Software Metrics and Measurement Principles

John M. Roche BSc, Msc
School of Computing and Information Technology
University of Wolverhampton
Wulfruna Street
Wolverhampton
WV1 1LY
England
email: cm1975@uk.ac.wlv.ccub

Abstract

Software measurement is widely advocated as a fundamental constituent of an engineering approach to planning and controlling software development. Unfortunately, there is a dichotomy between the quantity of developed metrics and those used. This paper provides a tutorial review of software engineering measurement indicating the depth and breadth of the field. Individual metrics are not described due to the interest of this paper being on the measurement process and not the products of that process. Generic problems have been identified within existing measurement processes, these provide learning points for the expression of measurement principles. These principles are classified and described according to their position within the formulation, analysis and application stages of measurement. Conclusions are elaborated that suggest that existing measurement frameworks for applying measurement - often called measurement methods - do not provide sufficient support for the principles and their continued use will only serve to replicate the problems. In order to improve the products i.e. metrics, the measurement process requires improvement through inclusion of these principles in a new method.

Key words: software metrics, measurement process, measurement methods, process improvement

1. INTRODUCTION

The concerns of this paper are the overabundance of proposed but unused software metrics; the lack of use of software measurement within software organisations, and the deficiencies of extant measurement methods to develop measures for the evaluation of products and processes that can be applied by organisations. The limited use of software measurement is disturbing when it has provided the discipline of analysis and synthesis for other branches of engineering for the purposes of understanding, control and improvement.

This paper comprises section 2, an overview of the field of software measurement, and section 3, a classification and description of measurment principles. The measurement overview indicates the profusion of types and instances of software measures, and their poor uptake within software development. An agenda of metric principles are considered from the view that if they are applied they will encourage the application of a more correct and improved measurement process i.e. formulation, validation and use, leading to less problems and increased application of measurement. Brief conclusions are made on the need for measurement methods to incorporate the measurement principles and a research agenda is established.

2. AN OVERABUNDANCE OF SOFTWARE MEASURES

Late, over budget and poor quality software still exist despite application of engineering techniques, tools and discipline. Software measurement, the assignment of numerals to attributes of software for a specific purpose, is a technique that has been advocated for over 20 years but has been markedly unsuccessful in gaining acceptance. An introductory paper, by Ince, on software measurement (Ince 1990), for those readers encountering the concept for the first time, indicates the significance of measurement for the purpose of management and control together with an annotated bibliography of important work.

The aims of objective software measurement i.e. measurement that is independant of the collector, are both technical and managerial in nature and include characterisation and evaluation; control and improvement of software quality and productivity, and comparison, prediction and estimation. The scope of software measurement covers such topics as:

- Performance evaluation and models
- Algorithmic/computational complexity
- Reliability models
- Cost and effort estimation
- Productivity measures and models
- Quality models and measures
- Structural and complexity measures
- Design measures

This paper concentrates on those measurements that are applicable to the management and control of software development processes i.e. the last five itemised above. A generally accepted classification of software measures is the two-fold division into *product* and *process measures*, the former being centred on the assessment and prediction of product characteristics, the latter on characteristics of the software construction process i.e. predominantly cost and duration though the quality of the construction process is gaining recognition as a major contributor to software quality. The product class of measures can be specialised into sub-classes characterised by the stage of the life-cycle in which they are collected i.e. code, design or specification.

2. 1 Measures from Code

The historical development of quality measures initially concentrated upon code through the assessment of single attributes of code e.g. code length - lines of code,

complexity by the number of operands and operators (Halstead 1973,1975,1977), control flow complexity (McCabe 1976); and more lately hybrid or composite measures are developed i.e. those capturing more than one attribute (Hansen 1978, Oviedo 1980, Harrison et al 1981). The late availability of these measures and their poor theoretical underpinning has led to both criticism (Halstead's software science - Johnston 1981, Lassez 1981, Hamer et al 1982 Coulter 1983, Evangelist 1983, Shen 1983, Ince 1987, Weyuker 1988 and Shepperd 1988. McCabe's Cyclomatic number - Ince 1988, Weyuker 1988, and Shepperd 1988) and some supportive arguments though the generally held view is that software science measures and cyclomatic number do not provide a firm foundation for use in measurement programmes. Although these measures are generally discredited they do have limited application e.g. Cyclomatic number does provide an indication of the relative difficulty of testing programmes. However, it is somewhat disturbing to record that this adverse criticism has not prevented the measures being regarded as 'classic measures' or being used in more diverse roles e.g. the psychological complexity of maintenance tasks (Curtis et al 1979b, Henry 1990), programmer performance (Curtis et al 1979a), inclusion in a measurement methodology (Lewis 1990), developing software processes (Selby et al 1991), and even the detection of plagiarism (Whale 1990).

