lies on the ancient discipline of measurement mainly developed by scientists (physicians). On this basis, some take up measurement principles in order to measure software activities.

So metrics origin goes back to the sixties with the Lines of Code (LOC) metric used to measure programmer's productivity and program quality (e.g. number of defects per KLOC[1]). The main aim was to provide information to support quantitative managerial decision-making during the software lifecycle [20, p.357].

1.2 Definitions

There's two main concepts to define at this point : *the measurement activity* and *the software metrics.*

1.2.1 Measurement activity

Norman Fenton gives the two following definitions of the *measurement activity* [22, p.28]:

> *Formally, we define measurement as a mapping from the empirical world to the formal, relational world. Consequently, a measure is the number or symbol assigned to an entity by this mapping in order to characterize an attribute.*

He gives this second definition introducing the numerical aspect [22, p.5]:

> *Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules.*

[1] KLOC for thousands of lines of codes

1.2.2 Software metrics

The first definition of *software metrics* is proposed by Norman Fenton [20, p.358]:

> (...)*software metrics is a collective term used to describe the very wide range of activities concerned with measurement in software engineering. These activities range from producing numbers that characterize properties of software code (these are the classic software 'metrics') through to models that help predict software resource requirement and software quality. The subject also includes the quantitative aspects of quality control and assurance - and this covers activities like recording and monitoring defects during development and testing.*

An other definition of software metrics is due to Paul Goodman [25] :

> *The continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products". Applied To Engineering & Management Processes, Products & To Supply*

Chapter 2

Theory of Measure

2.1 Theory

Measurement theory has first been developed as a particular discipline of physics. Physicians defined fundamental rules in order to use correctly the new framework of measures. Further research led to develop several theories of measures. Among them the *representational theory of measurement* has been used to build the software metrics [22, p.24].

The representational theory of measurement aims at formalizing our intuition about the way the world works. We collect a set of data, the measures, which should represent attributes of the entities observed. Manipulation of these data must preserve the relationship observed among these entities [22, p.24-25]. Actually we collect measures about the real world and try to understand these measures by comparing them. By example, we observe that certain people are taller than others without measuring them. In fact, says Fenton, *our observation reflects a set of rules that we are imposing on the set of people.*[22, p.25]. So we define some binary relations between these entities.

> Example 1 : When we say that X is taller than Y we define a binary relation between X and Y. In this case, "taller than" is an empirical relation for height.

The measurement activity is then defined as the mapping between the empirical and the formal world[1]. So a measure is a number or symbol assigned to an entity in order to characterize an attribute.

Basic rules are simple : the real world is the *domain* and the mathematical world, the *range*. When we map an attribute to a mathematical system the following rules must be respected :

> *(...)the representation condition asserts that a measurement mapping M must map entities into numbers and empirical relations into numerical relations in such a way that the empirical relations preserve and are preserved by the numerical relations.*[22, p.31]

[1] See Section 1.2.1

The key stages of the formal measurement are easily representable :

