A framework for software project metrics

Terence L. Woodings, Gary A. Bundell

Abstract

Software Engineering is a discipline which is making exceptional progress in the development of new paradigms, methods, tools and techniques for the production of information systems. A weakness is that the academic development of the theory underlying these advances is often "out of synchronisation" with current usage. The use of metrics is a key factor in the establishment of a scientific basis for Software Engineering and the reduction of the gap between academics and practitioners. Systematic measurement is necessary for the formal modelling and evaluation of the new techniques and their optimal use within industry. A number of approaches (e.g. the Software Factory, CMM, Bootstrap, SPICE, GQM, Balanced Scorecard), have been advocated for the systematic design and introduction of software metrics for the purposes of process improvement and capability assessment in an organisation.

This paper defines a taxonomy of software metrics which is derived from the needs of users, developers and management. The properties of the classifications are discussed. Rules are then derived which generate a new set of metrics from an existing class. The rules complement standard approaches by focusing attention on user driven aspects of an organisation's measurement programme. They do this by directly linking product characteristics with organisational improvements. The method also has promise as an approach for validating process models and metrics.

Examples of the theoretical and pragmatic use of the taxonomy are provided. The paper concludes with a discussion of useful applications.

Keywords

Software Metrics, Software Engineering Management, Capability Assessment.

1. Background

Software Engineering is an immature discipline which has been making exceptional progress in the development of new paradigms, methods, tools and techniques for the production of information systems. Unlike other mature engineering disciplines, there is a demonstrable inability to estimate, plan and control projects. A hurdle to improving the respectability of the field is that the academic development of the theory underlying advances is often "out of synchronisation" with current usage. In some areas, such as formal verification methods, academia has had considerable trouble convincing industry to close the gap. In others, such as the human-computer interface, developers race ahead with new and practical concepts without waiting for theoretical justification.

This lack of synchronicity should be a cause for real concern. The use of metrics is now a key factor in the establishment of a scientific basis for Software Engineering. Systematic measurement is necessary for the formal modelling and evaluation of the new techniques and their optimal use within industry. Without metrics, it is difficult to convince industry of the advantages of adopting new academic advances such as proofs of program correctness. In medicine, new drugs are subject to rigidly controlled clinical trials. However, in information systems development, the selling of new techniques without prior assessment by an independent institution leaves it little better than a pseudo science.

The early, pioneering efforts to implement software metrics for process improvement [1, 2, 3] were often based on ad-hoc measurement programmes, which either concentrated on

some key aspect of the system (such as defect counts) or came as a byproduct of other activities (such as counting lines of code during compilation). The concept of the Software Factory [4, 5, 6] was based, in part, on the philosophy of measuring every aspect of the development process in order to discover those factors of central importance.

Reaction to the limited success of these approaches and attempts to put the work on a more disciplined and scientific footing [7] led to top-down approaches [8, 9], with metrics programmes being derived from (and thus aligned with) organisational goals. This approach was further encouraged by such influential work as the Goal-Question-Measure (GQM) paradigm [10], the Capability Maturity Model (CMM) [11] and the Balanced Scorecard [12, 13].

The SPICE initiative, which led to the ISO 15504 [14] scales for organisation assessment, has provided a focus for work on software process metrics. It supplies a foundation for the measurement of an organisation's processes so that new assessments may be mapped to a standard (0 to 5) point scale.

However, there is still considerable concern regarding the potential for (and consequences of) misalignment of organisational strategy and project measurement [15]. In a recent paper, Boehm and Basili noted that "the primary difficulty in determining such metrics is that software projects are effectively unrepeatable in practice." They go on: "This changing nature means that most traditional measures of software productivity are increasingly irrelevant" [16].

