SOFTWARE QUALITY MEASUREMEN AND METRICS (REVIEW SOME ARTICLES)

Ramadiani

Program Studi Ilmu Komputer FMIPA Universitas Mulawarman
Email : ilkom.ramadiani@gmail.com

ABSTRACT

Software development processes have less variability than the projects or products upon which they are applied, a feature that has provided the source for considerable debate on how to capture and describe software processes for the purposes of understanding or inclusion within Software Engineering Environments. The software measurement is potentially very diverse in nature, there are four basic stages of formulation, collection, analysis and interpretation The successful measurement programmes must support the collection of cost, duration and quality values. There are some varian measurement metrics such as; performance analysis, deliverable analyses, quality objectives, schedule analyses, effort analyses etc.

Kata Kunci: *Software Quality, Measurement, Matric.*

INTRODUCTION

Why should software measurement be problematic? The answer, in brief, is because software engineering is a highly complex process producing highly complex products. Moreover, each project and its products tend to be something of “one off” in nature, a point highlighted by Schneidewind as a difficulty in validating metrics even when a defined validation methodology is used.

However, software development processes have less variability than the projects or products upon which they are applied, a feature that has provided the source for considerable debate on how to capture and describe software processes for the purposes of understanding or inclusion within Software Engineering Environments. Kitchenham points out that successful measurement programmes must support the collection of cost, duration and quality values to ensure that measuring one aspect, for example cost, does not cause the problem to migrate to another aspect such as quality.

EVALUATION

The general shortage of effective metric methods to provide advice and guidance for the usage of measurement is a severe constraint on management's ability to retain effective control. As has been suggested, the drive for improvement in the usage of software metric is now focusing upon the application of measurement methods. Methods guide developers on what to do next and limit the available choices to a structured process of manageable steps. They are not prescriptive nor do they provide details about how each step should be carried out.

There are some template guides for engineers to define system attributes, using a standard set of headings e.g. scale, date, test, worst case, from which terminology can be clarified and measures established. These features and supportive principles have value for motivating staff to use objective measurement as opposed to subjective measurement based upon checklists of software attributes. While these are important aspects of the measurement process they do not on their own constitute a measurement method, they support initial definition of a set of standard metrics.

In order to help assess the support for the measurement process provided by the various measurement methods it is helpful to break down the software measurement process into its constituent stages (as illustrated in Figure 1). Although software measurement is potentially very diverse in nature, thereare still only four basic stages of formulation, collection, analysis and interpretation with validation as an on-going activity throughout.

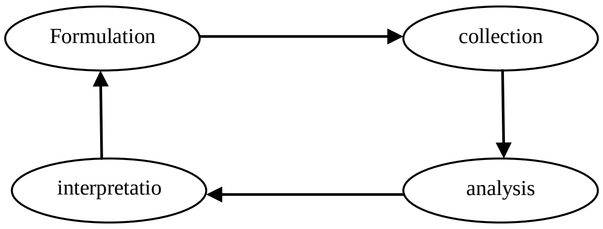


Figure1: The Measurement Process

Formulation involves setting measurement goals, identifying the metrics required and defining them in terms of the particular measurement environment. Lines of code seldom has exactly the same meaning between different software organisations: one environment might count delimiters, another number of carriage returns imd so forth. The diversity of measurement perspectives e.g. customer, developer, manager and researcher also need to be catered for by this stage.

Collection is concerned with setting up the actual measurement processes and any tool development that might be necessary. It must also address the training and education of those involved in the process, il it is to run smoothly software developers must know what to collect, when, where and how.

Analysis is the stage dealing with measurements once they have been obtained. Statistical analysis may be important to help uncover pattems, discriminate between software components and identify anomalies. The results can be used to provide feedback into the software process at all levels from individual through team to organisation.

Interpretation is the assignment of meaning to the collected values, determining the cause(s) of the values, distinguishing which cause was responsible and identifying the appropriate corrective action to be &en. The stage is problematic due to each value having many causes.

Last, and frequently overlooked, is the issue of *validation.* Basically, measurers continually need to ask themselves whether the measurement is a true, or adequate, representation of whatever atlribute they believe is being captured. Validation should take place throughout the measurement process.