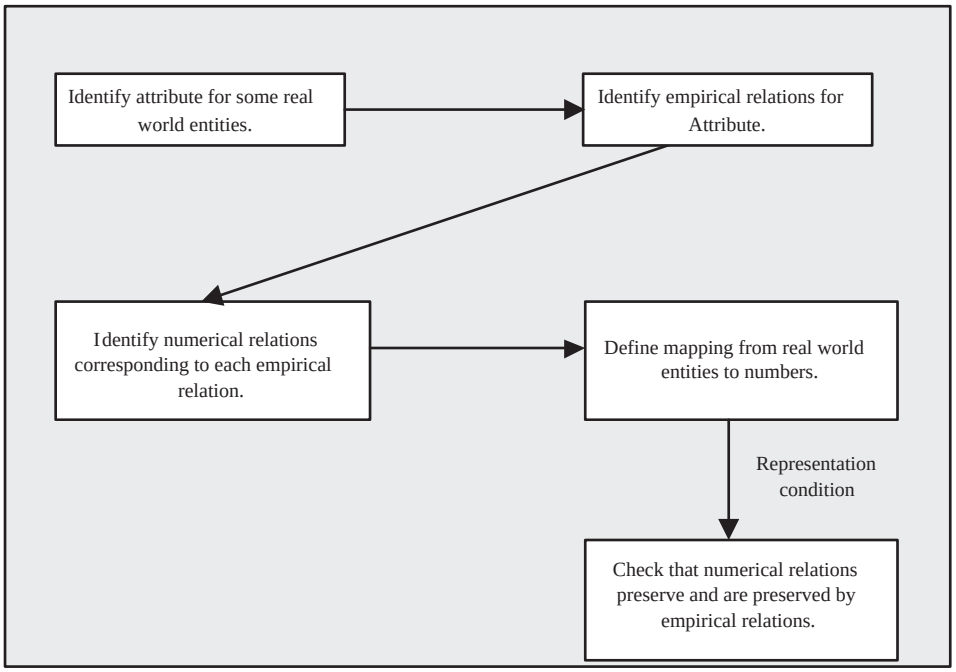

figure 2.1: Formal measurement [22, p.33]

To avoid traps (compare measures from entities which are not comparable) several models have been introduced[2]. A model is an abstraction of reality, allowing us to strip details away and view an entity or concept from a particular perspective. [22, p.36-37] Several models are likely to interest software metrics : *cost-estimation model, quality models, capability-maturity model,....* The use for such models avoid to focus only on the formal and mathematical systems and neglect the empirical one. A model will show fundamental characteristics and how they are articulate. This will allow to define a metric for each characteristic.[22, p.38]

The model chosen, entities and attributes defined, measures can be defined too. But when there are complex relationships among attributes, or when an attributes must be measured by combining several of its aspects, then we need a model for how to combine the related measures. For this reason, direct and indirect measurements are distinguished.

2.1.1 Direct measurement

Direct measurement of an entity attribute involves no other attribute or entity. For example, we can measure the length of a physical object without any other object. Measures below are direct measures used in software engineering :

[2]More details on models in chapter 3

- Length of source code (LOC)
- Duration of testing process (Hours)
- Number of defects discovered (counting defects)
- Time a programmer spent on a project (Months)

2.1.2 Indirect measurement

Indirect measurement are measures of an attribute obtained by comparing different measurements. For example, *Number of defects* divided by *module size* gives the *Module defect density.*[22, p.40]

2.2 Measurement scales

Previous section shows how direct measurement assigns a representation or mapping from observed relation system to numerical relation system. These kind of measures are done in order to extract relationship between data and to draw conclusion about them. However all the mappings are not the same. Differences between them restrict the possible kind of comparison and analysis. To avoid unappropriate analysis, ***measurement scale*** concept has been introduced as a principle by scientists.[22, 45-47] Five major measurement scales are identified[37][22] :

- Nominal
- Ordinal
- Interval
- Ratio
- Absolute

A ***nominal*** scale puts each entity into a particular category, based on the value of the attributes. It's the same process when we identify a programming language. By reading the code you can recognize it and classify it. This scale has two major characteristics :

- The empirical relation system only consists of different classes ; there is no notion of ordering among the classes.
- Any distinct numbering or symbolic representation of the classes is an acceptable measure. But there isn't any notion of magnitude associated with the number or symbol.

  Example : For instance, we try to classify the set of software faults in the code. We choose a measurement scale where faults are entities and their location are attributes. So fault location could be in three different sets : specification, design or code. Then we can define a mapping M that assign the different classes to a particular number. 45 if x is a specification fault, 2 if x is a design fault and 37 if x is a code fault. The value is not important here.

An ***ordinal*** scale ranks items in an order, such as when we assign failures a progressive severity like minor, major, and catastrophic. This scale has three characteristics :

- Empirical relation system consists of ordered classes with respect to the attribute.
- Any mapping that preserves the ordering is acceptable.
- Numbers represent only ranks, so addition, and other mathematical operations have no sense.

  Example : You want to classify the different modules of your software in three classes which denote the complexity (trivial, simple, complex). Then you choose a mapping M like in the nominal scale : 1 if x is trivial, 2 if x is simple and 3 if x is complex. The difference with the previous scale lies in the fact that the measurement mapping must preserve the complexity order. 3 is bigger than 1 preserve the relation more complex.