However, there is a complementary bottom-up approach possible, where the measurements are driven by user needs and product characteristics. Quality Function Deployment [17], with its use of the 'Voice of the Customer', is an example of this - although measurements linked to QFD have been limited so far. In a similar vein, Gilb challenged: "We must stop messing with 'internal' software metrics such as complexity and function points and learn really powerful software metrics based on final-product, customer perceived results such as adaptability, availability, reliability, maintainability, security, portability and performance." [18]

Despite these difficulties with current metrics, it is essential that software engineering be placed on a sound basis through scientific measurement. Approaches to a formal theory for software metrics have been provided by a number of researchers [8, 9, 19, 20]]. A framework to facilitate understanding of the complementary uses of the various types of metrics and how they relate to each other is now necessary. In particular, this paper focuses on the dynamic performance of metrics - how their accuracy, precision and utility changes over the duration of a project, the life of a product or the strategic plan of an organisation.

2. Use of degrees of influence

One useful approach to the definition of a framework for the study of metrics is to examine the influence of classes of metrics upon each other. Distinguishing between Product and Process metrics has now become a well established practice (see for example Schulmeyer and McManus). Just as the properties of the product (the information system) are determined by the quality and effectiveness of the development process, so it is possible to extend this approach to other levels of the software industry. The attributes of the product and supporting services influence the mode of usage and degree of acceptance by the customer. Similarly the definition, extension of, and compliance to the process are determined by the management and staff of the software development organisation. In turn, there are a multitude of factors affecting the organisation (the economic situation, the marketplace, availability of resources, government policies and so on), but these drivers are generally external and outside the control of the software producer. A simple model of these hierarchical levels of influence is depicted in figure 1.

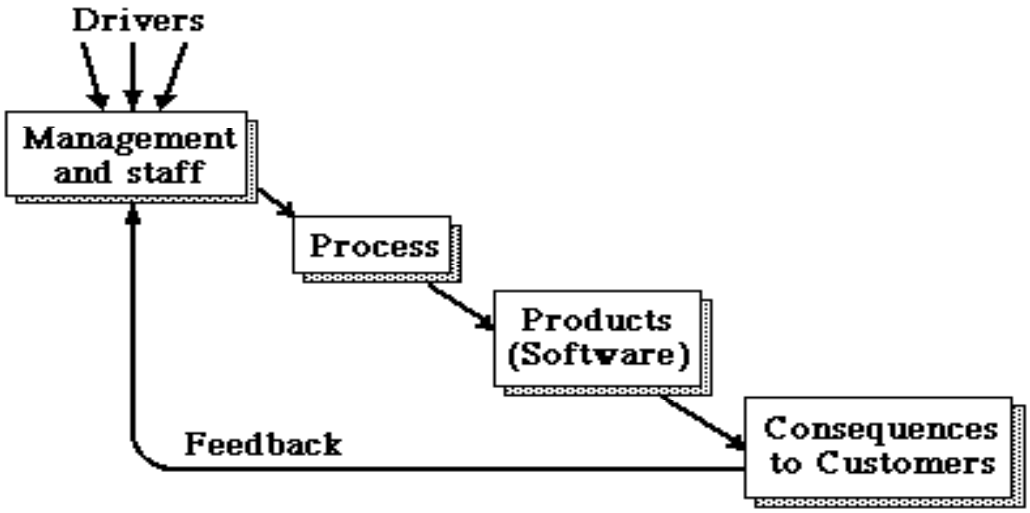

Figure 1 A hierarchy of levels of influence in software production

To state the obvious, in order to improve some aspect of software production and usage, it is necessary to modify the class of factors immediately above it in the hierarchy. As measurement is a prerequisite for an understanding of control and improvement, it is useful to attach a class of metrics to each level.

Returning to Gilb's previously quoted challenge, if there is to be an improvement in "The Only Thing That Matters" [21], the acceptance of information systems by the end-user, then it is necessary to trace and measure the cause and effect path of specific software usage attributes up through the hierarchy. This leads to a bottom-up approach to metric definition, which is complementary to the usual ISO, CMM or GQM top-down methods.

3. A taxonomy of software metrics

The class of metrics associated with each level of the hierarchy may be further divided according to usage (for example, internal versus external publication), degree of resolution (statements, objects, modules or systems) or scale. A simple taxonomy is given in table 1. This is taken from [22] which discusses and supplies references to the individual examples. Alternative discussions of the design and use of these (in some cases historical) metrics can be found in [7, 8].