Even the more theoretically sound modelling of control flow proposed by Fenton (Fenton 1986, Fenton 1992) based upon defining flow graphs as unique irreducible graphs has received adjustment for application to real-time systems (Fuchs 1991).

2. 2 Design metrics

The general acceptance that earlier life-cycle stage measurement offers greater opportunities for exerting leverage on development has led to increased production of design measures. Predominantly derived from structural design charts to capture the internal characteristics of modules, the degree of connection and information flow between modules, or a hybrid of the two. Shepperd (Shepperd 1988) describes these three classes as intra-module, inter-module, and hybrid design measures.

The intra and hybrid classes (Intra: Szulewski 1981, Emerson 1984, Reynolds 1984; Hybrid: Woodfield 1980, Yau 1980, Henry 1981, Harrison 1982, Card 1988, and Bourque 1991.) are flawed through requiring either implementation details not available at design time, or they use flawed derivatives of software science measures. Inter-module measures founded upon the design concept of coupling record design coupling between modules and between structural levels within a system (McClure 1978, Benyon- Tinker 1979, Yin 1978, Chapin 1979, Belady 1981 and Shepperd 1988). The more notable of which is Ince and Shepperd's IF4 (Ince 1989), an improved derivative of Henry's Information Flow - without the length factor, that has been shown to have some validity for assessing development effort and ease of maintenance of comparable design representations. The achievement of Ince and Shepperd's work with respect to the measurement process is that by eradicating an

existing metric's deficiencies and introducing designed validations, an improved metric has been derived. In adopting this approach the measurement process and the product has been improved! Unfortunately, the approach has to have a potentially useful measure upon which to start improvement. It appears more logical to identify a more appropriate method of metric development that adopts the stance of getting a metric 'right first time'.

2. 3 Specification Measures

Only a small number of specification measures are cited in the literature, Albrecht's function point metric (Albrecht 1983), for assessing a systems functionality, is the most 'popular' having received support within data processing environments. Despite this support it has been subject to amendment to form the Mark II Function Point (Symons 1988) and criticism for not being development independent (Ratcliffe 1990), easy to calculate (Low and Jeffery 1990). Kemerer's recent work confirms that the inter-rater and inter-method reliability should not prohibit Function Point's use though Kitchenham's findings indicate that such a complex formulation is not always necessary (Kitchenham 1993). This illustrates that even widely accepted measures have room for improvement. Other specification measures are deMarco's Bang metric (deMarco 1982), Samson's application of McCabe, Halstead and LOC to specifications written in the OBJ language (Samson et al 1987), Whitty's structural assessment of Z schemas (Whitty 1989), and Bainbridge's analysis of flowgraphs produced from Z schemas (Bainbridge 1990). These measures have received no further confirmatory support other than from their original proponents.

2. 4 Process Measures

Cost, duration and quality are the dominant process measures required for understanding, monitoring and controlling purposes at the planning stage of project management; to date cost and duration have received most attention. Cost estimation is assisted by using estimation models, an introductory overview, recent developments and industrial experience in this field can be found in (Cowderoy 1987, Londeix 1988, Kitchenham 1991 and Hihn 1991). The models may involve making estimates of size adjusted by a number of factors related to product, computer, personnel and project, that may affect productivity e.g. COCOMO (Boehm 1981); or the evaluation of relationships between constraints on a project e.g. SLIM (Putnam 1978). Boehm's (Boehm 1984) evaluation of 8 cost estimating models and Bredero's (Bredero 1989) summary of evaluative studies of cost models provide a concise review of model problems. An objective asessment of estimation models by Kemerer (Kemerer 1987), indicates that calibration to individual organisations is a major requirement for successful model use. Kitchenham (Kitchenham 1991, Kok et al 1990) criticises the COCOMO model on the basis that the model's assumptions are untenable. In particular the raising of the term S - source lines of code, to a power greater than 1 is not supported i.e. the belief does not hold that more effort is required to produce larger projects. Kitchenham's work concludes that COCOMO is

not supported by empirical work even when using the original COCOMO data, and there is an obligation for disciplined application of statistics in validating model assumptions. The assessment of the quality of software processes has yet to be addressed by the application of objective measurement, this paper makes at least one small step in that direction.