An ***interval*** scale defines a distance from one point to another, so that there are equal intervals between consecutive numbers. This property permits computations not available with the ordinal scale, such as calculating the mean value. However, there is no absolute zero point in an interval scale, and thus ratios do not make sense. Care is thus needed when you make comparisons. The three main characteristics are :

- Order are preserved.
- Differences are preserved but not ratios.
- Addition and substraction are acceptable but not multiplication and division.

  Example : Take the temperature measurement on a Celsius or Fahrenheit where each degree is a class related to heat. We say that the temperature in a place X is 20 degrees Celsius and, at the same time, 30 degrees Celsius in place Y. If the temperature move, in X, from 20 to 21 the heat will increase exactly in the same way if it change from 30 to 31 in place Y. So the relationship is preserved.

The scale with more information and flexibility is the ***ratio*** scale, which incorporates an absolute zero, preserves ratios, and permits the most sophisticated analysis. Measures such as lines of codes or numbers of defects are ratio measures. It is for this scale that we can say that A is twice the size of B. There are four characteristics :

- Ordering, size of the intervals between entities and ratios are preserved.
- There is a zero element (represents total lack of attribute).
- Measurement mapping start at zero and increases at equal intervals (units).

- All arithmetic can be applied to the classes in the range of the mapping.

  Example : You use a ratio scale when you measure the physical size of entities. The scale start at zero which represent the total lack of size (theoretical - no existence). You can measure size in centimeters, meters,...

The ***absolute*** scale of measurement is the more restrictive scale. For any two measures, *M* and *M'*, there is only one admissible transformation : the identity transformation. So there's only one way in which the measurement can be made, so *M* and *M'* must be equal. The absolute scale respects the four following properties :

- The measurement is made simply by counting the number of elements in the entity set.
- The Attribute always takes the form "number of occurrences of x in the entity".
- There's only one measurement mapping, namely the actual count.
- All arithmetic analysis of the resulting count is meaningful.

  Example : Lines Of Codes (LOC) is an absolute scale measurement of the attribute "number of lines of codes" of a program. But "number of centimeters" is not an absolute scale measurement of an person's size because you can also use inches, meters,...

2.3 Validation

Validation of the measures is necessary to do before analysis. It aims prove that metrics used are actually measuring what they claim they do. Pfleeger[37] say that a measure is *valid* if it satisfies the representation condition : if it captures in the mathematical world the behavior we perceive in the empirical world. For example, we must show that if **H** is a measure of height, and if **A** is taller than **B**, then **H(A)** is larger than **H(b)**. But such a proof must, by nature, be empirical and is difficult to demonstrate. So, we must consider wether we are using direct measure (size) or an indirect measure (number of decision points as measure of size) and which entity and attribute are being addressed.Currently there isn't any accepted standard for validating a measure [22, p.106-108].

Chapter 3

Software measures

3.1 Classes and attributes

In the previous section measurement basis and rules have been presented. The first activity to achieve in measurement is the entity and attribute identification. In software there are three classes :

- Processes : collection of software-related activities.
- Products : artifacts, deliverables and documents resulting from processes
- Resources : entities required by a process activity.

In each class of entity, we distinguish between internal and external attributes :

- Internal attributes *of a product, process or resource are those that can be measured purely in terms of the product, process or resource itself. In other words, an internal attribute can be measured by examining the product, process or resource on its own, separate from its behavior.*[22, p.74]
- External attributes *of a product, process or resources are those that can be measured only with respect to how the product, process or resource relates to its environment. Here, the behavior of the process, product or resource is important, rather than the entity itself.*[22, p.74]

3.2 Processes

Processes are measured to inform on duration, cost, effectiveness and efficiency of software development activities. There is several internal process attributes which can be measured directly :

- the duration of the process or activity
- the effort associated with process or activity
- the number of incidents of a specified type arising during process or activity

Example 1 of measure for a process [22, p.77]:

> *(...)we may be reviewing our requirements to ensure their quality before turning them over to the designers. To measure the effectiveness of the review process, we can measure the number of requirements errors found during specification. Likewise, we can measure the number of faults found during integration testing to determine how well we are doing. And the number of personnel working on the project between May 1 and September 30 can give us insight into resources needed for the development process.*