This taxonomy has been found to be useful in focussing the attention of practitioners on the broad range of uses of metrics during software engineering courses.

Table 1 A simple taxonomy of metrics with examples

Measures of Customer satisfaction
- Proportion of Requirements met
- Amount of usage of new system
- Numbers of customers purchasing system
- Qualitative preference measures using Likert scales

Product Metrics
- System Size (e.g. Halstead's Volume, Function Points, KLOC)
- Complexity of design at various levels of resolution:
  - Code Statements (e.g. Halstead)
  - Control flow (e.g. McCabe's Cyclomatic metric)
  - Component (e.g. Troy & Zweben, Kafura)

- Complete System (e.g. McCabe & Schulmeyer)
- Reliability - based on MTBF or statistical models
- Quality attributes - McCall's 'ilities' (based on ISO 9126 or COQUAMO templates)

Process Metrics

- Risk Assessment - given as probability x cost of impact
- Estimation accuracy
  (predicted versus actual delivery dates)
- Open question clearance rate
  (may be used to measure requirements creep)
- Defect occurrence rates and propagation
  (defect estimates versus historical data)
- Inspection and Test Efficiency
  (proportion of defects found by a set of tests)

Organisation Metrics

- Profitability (counting the dollars)
- Staff job satisfaction (shown by staff turnover, sick leave)
- Organisational Capability measured against:
  - Previous performance
    (e.g. process improvement over time)
  - Industry average performance
    (e.g. Capers Jones, ASMA)
  - An arbitrary and absolute standard
    (e.g. ISO 9001, CMM, SPICE, Baldrige)
  - Superior organisations
    (e.g. Benchmarking)
  - The theoretical state of best practice possible
    (e.g. Woodings [23])

Driver Metrics

- Management Commitment to improvement
- External pressures for change (e.g. % of competitors with better measurements)

4. The inter-relationship of classes of software metrics

As a quality improvement technique, software managers should be encouraged to explore the likely results (perhaps using Ishikawa cause & effect diagrams) of focussing staff attention on specific issues. For example, client satisfaction may be a function of 'Proportion of Requirements Met'. Over the life of a system, the ratio will depend on Change Requests received and implemented. One is outside the control of the software producer (although it may be predicted using patterns from previous projects), but the other is determined by productivity. This, in turn depends on resources, skills, tools, techniques, etc. These, in their turn, depend on the organisation's size, capability and management. Thus a trivial metric, 'Proportion of Requirements Met', which is simple to apply, can provide some useful insights into the performance of various levels of the organisation. (Note, the important pre-requisites for setting up successful metrics programmes are widely covered in the literature [e.g. 8] and will not be discussed here.)

Differences in a series of observations of a metric for a particular software property, over time or a number of project milestones, will provide the data for empirical prediction of future performance or the validation of theoretical models.

Clearly, in order to improve a software product by the application of some process, then *the rate of change of the product's characteristics is a measure of the effectiveness of the process.*

Similarly, an organisation's future competitiveness will be indicated by the rate of improvement of its processes.

In other words, modelling the differences between observations leads to a 'natural' evolution from product to project to organisation metrics. Most Software Engineering texts (see, for example, [24]), identify and discuss the use of specific metrics at various levels for software process improvement. Because they have quite different purposes, these levels have generally been studied in isolation. However, for measurements using a ratio scale, we postulate a simple and pragmatic mathematical relationship between the hierarchical levels of the taxonomy.

For any valid product metric, its derivative with respect to time is a valid process metric.

For any valid process metric, its derivative with respect to time is a valid metric for the organisation.

The authors have named these the Metrel (Metric Relationship) Rules. That is to say, when defining, modelling and validating a metric for a particular purpose, it is appropriate to also examine its rate of change, for the extra information it can provide.

Note that it is not necessary for the underlying mathematical model of the characteristic being measured to be continuously differentiable although this would be a preferred condition. These rules can provide useful insights and information using approximations based on discrete observations.

5. Use of the Metrel rules

To illustrate the use of the rules, their coverage of Moller's software measurements [25] is demonstrated. A practical example of their application is provided later in the paper.