3. MEASUREMENT PRINCIPLES

The above review has indicated the depth and breadth of software metrics; the preoccupation with measuring software products, process cost and duration; and the existing metric processes result in most of the metrics being erroneously formulated and inapplicable for widespread use. The measurement process can be characterised as consisting of five activities:

- formulation
- collection
- analysis
- interpretation
- feedback

Associated with these steps is on-going validation. The metrics and their development process exhibit problems that can be classified into three types formulation, analytical and applicational i.e. issues related to applying measurement within organisations. In order that such problems do not continue to manifest themselves it is necessary to incorporate a set of measurement principles into future measurement processes. This section details these principles and indicates their value to the measurement process.

3. 1 Formulation Principles

The major principles identified within the formulation of metrics are:

- measurement objectives should be established prior to data collection
- metric definitions should be clear and unambiguous
- metrics should be constructed upon an underlying theory that has validity within the domain of application.
- metrics to be useful ought to be tailored to specific products or processes

An objective approach targets data collection to that necessary to satisfy the objectives and interpretation would be in context of these goals. Organisations desperately

require guidance on how to collect measures, objective based data collection encourages measurement while preventing the approach of collecting data on everything. This should remove management's fear that data collection will be prohibitively expensive. Deciding what measures have to be collected has to be followed by explicit metric definitions to ensure that collectors know what and how to collect. Application of this principle ensures that the collection step of the measurement process is consistent and repeatable, a feature not present in the formulation of Henry's Information Flow measure (Kitchenham 1990, Ince 1988). Inadequate metric definitions illustrate the potential for counting and replication problems and the unlikely success of transferring a metric developed in one domain to a different environment. A more acceptable formulation process that ensures correct definition of metrics would be to develop metrics that are project or domain specific (Evans 1987). In doing so measurement would become an integral part of software development and project teams would have a sound basis for collection,analysis and comparison of project results.

For metrics to be effective they must be constructed upon an underlying theory that has validity within the project and software engineering domains. A theory provides the foundations for constructing a simplified model of the real world, and yields a context for interpreting the measures. In the rush to produce metrics only cursory consideration has been given to developing an underlying theory. The near classic Halstead, McCabe, and Henry and Kafura measures were all developed upon a theory subsequently found to be flawed. The predominant failing being that theories are imported, often from other domains, without modification or concern for their compatibility with generally accepted software engineering principles (Shepperd 1991). Theories and validations that underpin metrics should be developed local to and shown to be valid for, each specific collection programme. This is difficult to achieve currently where the approach is to develop measures in one specific application area and hope that they can be widely used.

A more suitable approach for metrology to follow is that advocated by Basili (Basili 1984,1988) and Kitchenham (Kitchenham 1990), where measures are tailored to specific products or individual projects. This approach enables more accurate data definition, collection and interpretation, an important feature for those organisations contemplating a measurement programme. The correct application of theories and application of other formulation principles are pre-requisites for effective analysis, validation and interpretation of data. Sound measurement can only be based upon the application of appropriate principles at each stage of the process.

3.2 Analytical Principles

A measure of a metrics success is its frequency of use within software development, correct formulation can not ensure use. Measurement must adhere to the following analytical principles:

- data collection and analysis should be automated
- appropriate statistical techniques are necessary to establish relationships between intermediate and final product characteristics
- ensure the independence of factors used in a metric formulation
- interpretative guidelines and recommendations for action are necessary
- metric validations require suitable construction and should follow a scientific method

Reducing the amount of data collected through objective measurement does not preclude the collection of a substantial amount of data (Evans 1987). The advantages of automated data collection are in easing the collection load and reducing the possibility of data collection errors (Ince 1988, Shepperd 1988, Kitchenham 1986, 1990). Unfortunately, metric collection tools require precise definitions, without them automation will only entomb imprecise data collection into the measurement process. The prerequisite of adequately defined metrics will encourage automatic collection, and so increase the application and use of measurement within organisations.