Example 2 of measures from AT&T [22, p.77] :

> *AT&T developers wanted to know the effectiveness of their software inspections. In particular, managers needed to evaluate the cost of the inspections against the benefits received. To do this, they measured the average amount of effort expended per thousand lines of code reviewed. As we will see later in this chapter, this information, combined with measures of the number of faults discovered during the inspections, allowed the managers to perform a cost-benefit analysis.*

3.3 Products

Products can be also measured. By products we mean not only items delivered to customer. All the artifacts, documents and prototypes produced during the process are considered as products. All these process outputs can be measured in term of quality, size,... For all of them we distinguish both external and internal attributes.

3.3.1 External attributes

External product attributes depend on product behavior and environment that influence the measure. Example of external attributes are : *usability, integrity, efficiency, testability, reusability, portability, operability*[22, p.78].

3.3.2 Internal attributes

Internal products attributes are easy to measure in terms of size, length, functionality, correctness.[22, p.78] Code clarity is an example of internal attribute according to defined rules like *"avoid GOTO"*.

3.4 Resources

Last measurable entities are the resources like personnel, materials and methods.[22, p.82] Measuring resources help managers to understand and control the process. Programmer's productivity is often measured in terms of lines of code.

Chapter 4

Models

4.1 Measures and Models

As explained in chapter 2 models must be used with metrics to avoid metrics misuse. A *Model* is *an abstraction of reality, allowing us to strip away detail and view an entity or concept from a particular perspective*[22, p.36]. Models can take the form of equations, mapping or diagrams. It allow to understand relationship between the component parts related one to another in the model. Fenton gives an example of this kind of relation highlighted in a model :

> *To measure length of programs using lines of code, we need a model of a program. The model would specify how a program differs from a subroutine, wether or not to treat separate statements on the same line as distinct lines of code, wether or not to count comment lines, wether or not to count data declarations, and so on. The model would also tell us what to do when we have programs written in a combination of different languages. It might distinguish delivered operational programs from those under development, and it would tell us how to handle situations where different versions run on different platforms.*[22, p.37]

A model gives the domain and range of the measure mapping and it describes the entity and attribute being measured, the set of possible resulting measures, and the relationship among several measures.

Another characteristic of models is that they distinguish the prediction from the assessment (measure to estimate future characteristics from previous ones or the determination of the current condition of a process, product or resource).

4.2 Goal-Question-Metric paradigm

4.2.1 Origins

The Goal Question Metrics approach (GQM) has been suggested by Basili and his colleagues in 1984.[5][4] They proposed an original approach to selecting and implementing metrics. The GQM principle consists first in expressing overall goals of the organization. On this basis, questions whose answer to these goals

are derived. Finally each question is analyzed in terms of what measurement is needed to answer each question.

4.2.2 The paradigm

GQM provides a measurement framework involving three steps :

1. List the major goals of the development or maintenance projects.
2. Derive from each goal the questions that must be answered to to determine if the goals are being met.
3. Decide what must be measured to answer the questions adequately.

The following figures illustrate how metrics are generated :