Moller describes five of the most widely used measurements (System size, Test faults, Change requests, Departure from Schedule and Productivity), which are termed Global Metrics. Under Metrel, they may be covered in the following three linkages:

(i) The Size of a system (in thousands of lines of code (KLOC) or Function Points) is a valid and useful Product Metric.
Then the rate of production of code, (in KLOC/week), is a valid Process metric (assuming the project team is of constant size).
And the rate of change of productivity over time (in KLOC/week/week) is a valid Organisation metric.
This example is illustrated in figure 2.

(ii) The number of defects in a system is a useful and valid Product Metric.
Then the rate of insertion of defects into code per phase (the error proneness of the methodology) is a valid Process metric.
The rate of removal of defects per inspection (the test efficiency) is also a valid measure of the Process.
And the rate of improvement in test efficiency, over a series of projects, is a

valid Organisation metric.

(iii) Lateness (slippage from schedule in days) is an important and valid attribute of the Product which is highly visible to the customer.

The variation in delivery date over a series of project milestones (the accuracy of estimates and schedule) within a project is a valid Process metric. This is usually measured by the Magnitude of the Relative Error (MRE), which is a ratio of times, a unitless number. (Note that this variation is also related to requirements creep which may be estimated by the number of change requests being logged. This leads to a set of complementary metrics.)

Then the rate of change of estimation accuracy over a number of projects (the decrease in MRE/project), is a valid measure of the Organisation's determination to improve.

6. The inverse of the Metrel rules

It should be noted that the rules have a natural (although seemingly not as useful), corollary based on integration, over time, of the higher level metrics. Thus it may be possible to gain some insights into measurements by starting with the organisation or process characteristic and then working down to the product. To add a further example:

(iv) The probability of schedule slippage for any given day of a project is a valid indicator of Process risk.

Thus, the integral over the project schedule gives the expected number of days delay in getting to market and represents the waste or cost associated with that system Product. This is related to a product's failure to meet customer delivery expectations.

7. Benefits of the Metrel rules

With the use of the Metrel rules, a number of advantages and useful techniques are apparent:

First, Metrel provides an efficient method for depicting on one graph the information needed for management, staff and customers to view the success or otherwise of process improvement efforts. There is a critical need for greater visibility of project activity in order to reveal the first signs of departures from plan [26]. Kitchenham in [27] notes: "A planning process in which full information about estimates is recorded and available to interested staff helps to avoid situations in which line managers are required to 'manage' schedule and budget risks resulting from 'political' estimates." Figure 2 uses the weekly graphing of productivity rates (perhaps corrected for team size) to portray three important measures for project progress meetings.

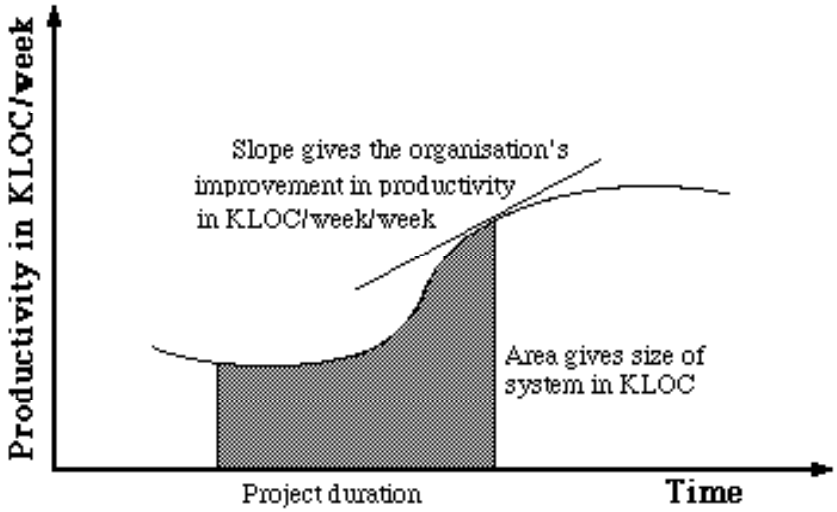

Figure 2 Three metrics portrayed as variation over time

Second, Metrel also provides a check into the way sets of software metrics are chosen. For example, stating a useful and valid unit of measurement for one level automatically determines the units for metrics above or below it. If clearly appropriate, then there is a partial confirmation of the choice.