SOFTWARE QUALITY

In the field of software engineering, the term “metrics ” is used in reference to multiple concepts; for example, the quantity to be measured (measurand 1), the measurement procedure, the measurement results or models of relationships across multiple measures, or measurement of the objects themselves.

In the software engineering literature, the term was, up until recently, applied to:

- measurement of a concept: e.g. cyclomatic complexity [McCabe 1976],
- quality models: e.g. ISO 9126 — software product quality, and estimation models: e.g. Halstead ’ s effort equation [Halstead 1977], COCOMO I and II [Boehm, 1981, 2000], Use Case Points, etc.

Software engineering definition from IEEE Computer Society:

> “(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1) ” [IEEE 610.12]

A measurand is defined as a particular quantity subject to measurement; the specification of a measurand may require statements about quantities such as time, temperature, and pressure [VIM 2007]. In the scientific fields, including engineering, as well as in others, like business administration and a significant number of the social sciences, measurement is one of a number of analytical tools. Measurement in those sciences is based on a large body of knowledge built up over centuries, even millennia, which is commonly referred to as “ metrology ” .

Software Engineering Metrics: What Do They Measure and How Do We Know?
by Cem Kaner, Senior Member, IEEE, and Walter P. Bond (2004),

- Construct validity starts with a thorough analysis of the construct, the attribute we are attempting to measure. In the IEEE Standard 1061, direct measures need not be validated.

- "Direct" measurement of an attribute involves a metric that depends only on the value of the attribute, but few or no software engineering attributes or tasks are so simple that measures of them can be direct. Thus, all metrics should be validated

The research continues with a framework for evaluating proposed metrics, and applies it to two uses of bug counts. Bug counts capture only a small part of the meaning of the attributes they are being used to measure. Multidimensional analyses of attributes appear promising as a means of capturing the quality of the attribute in question.

Defining Measurement

- Fenton and Pfleege provide a concise definition:
- 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. [p. 28]
  - Standard 1061 (section 4.5) lays out several interesting validation criteria, which we summarize as follows:
  - Correlations
  - Consistency
  - Tracking
  - Predictability
  - Discriminative power
  - Reliability

Direct Measurement

- The IEEE Standard 1061 answer lies in the use of direct metrics. A direct metric is "a metric that does not depend upon a measure of any other attribute."

- Direct metrics are important under Standard 1061, because a direct metric is presumed valid and other metrics are validated in terms of it ("Use only validated metrics (i.e. either direct metrics or metrics validated with respect to direct metrics)")

Some common derived metrics in software engineering are :

a) Programmer productivity (code size/ programming time)
b) Module defect density (bugs / module size)
c) Requirements stability (number of initial requirements / total number of requirements)
d) System spoilage (effort spent fixing faults / total project effort)

Standard 1061 offers MTTF (Mean Time To Failure) as an example of a direct measure of reliability:

- Mean
- Time
- To
- Failure

But if we look more carefully, we see that this measure is not direct at all. Its values depend on many other variables. As we'll see, this is true of many (perhaps all) software engineering metrics

Consider the four examples of direct measurement provided by Fenton & Pfleeger:

- Length of source code (measured by lines of code);
- Duration of testing process (measured by elapsed time in hours);
- Number of defects discovered during the testing process (measured by ounting efects)
- Time a programmer spends on a project (measured by months worked). [7, p. 40]

The Evaluation Framework

To evaluate a proposed metric, including one that we propose, we find it useful to ask the following ten questions:

1) What is the purpose of this measure?
2) What is the scope of this measure
3) What attribute are we trying to measure?
4) What is the natural scale of the attribute we are trying to measure
5) What is the natural variability of the attribute?
6) What is the metric (the function that assigns a value to the attribute)?
7) What is the natural scale for this metric?
8) What is the natural variability of readings from this instrument?
9) What is the relationship of the attribute to the metric value?
10) What are the natural and foreseeable side effects of using this instrument?