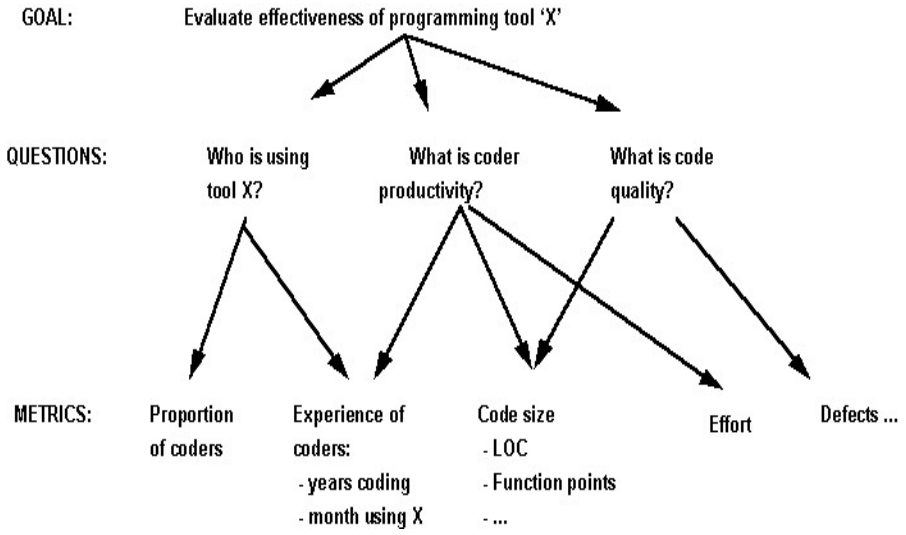

figure 4.1: GQM example tree

Fenton gives an example [22, p. 84]:

> *Suppose your overall goal is to evaluate the effectiveness of using a coding standard (...). That is, you want to know if code produced by following the standard is superior in some way to code produced without it. To decide if the standard is effective, you must ask several key questions. first, it is important to know who is using the standard, so that you can compare the productivity of the coders who use the standard with the productivity of those who do not. Likewise, you probably want to compare the quality of the code produced with the standard with the quality of non-standard code. Once these questions are identified, you must analyse each question to determine what mst be measured in order to answer the question. For example, to understand who is using the standard, it is necessary to*

know what proportion of coders is using the standard. However, it is also important to have an experience profile of the coders, explaining how long they have worked with the standard, the environment, the language, and other factors that will help to evaluate the effectiveness of the standard. The productivity question requires a definition of productivity, which is usually some measure of effort divided by some measure of product size. (...) the metric can be in terms of lines of code, function points, or any other metrics that will be useful to you. Similary, quality may be measured in terms of the number of errors found in the code, plus any other quality measures that you would like to use. In this way, you generate only those measures that are related to the goal. Notice that, in many cases, several measurements may be needed to answer a single question. likewise, a single measurement may apply to more than one question; the goal provides the purpose for collecting the data, and the questions tell you and your project how to use the data.

4.3 Other models : quality models

Quality models aims at capturing the composite characteristics and their relationship in order to measure quality. Among them, the McCall [32]and Boehm [9]software quality model propose a decompositional approach.

Since 1992 ISO [28] proposes the *Software Product Evaluation : Quality Characteristics and Guidelines for their use* also know as ISO9126.

Chapter 5

Function Points

5.1 Goals of the Function Points

In 1979, Allan J. Albrecht proposed the first function point's model and analysis method called *Function Point Analysis*[1].[1]

This method's goals were to measure achievement and refine valuation. Albrecht proposes the three following definitions for the function points :

- Function points are a measurement of the software product based on the user's function information treatment.
- Function points measure software by counting number of functionality of the information's treatment associated with external and control data, output and files types.
- This particular treatment is adjusted for the global function of information treatment by applying an adjustment based on the software characteristics.

So function points are essentially based on the software's number of functionality proposed to the user. The goal was to obtain a technique to measure productivity among different IBM's projects from 1974 to 1978. These projects had been developed with different programming languages and tools. So the objective was to provide a fitted method to measure services provided to the users.

5.2 History and evolution of function points

Albrecht began his research on this topic in the 70's. In the same time, Tom DeMarco has also leading research on the same topic. His results are quite the same in the concepts but not on the form. DeMarco's function points have been recently used as basis for new research like : FFP (*Full Function Points*), COSMIC-FFP (*Common Software Measurement International Consortium - Full Function Points*)[16].

[1] A large part of the content of this chapter comes from the following thesis :[35]

In November 1983 Symons published a simplified version of Albrecht's function points called *Mark II function points*.[45]

In 1984 IBM proposed a major review of the function points counting rules. They added a evaluation procedure to assess complexity. This method became the basic one to count function points taken by the IFPUG (*International Function Points User Group*[2]). The function point's success and expansion and the IFPUG creation contributed to the normalization of the function points measurement method.

5.3 Reliability of the function point's method

Navlaka proposes two fundamental rules that must be respected by measurement method [34]:

- Correctness : from the same data and rules, the same results must always be obtained.
- Repetitiveness : it doesn't matter the person who makes the measurement, results must always be exactly the same at different time.