Third, Metrel may be employed as a supplement to existing methods of metric definition and generation such as the GQM paradigm [10], the Constructive Quality Model (COQUAMO) [28] and Shepperd's Model-based approach [7]. As can be seen from the examples given above, the Metrel Rules lead naturally towards a refocussing of attention on 'impact on the customer'. From that basis, differentiation leads to succeeding higher levels of metrics being defined to improve the working methods (the quality culture) of the organisation. As customer satisfaction is the chief indicator of the future success of an organisation, this approach is a useful complement to the usual top-down methods of choosing properties to measure. It provides a direct mathematical link between product measurement and organisational improvement. Fenton and Pfleeger [8] note that "the GQM approach must be supplemented by one or more models that express the relationships among the metrics". The use of Metrel in this way is the subject of a separate paper, currently in preparation.

Fourth, Metrel is particularly important for the modelling of the software development process. It provides a mechanism for the expansion and transformation of models. The conversion of a mathematical statement for one level to represent activities at another, by means of differentiation or integration (over time or phases or projects), provides a simple but powerful extension for the analysis of the process.

8. Applications of the taxonomy and rules

The efficiency of a software test methodology may be measured by the proportion of defects removed from the product prior to installation. (Note, there are several ways of assessing the number of defects in a system - see, for example, [29].)

The model for improvement in test efficiency will be based on some research [30] into Post Implementation Reviews (PIRs). Figure 3 depicts the improvement in test efficiency from current capability c towards a desired benchmark b of 'best practice' over time t which is measured in units normalised to the average length of a project. The fraction g is the gain in efficiency per project PIR. The improvement in the process metric of Test Efficiency may be given by:

$$f(t) = b - (b - c)(1 - g)^t$$

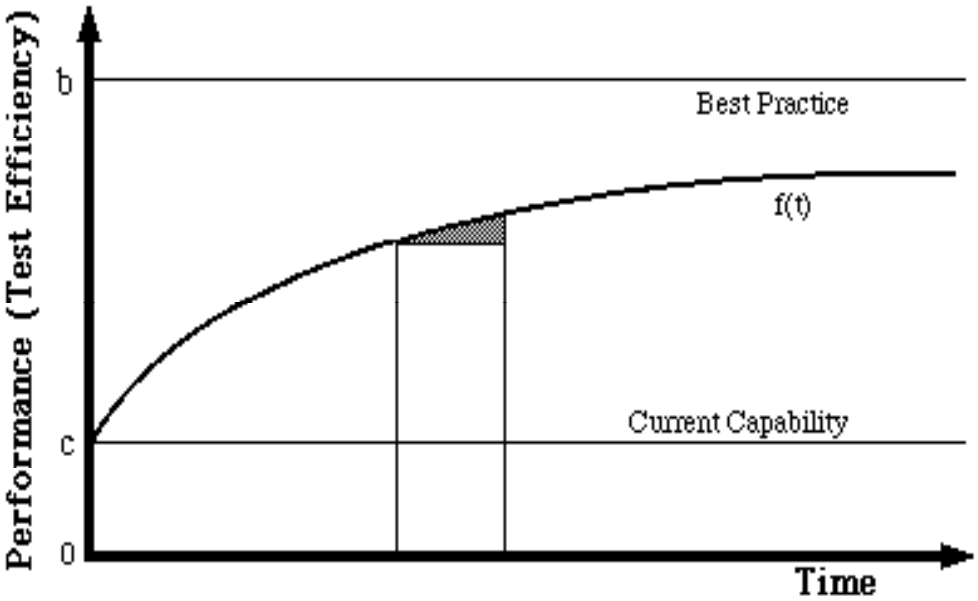

Figure 3 Improvement in capability over time, f(t)

In order to simplify the example, it is assumed that the benchmark and the existing baseline capability of the organisation are varying in much the same way and therefore the ratio c/b is considered a constant. The difference $f(t) - f(t-1)$ gives the improvement in testing due to better techniques identified from the PIR,

with $f(0) = c$ and $f(1) = c + (b - c)g$

Typical figures would be $c = 0.7$ with $b = 0.9$ as the target for test efficiency.