Applying the Evaluation Framework

- Bug counts are chosen because they are ubiquitous. For example, in Mad About Measurement, Tom DeMarco says: "I can only think of one metric that is worth collecting now and forever: defect count.“
- Bug counts have been used for a variety of purposes, including:
  - Private, personal discovery by programmers of patterns in the mistakes they make.
  - Evaluation (by managers) of the work of testers (better testers allegedly find more bugs) and programmers (better programmers allegedly make fewer bugs).

In this research, the discussion to two attributes, that are popularly "measured" with bug counts.

a. Quality (skill, effectiveness, efficiency, productivity, diligence, courage, credibility) of the tester. Whatever the variation, the idea is that more bugs indicate better testing (and fewer bugs indicate worse testing).
b. Status of the project and readiness for release. One of the key release criteria for a project is an acceptably low count of significant, unfixed bugs
c. A group of test managers has been developing this approach for their use, and many of them are now experimenting with

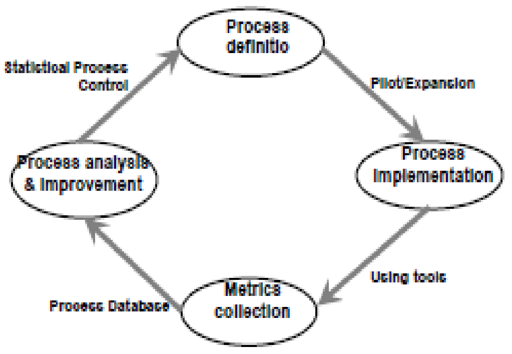
it, to the extent that they can in their jobs. There are too many simplistic metrics that don't capture the essence of whatever it is that they are supposed to measure. There are too many uses of simplistic measures that don't even recognize what attributes are supposedly being measured. Starting from a detailed analysis of the task or attribute under study might lead to more complex, and more qualitative, metrics, but we believe that it will also leads to more meaningful and therefore more useful data.

A Software Metrics Case Study

- a case study on gathering process and performance metrics, which is useful in improving software engineering processes.
- The metrics gathered by a large multi-project-team, software engineering outsourcing company, staffed with over 500 engineers, with clients throughout USA, Asia and Europe.
- Metric data has been collected over a period of five years, across hundreds of projects, with an average project size of 60 man-months, and team sizes from 6-12 members.

PROCESS IMPROVEMENT CYCLE
(AGILIS SOLUTIONS 2009)

Table 1. Process Improvement Cycle



Performance Analysis

Table 2. Performance

Metric	Unit	Planned	Actual	Deviation (%)
Plan start date	DD-MMM-YY	29-Jul-04	29-Jul-04	0
Plan end date	DD-MMM-YY	29-Oct-04	N/A	15.22
Duration	Days	92	106	15.22
Max team size	Persons	12	14	16.67
Effort usage:	Person-day	420	245.98	-41.43
Development	%	60	56.84	-5.27
Management	%	7	5.36	-23.41
Quality	%	33	37.8	14.54
Product size	UCP	120.06	93.7	-21.96
Productivity	UCP/pd	0.29	0.38	33.28
Efficiency	USD/pd	0	0	N/A

- Team size increased over plan from 12 to 14
- Actual effort is 43.43% below plan

Deliverable Analyses

Table 3. Deliverables

Deliverable	Committed date	Actual date	Status	Deviation (%)
Claims Work-prototype	26-Aug-04	26-Aug-04	Accepted	0
Claims Work-source code and/or executable	15-Sep-04	4-Oct-04	Accepted	20.65
Claims Work-updated	28-Sep-04	N/A	Cancelled	N/A

- Indicator of where and when problem may exist in anya project
- 20.26% changes deviation equalizes the differences
- Data aggregated and compare with performance norms

Quality Objectives

Table 4. Quality Objectives

Name	Unit	Norm	Targeted Value	Actual Value	Deviation (%)
Timeliness	%	85	100	80	-20
Requirement Completeness	%	95	100	100	0
Leakage	WDef/UCP	0.3	0.3	0.17	-43.08
Customer Satisfaction	Point	90	90	N/A	N/A
Correction Cost	%	9	7	9.81	40.11
Process compliance	NC	3	3	2	-33.33
Response Time	Hour	24	N/A	0	N/A
Translation cost	%	N/A	N/A	0	N/A