According industrial experiments the observed accuracy is more or less 95%.[48]

[2]http://www.ifpug.org

Chapter 6

Object Oriented Measurement

6.1 Introduction

This section present several concepts on Object Oriented metrics. The importance of the current research lead on this subject conduct us to devote it a particular chapter.

6.2 Size measures

Object-Oriented systems generally grow in size between requirements analysis and the testing phase. So different research have been done on this topics. Pfleeger used objects and methods as a basic size measurements which is more accurate than COCOMO according to commercial applications [38, p.294].

An other method has been developed by Lorenz and Kidd [31]. They defined nine aspects of size that reflect how the class characteristics affect the product. They propose the following aspects :

Number of scenario scripts (NSS) : It's the number of scenario scripts counted in the use cases. This measure is correlated with application size and the number of tests. NSS mainly allow to predict development and testing efforts.

Number of key classes : This measure evaluate the high-design effort.

Number of support classes : This measures evaluates the low-level design.

Average number of support classes per key class : This measure gives an idea of the system's structure.

Number of subsystems : This one provide more information on the system's structure.

Class size : This measure include the number of operations and attributes.

Number of operations overridden by a class : Allow to evaluate inheritance effects.

Number of operations added by a subclass : Measures also the inheritance effects.

Specialization index

6.3 Design measures

Chidamber and Kemerer have also provide a suite of metrics for object-oriented developments [15]. They focuse their work more on design than on size so they complement the Lorenz and Kidd's method. They focuse on the coupling between objects, the response of a class and the lack of cohesion in methods[38, p.297].

They calculate weighted methods per class in order to measure complexity. They also define a class's depth of inheritance (It's the maximum length of the path in the hierarchy from the class to the root of the inheritance tree). So more deeper is a class in the hierarchy, more methods are inherited by this class.

Similarly, the number of children is the number of immediate subclasses subordinated to the given class.

Chapter 7

Collect, store, analyze and comment metrics

7.1 The data

7.1.1 Data properties

Before analyzing the data collection process there are several points to clarify. Data that will be collected must satisfy several essential properties [22]:

Correctness The data were collected according to the exact rules of definition of the metrics. For example, if comments are not supposed to be included in the lines of codes count, then a check for correctness assures that no comments were counted.

Accuracy This property refers to the differences between the data and the actual value. For example, time measurement will be less accurate on an analog clock than on a digital one.

Precision It deals with the number of decimal places needed to express the data.

Consistent In fact data must be consistent from one measuring device or person to another, without large differences in value.

Time-Stamped We must know exactly when data has been collected in order to allow comparison.

Replicated Last fundamental property hat assumes that the data definition must be very accurate to allow other person to replicate the same measurement.

7.2 Data collection

There are two ways to collect data : the manual and the automatic. Manual collection often conduct to bias, error, omission and delay. The automatic data collection is preferable but often more difficult to implement. Fenton gives the following guidelines to collect data :

- keep procedures simple;
- avoid unnecessary recording;
- train staff in the need to record data and in the procedures to be used;
- provide the results of data capture and analysis to the original providers promptly and in a useful form that will assist them in their work;
- validate all data collected at a central point.

Data collection forms are interesting because they provide a frame to collect data. But this form must be self-explanatory. [1] The GQM method previously explained also provides methodology to collect data.

The data collection must be planned as ordinary project which is linked to other projects to measure.

The collected data could be stored in database to allow further manipulations[22].

7.3 Analyze and comments

Data analysis and comment implies statistic methods and tools. They must be used in an appropriate way. According to Fenton *Data sets of software attributes values must be analyzed with care, because software measures are not usually normally distributed*[22, p.235] There are different techniques that address a wide variety of situations. Fenton gives the following advice :

- describe a set of attribute values using box plot statistics (based on median and quartiles) rather than on mean and variances;
- inspect a scatter plot visually when investigating the relationship between two variables;
- use robust correlation coefficients to confirm whether or not a relationship exists between two attributes;
- use robust regression in the presence of atypical values to identify a linear relationship between two attributes, or remove the atypical values before analysis;
- always check the residuals by plotting them against the dependant variable;
- use Tukey's ladder to assist in the selection of transformations when faced with non-linear relationships;
- use principal component analysis to investigate the dimensionality of data sets with large numbers of correlated attributes.