At the end of the first project, following a certain amount of effort expended on the initial PIR, the efficiency $f(1)$ is discovered to be 0.75 which gives

$$g = (0.75 - 0.7) / (0.9 - 0.7) = 0.25$$

This figure may be recomputed at the end of each project. The application of the Metrel Rules, states that the derivative of f with respect to time gives a valid indicator of the organisation's capacity to improve further.

$$f'(t) = -(b - c)\,(1 - g)^t \ln(1 - g)$$

Thus if the commitment of extra resources for improving test efficiency is held steady then

$$f'(t) = 0.06\,(0.75)^t$$

This figure would be useful for an annual benchmarking of the organisation Similarly, the integral of f with respect to time, over a given period, gives a measure of the defects found in projects in that interval.

$$\int f(t)\,dt = bt - (b - c)\,(1 - g)^t / \ln(1 - g)$$

To determine the expected number of defects d in the (n+1)st project, the integral from $t = n$ to $t = n+1$ is computed. This gives:

$$d(n+1) = (n + 1)b - (b - c)\,(1 - g)^{n+1} / \ln(1 - g) - bn + (b - c)\,(1 - g)^n / \ln(1 - g)$$
$$= b + (b - c)\,g\,(1 - g)^n / \ln(1 - g)$$

This is of pragmatic use in estimating the benefit of continuing with the process improvement. For this one project, the fraction of extra defects is indicated by the shaded area in figure 3 and is found to be

$$d(n+1) - f(n) \times (n+1 - n) = b + (b - c)\,g\,(1 - g)^n / \ln(1 - g) - b + (b - c)\,(1 - g)^n$$

$$= (b - c) (1 - g)^n (1 + g / ln(1 - g))$$

The project manager would be aware of the extra effort being expended on PIRs and the improved test techniques and this can be compared with the difference in cost of finding and fixing defect before rather than after installation of the system. In this example the fraction of extra defects to be found in the third project is expected to be

$$0.2 \times 0.75^2 \times (1 + 0.25 / \ln(0.75)) = 0.015$$

which appears quite small but when most new systems report defects in excess of 1000, then this is quite significant. More importantly, as many studies (for example, [31]), have suggested a cost ratio of at least five to one in not finding errors until after release, this suggests that the extra effort for PIRs and similar strategies up to 5 x 0.015 = 7.5% is well spent. That is to say, the average software house would benefit from investing ten days per person per year on improvements in testing.

It is worth noting that should the savings from better testing be reinvested to find yet further errors, thus keeping the length of the average project roughly constant, then this metric of test efficiency is similar to Hitachi's System Spoilage [32]. Over six project periods from 1976, the Hitachi metric had a gain rate of g approximately equal to 0.32, which was excellent.

To summarise, this example of the use of the Metrel Rules illustrates how managers can use this simple and pragmatic approach for quantifying process improvement.

9. Conclusion

The use of the Metrel Rules, and the corollary, leads to a number of insights into the derivation of software process models and the definition of appropriate metrics. In particular it leads to greater visibility of the many factors (and their relationships), in process improvement initiatives. Most importantly, it provides a direct linkage between the product characteristics, which impact on the end-user, and the measurements which are viewed by executives in managing a software organisation.

Current research is focused on the practical application of this rule in generating and validating useful metrics, particularly in the area of risk management. Further research will be needed on the use of this theory to optimise process improvement strategies.

With current software engineering practices exhibiting the symptoms of an immature discipline, the use of new approaches to promote measurement, modelling and optimisation of development processes is to be encouraged.

Acknowledgements

The authors appreciate the feedback of colleagues, particularly Peter Jones and Richard Thomas, in the preparation of this work. They would also like to express their thanks to the referees of a previous version of this paper.