- 20% Improvement in times raises some questions.
- 43.08% Leakage will cause someone to ask if quality assurance team actually caught the defect.
- 40.11% correction cost will definitely raise the question

Schedule Analyses

Table 5. Project Schedule

Stage	Is on time	Planned duration	Actual duration	Duration deviation (%)
Initiation	Yes	13	13	0
STAGE-Aug04	Yes	34	34	0
STAGE-SEP04	Yes	34	34	0
Termination	N/A	8	N/A	N/A

Table 6. Product Schedule

Product name	Planned release date	Actual release date	Schedule deviation (%)
Claims Work-SRS doc	21-Aug-04	21-Aug-04	0
Claims Work-prototype	26-Aug-04	26-Aug-04	0
Claims Work-design	1-Sep-04	1-Sep-04	0
Claims Work-Test case & data	3-Sep-04	3-Sep-04	0
Claims Work-source code	15-Sep-04	4-Oct-04	55.88

- Analyzing schedule is look at the macro view of the deliverable schedule
- Product represent aggregated deleverables 55.88% schedule delay in code release.

Effort Analyses

Table 7. Effort by Process

Process	Planned (pd)	Planned (%)	Actual (pd)	Actual (%)	Deviation (%)
Requirement	42	10	15.12	6.15	-63.99
Design	63	15	47.69	19.39	-24.31
Coding	138.6	33	95.56	38.85	-31.05
Deployment	12.6	3	2.25	0.91	-82.14
Customer Support	21	5	2.88	1.17	-86.31
Test	84	20	55.56	22.59	-33.85
Configuration Management	4.2	1	1.62	0.66	-61.31
Project planning	4.2	1	3	1.22	-28.57
Project monitoring	25.2	6	10.19	4.14	-59.57
Quality Control	12.6	3	9.1	3.7	-27.78
Training	8.4	2	0.88	0.36	-89.58
Total	420	100	245.98	100	-41.43

- Compare the percent requirement to percent in design
- Look at the anomalies between actual and planned percentage
- Requirement took much less effort than expected

Effort Analyses

Table 8. Effort by Type

Type	Effort (pd)	Effort (%)
Study	9.75	3.96
Create	150.25	61.08
Review	15.1	6.14
Test	46.75	19.01
Correct	24.12	9.81
Total	245.98	100

- To indicate either the maturity of the development team or difficulty of a new technology.
- Too much time testing and correcting spot poor coding technique
- Handle by strengthening the review process, training personnel on coding and technique.

CONCLUSION

- There are too many simplistic metrics that don't capture the essence of whatever it is that they are supposed to measure.
- There are too many uses of simplistic measures that don't even recognize what attributes are supposedly being measured.
- Starting from a detailed analysis of the task or attribute under study might lead to more

complex, and more qualitative, metrics, but we believe that it will also leads to more meaningful and therefore more useful data.

- Decision of whether to change a work process, provide employee training, modify the estimation tool or some other action subject to judgement of the evaluator.
- The metrics collected must be specifically relevant to improving the engineering process.

REFFERENCES

[1] Alain Abra.n, 2010 Software Metrics and Software Metrology, Copyright IEEE Computer Society.

[2] Agilis Solution, 2009. Software Metrics Case Study

[3] Cem Kaner, Senior Member, IEEE, and Walter P. Bond (2004), Software Engineering Metrics: What Do They Measure and How Do We Know?, 10TH International Software Metrics Symposium, Metrics 2004

[4] IEEE, "IEEE Std. 1061-1998, Standard for a Software Quality Metrics Methodology, revision." Piscataway, NJ,: IEEE Standards Dept., 1998.

[5] Linda Westfall (2005), 12 Steps to Useful Software Metrics, The Westfall Team, westfall@idt.net

[6] N. E. Fenton, 1999. "Software Metrics: Successes, Failures & New Directions,"presented at ASM 99: Applications of Software Measurement,S a n J o s e , C A , http://www.stickyminds.com/s.asp?F=S2624_ART_2

[7] N. E. Fenton and S. L. Pfleeger, "Software Metrics: A Rigorous and Practical Approach," 2nd Edition Revised ed. Boston: PWS Publishing, 1997.