In fact, the most important point, underlined by Fenton is the correlation between the choice of the analysis technique and the goals of the investigation. In this way you can support or refute the hypothesis you are testing.

[1] For recent experiences of web-based data collection see [41].

Chapter 8

Glossary

Attribute (An) is a feature or property of an **entity**.Typical attributes include the area or color (of a room), the cost (of a journey), or the elapsed time (of the testing phase). [22]

Class is a particular set of entities. In software there are three classes : processes, products and resources.

COCOMO *COnstructive COst MOdel*. software cost estimation model developed by Boehm.

Entity (An) is an object (such a person or a room) or an event (such a journey or testing phase of a software project) in the real world. [22]

Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules. [22]

Bibliography

[1] Albrecht, A. Measuring application development. In *Proceedings of IBM Applications Development Joint SHARE/GUIDE Symposium* (1979), pp. 83–92.

[2] Albrecht, A., and Gaffney, J. Software Function, source Lines of Code, and Development Effort Prediction : A Software Science Validation. *IEEE Transaction On Software Engineering SE-9*, 6 (1983), 639–648.

[3] Archer, C., and Stinson, M. Object-oriented software measures. Tech. Rep. ESC-TR-95-002, 1995.

[4] Basili, V. R., and Rombach, H. D. The TAME project: Towards improvement-oriented software environments. *tose 14*, 6 (June 1988), 758–773.

[5] Basili, V. R., and Weiss, D. M. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering 10*, 6 (Nov. 1984), 728–738.

[6] Bennett, W. R., and Jr. Predicting software system development effort very early in the life-cycle using idef0 and idef1x models.

[7] Bieman, J., and Kang, B. Measuring design-level cohesion. *IEEE Transactions on Software Engineering 24*, 2 (1998), 111–124.

[8] Boehm, B. Metrics-based feedback cycles for software life-cycle management and process improvement.

[9] Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., Macleod, G., and Merritt, M. *Characteristics of Software Quality.* TRW Series of Software Technology. North-Holland, 1978.

[10] Briand, L., Daly, J., Porter, V., and Wst, J. A comprehensive empirical validation of product measures for object-oriented systems, 1998.

[11] Briand, L., Morasca, S., and et al. An operational process for goal-driven definition of measures.

[12] Briand, L. C., Differding, C., and Rombach, H. D. Practical guidelines for measurement-based process improvement. *Special issue of International Journal of Software Engineering & Knowledge Engineering* (1997).

[13] Briand, L. C., Morasca, S., and Basili, V. R. Property-based software engineering measurement. *Software Engineering 22*, 1 (1996), 68–86.

[14] Chidamber, S. R., Darcy, D. P., and Kemerer, C. F. Managerial use of metrics for object-oriented software: An exploratory analysis. *Software Engineering 24*, 8 (1998), 629–639.

[15] Chidamber, S. R., and Kemerer, C. F. A Metric Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering 20*, 6 (June 1994), 476–493.

[16] Desharnais, A. A. J.-M. Measurement manual. version 2.1, 2001.

[17] Emam, K. E. A methodology for validating software product metrics.

[18] Fenton, N., Krause, P., and Neil, M. Software Metrics: Uncertainty and Causal Modelling. In *Proc 2nd European Software Measurement Conference (FESMA'99)* (2001). EuroSPI conference,Limerick Institute of Technology, Limerick, 10th-12th October 2001.

[19] Fenton, N., and Neil, M. Software Metrics and Risk. In *Proc 2nd European Software Measurement Conference (FESMA'99)* (1999), pp. 39–55.

[20] Fenton, N., and Neil, M. Software metrics: roadmap. In *ICSE - Future of SE Track* (2000), pp. 357–370.

[21] Fenton, N., and Ohlsson, N. Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software Engineering 26*, 8 (Aug. 2000), 797–814.