References

[1] Conte, S D, Dunsmore, H E and Shen, V Y (1986) "Software Engineering Metrics and Models", Benjamin-Cummings.

[2] Gilb, T (1977) "Software Metrics", Winthrop.

[3] Watts, R A (1987) "Measuring Software Quality", NCC.

[4] Evans, M W (1989) "The Software Factory", John Wiley.

[5] Grady, R B and Caswell, D L (1987) "Software Metrics: Establishing a Company-Wide Program", Prentice-Hall.

[6] Johnson, J (1989) "The Software Factory", QED.

[7] Shepperd, M and Ince, D (1993) "Derivation and Validation of Software Metrics", Clarendon Press.

[8] Fenton, N E and Pfleeger, S L (1997) "Software Metrics - A Rigorous and Practical Approach" (2nd edition), International Thomson Publishing.
[9] Shepperd, M (1995) "Foundations of Software Measurement", Prentice Hall.
[10] Basili, V R and Rombach, H D (1988) "The TAME project: Towards improvement-oriented software environments", IEEE Transactions on Software Engineering, 14:6 pp758-773.
[11] Humphrey, W S (1989) "Managing the Software Process", Addison-Wesley.
[12] Ibanez, M and Reo, D (1998) "Measuring the Impact of Software Process Improvement on Business Objectives - ESI Balanced IT Scorecard", European Software Institute.
[13] Kaplan, R S and Norton, D P (1996) "The Balanced Scorecard", Harvard Business School.
[14] ISO (1998) "ISO/IEC TR 15504-1:1998 Information technology- Software process assessment. Part 1: Concepts and introductory guide", ISO.
[15] Becker, S A and Bostelman, M L (1999) "Aligning Strategic and Project Measurement Systems", IEEE Software, 16:3 pp46-51.
[16] Boehm, B W and Basili, V R (2000) "Gaining Intellectual Control of Software Development", IEEE Computer, 33:5 pp27-33.
[17] Zultner, R (1992) "Quality Function Deployment for Software Satisfying Customers", American Programmer, 5 pp28-41.
[18] Gilb, T (1996) "Level 6:Why we can't get there from here", IEEE Software. 13:1 pp97-98.
[19] Zuse, H (1997) "A Framework of Software Measurement", Walter de Gruyter.
[20] IEEE (1993) "Standard for a Software Quality Metrics Methodology P-1061", IEEE.
[21] Albrecht, K (1992) "The Only Thing That Matters". Harper Business.
[22] Woodings, T L (1995) "A Taxonomy of Software Metrics", Software Process Improvement Network (SPIN), available from Comast Consulting, Perth.
[23] Woodings, T L and Spencer, G J (1992) "Advances in Quality in the Information Technology Industry", Proceedings of Qualcon92, Australian Organisation for Quality, Perth.
[24] Vliet, J C van (1993) "Software Engineering - Principles and Practice", John Wiley.
[25] Moller, K H and Paulish, D J (1993) "Software Metrics - A Practitioner's Guide to Improved Product Development", Chapman & Hall.
[26] Youll, D P (1990) "Making Software Development Visible", John Wiley.
[27] Kitchenham, B A (1996) "Software Metrics - Measurement for Software Process Improvement", NCC/Blackwell.
[28] Kitchenham, B A (1987) "Towards a Constructive Quality Model - Part I: Software Quality Modelling, Measurement and Prediction", Software Engineering Journal, 2:4 pp105-113.
[29] Ebrahimi, N B (1997) "On the Statistical Analysis of the Number of Errors Remaining in a Software Design Document after Inspection", IEEE Transactions on Software Engineering, 23:8 pp529-532.
[30] Woodings, T L and Everett J E (1999) "A Methodology with Quality Tools to support Post Implementation Reviews", Proceedings of the 10th Australasian Conf on Inf Systems, Wellington.
[31] Boehm, B W (1981) "Software Engineering Economics", Prentice Hall.
[32] Tajima, D and Matsubara, T (1981) "The Computer Software Industry in Japan", Computer 14:5 pp89-96.