An effective data collection process requires appropriate statistical techniques for data analysis, a feature highlighted as a major problem in software metrology (Hamer 1982, Coulter 1983, and Ince 1988). A common failing is to assume that the collected data meets the requirements of parametric statistics i.e. the data is obtained from a normal distribution, an assumption not valid with the majority of software engineering data. Several tutorial reviews offer statistical and presentation techniques, such as scatter plots and box-plots, for application to software data have been produced (Kitchenham 1987b, Pickard 1989, Ross 1989 and Myrvold 1990) they indicate how very 'simple' techniques can be beneficial in identifying trends and software 'hot-spots'. One of the more important points advocated is the use of a set of measures to provide more effective characterisation of software attributes (Basili 1983, Kafura 1985, Kitchenham 1987a). The importance of using correct techniques is in ensuring that predictive relationships are real as opposed to illusory between intermediate and final product qualities with the required level of certainty leading to improved sofware management.

The formulation of metrics, collection of data and analysis of the results requires knowledge of the development environment and must be backed up by interpretation and recommendations for action (Kitchenham 1989a, 1990). Kitchenham (Kitchenham 1989a) proposes a four stage interpretation process consisting of:

- identify abnormal values
- find cause(s) of values
- distinguish between causes

- provide corrective action.

The model fails to stress the importance of interpreting values within context of the software development process. An example illustrates, two modules A and B of equal length are tested and the nuber of faults detected are 20 and 80 respectively. How do we interpret the data? Is B or A the problem module? Is B of poor quality or was the software engineer less experienced? It is rather difficult to allocate a specific cause and interpretation to these values if details are unknown about the module's creation and testing processes. This information is required to help answer questions such as who produced and tested the module, what is the developer's and tester's expertise level and experience, how long did it take to build, was more testing time allocated to one module than to others etc.

The common method of developing metrics in one environment and use in others makes the interpretation and recommendations obligations difficult to enforce. Developing metrics that are customised to an application/environment offers the advantage of more accurate and correct analysis, interpretation, and recommendations for action. A simple example illustrates, interpreting a high Henry and Kafura metric value from a system design as indicating areas of re-design could be erroneous if applied to a design that has had its modular structure collapsed for performance purposes. The metric's value in the second case would be interpreted in context of a process of 'design performance tuning' as pinpointing an area altered for peformance and not requiring re-design. This strongly suggests that a metric's value has to be interpreted within context of the specific software process in operation at the time of collection.

A further restriction on the acceptance of measures is the lack of validation evidence to confirm that the measures capture the relationship between the measured attribute and characteristics of interest. Fenton (Fenton 1991) distinguishes between internal validation necessary for the narrow view of measuring for assessment i.e. ensuring that the measure correctly describes the measured attribute - the Representation Condition of measurement theory, and broader external view of predicting a future attribute. Internal validation enables the scale type and appropriate statistical techniques to be determined e.g. for ordinal scale types non-parametric statistics should be applied.

Although internal validation is important, the majority of metric empirical validation studies are concerned with the broader predictive use of measures. The allure of being able to control and predict intermediate and end product attributes of interest has captivated metrologists but very little concrete evidence confirming such predictive relationships exist. Validating metrics requires a scientific method - hypotheses are established prior to data collection and experiments carried out to establish the validity or otherwise of the hypotheses, an approach supported by Fenton (Fenton 1991). The more usual method involves developing a metric and then searching for some data for a validation study that often involves correlations between the metric

values and some attribute that can be found to be correlated with the data!. Another approach to validation is showing that the new measure is as good as other established measures (Ramamurthy 1988, Li 1987, de Paoli 1990, Whale 1990). Unfortunately, the supposition is made that well known measures are valid predictors, more often than not this is incorrect - new metrics are often validated against lines of code, Halstead's, and McCabe's metrics. Some success in improving the validation process has occurred through analysis of a metric's defects - theoretical validation, and designing scientific empirical validation experiments based upon hypotheses testing (Ince 1989, Shepperd 1991) to confirm the predictive relationships.

Recommended guidelines for validations include the use of professional software developers, large scale software, and a large number of data points (Shepperd 1991). An important domain of metric formulation i.e. academia, presents severe difficulties in ensuring the above criteria are met though they would be easily achieved if metric development was integrated within organisation's software development. Conte (Conte 1986), Fenton (Fenton 1991), Kitchenham (1987b) and Basili (Basili 1985) provide detailed rationale and techniques to assist correct formulation of validations that meet the demands of measurement, experimental and statistical theories. A further requirement is that relationships should be validated within specific environments and not imported into other dissimilar environments. Despite all this sound advice software measurement still follows the 'search for or design a metric and hunt for data' syndrome. The inclusion of measurement within software development processes offers the potential for optimising metric validation guidelines and ensuring more rigorous validations.