[22] Fenton, N., and Pfleeger, S. L. *Software Metrics - A Rigorous and Practical Approach*, 2 ed. International Thomson Computer Press, London, 1996.

[23] Florac, W. Software quality measurement:a framework for counting problems and defects. Technical Report ESC-TR-92-022, CMU, 1992.

[24] Goethert, W., and Hayes, W. Experiences in implementing measurement programs. CMU/SEI, 2001.

[25] Goodman, P. *Practical Implementation of Software Metrics*. McGRAW HILL, New-York, 1993.

[26] Hitz, M., and Montazeri, B. Measuring Product Attributes of Object-Oriented Systems. In *Proc. 5th European Software Engineering Conf. (ESEC 95)* (Sitges, Spain, 1995), W. Schafer and P. Botella, Eds., vol. 989, Springer-Verlag, Berlin, pp. 124–136.

[27] Idri, A., Kjiri, L., and Abran, A. COCOMO Cost Model Using Fuzzy Logic. In *7th International Conference on Fuzzy Theory & Technology* (2000). Atlantic City, New Jersey.

[28] ISO. Information technology - software product evaluation - quality characteristics and guide lines for their use. iso/iec 9126, 1991.

[29] Lamb, D., and Abounader, J. Data model for object-oriented design metrics, 1997.

[30] Lewerentz, C., and Simon, F. A product metrics tool integrated into a software development environment. In *ECOOP Workshops* (1998), pp. 256–260.

[31] Lorenz, M., and Kidd, J. *Object-Oriented Software Metrics: A Practical Guide*. Prentice-Hall, 1994.

[32] McCall, J. A., Richards, P. K., and Walters, G. F. Factors in software quality, volume I: Concepts and definitions of software quality. Tech. Rep. RADC-TR-77-369, vol. I, II, III, Rome Air Development Center, Griffiss AFB, Rome, NY 13441-5700, July 1977. Available from Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145, order number AD-A049 014.

[33] Mills, E. Software metrics. sei curriculum module. CMU/SEI, 1988.

[34] Navlaka, J. Software productivity metrics : some candidates and their evaluation. In *proceedings of National computer Conferences* (1986), pp. 69–75.

[35] Ndagijimana, S. Etude de la mesure des points de fonction : application de la méthode de mesure fonctionnelle cosmic-ffp et analyse de la documentation fonctionnelle. Master's thesis, University of Namur, rue Grandgagnage, 21 - 5000 Namur(Belgium), June 2002.

[36] Park, R. Software size measurement: A framework for counting source statements, 1992.

[37] Pfleeger, S., Jefferey, R., and Kitchenham, B. C. B. Status report on software measurement. *IEEE Software 14* (march/april 1997), 33–43.

[38] Pfleeger, S. L. *Software Engineering: Theory and Practice*. Prentice-Hall, Upper Saddle River, NJ, 2001.

[39] Poels, G. Towards a size measurement framework for object-oriented specifications.

[40] Poels, G., and Dedene, G. Complexity metrics for formally specified business requirements, 1997.

[41] Rob Pooley, D. S., and Christie, D. Collecting and analyzing web-based project metrics. *IEEE Software 19*, 1 (Jan. 2002), 52–58.

[42] Robert Park, W. G., and Florac, W. Goal-driven software measurement a guidebook. CMU/SEI, 1996.

[43] Robert Park, W. G., and Webb, J. T. Software cost and schedule estimating: A process improvement initiative. CMU/SEI, 1994.

[44] Strike, K., Emam, K. E., and Madhavji, N. H. Software cost estimation with incomplete data. *Software Engineering 27*, 10 (2001), 890–908.

[45] Symons, C. Software ii fpa, sizing and estimating mark wiley series. *Software Engineering Practice* (1991).

[46] University, V. C. Cocomo ii model definition manual.

[47] W. Goethert, E. B., and Busby, M. Software effort & schedule measurement: A framework for counting staff-hours and reporting schedule information. Tech. Rep. ESC-TR-92-021, CMU/SEI.

[48] Whisehunt, C. How to implement function points both ways (right and wrong) and still survive. In *8th QAI international conference on Measuring* (1990), pp. 119–129.