Theoretical and empirical validations have been supplemented by other validation techniques such as Weyuker's Axiomatic approach for complexity measures (Weyuker 1988), Shepperd's Algebraic approach (Shepperd 1991) and Basili's method, evaluation and feedback approach (Basili 1991). Weyukers axioms have been subject to evaluation and criticism (Cherniavsky 1991), the axioms are considered as sufficient but not complete for validation studies; more localised development of metrics should lead to appropriate axioms for particular domains. There is very little evidence on the use of these validation techniques but they have potential as a supplement to theoretical and empirical validations.

3. 3 Application Principles

The deficit of applied measurement experience has led to recognition of a sparse number of principles, if more measurement was implemented within organisations and their experiences were published further principles could be derived. Those principles derived from the existing literature are:

- reliable metrics should be applied by organisations
- the cost implications of introducing measurement into organisations needs establishing prior to deployment
- organisations should plan the introduction and continued use of measurement
- measurement programmes should be 'owned' to reduce potential resentment and disruption
- measurement requires promotion through early results
- applying metrics places a training and educational resources requirement upon organisations

The laudable aims behind applying software metrics, often expressed within the software measurement fraternity and by managers (Basili 1991a, Frewin 1986), are tainted by expressions from software management regarding the reliability of metrics (Born 1986, Wingrove 1986, Lehman 1991). This polarisation of metrologists and managers' views is a problem that has yet to be addressed effectively in software metrics methods. There is even a reticence towards accepting measurement as a worthy candidate for improving software development that has not been shown to other methods of improvement e.g. use of CASE, though with all the measurement challenges still to be resolved it is hardly surprising. The rationale behind this attitude may be a fear of the unknown costs resulting from experimenting with virtually untried and untested metrics, and there is little objective evidence on the costs involved in using measurement - - though CASE products are equally untried and unevaluated!. Obviously introducing 'new' technology will involve additional costs at the outset but long term quality and productivity pay-offs should provide adequate cost benefits. Measurement will also disturb the environment in which it is applied - the Hawthorne effect, it can not be tried on a short term stab it needs commitment and resources.

Allied with cost problems is knowing how to introduce measurement into an organisations, introducing change without a plan can be cost ineffective through being unstructured and piecemeal. Planned introduction provides understanding, guidance and the ability to control what happens and to cost measurement activities. Unplanned measurement activities will fail and discourage further attempts. Blanket introductions through company dictat are also doomed to failure through being insensitive and ill tailored to personnel work practices. People will always work around those aspects of a practice they do not like which will lead to uncertainty in interpreting measurement results.

The introduction of change does place a requirement upon management to provide extra resources, motivation, training or guidance in the new skills and techniques, and assignment of responsibilities. Knowledge on these features is in short supply though the work reported by Grady and Caswell at Hewlett Packard (Grady 1987, Grady

1993), and the Metrics Educational toolkit (METKIT) project (Bush and Fenton 1990, Fenton 1991) provide limited evidence. They provide insights into software metric development and problems but they do not provide a method of incorporating measurement into working practices. A successful metric programme will result from individuals 'owning' the measurement process and the measurement objectives, and not feeling threatened by a programme of measurement. Without the 'goodwill' of staff the most detailed measurement programme will fail. The demands placed upon software engineers are sufficiently high to only warrant application of measurement programmes if time, quality and costs are acceptable. Minimising time and cost, and maximising the quality of metric programmes are possible if measurement becomes an integral part of software development processes. The educational and training needs of staff can also be satisfied through plans of their development process containing indications of what should be measured, when and by whom.

This section has indicated the major principles that should underpin successful measurement processes. The problems exhibited in the majority of measurement programmes are the result of applying an implicit ad hoc process of formulating, validating and applying metrics though some success has resulted from the development of metric methods.

CONCLUSIONS

The diversity and extent of software measurement has been outlined to illustrate that it is not just a flight of fancy but an important element of a software engineering discipline. The derivation of measurement principles from analysis of of the mistakes made by metriologists provides the basis upon which measurement practice can be improved. The following learning points are relevant to establishing a research agenda for future improvement of software measurement:

- software measurement is predominantly an error prone and unsucessful activity
- existing measurement processes appear to be inadequate for an engineering discipline through producing metrics that have deficiencies in their formulation, analysis or application activities.
- the identified measurement principles should be supported within any measurement programme
- measurement will be improved by the adoption of a structured framwork - method that incorporates these principles

These learning points suggest that measurement can be improved by application of a measurement framework containing the principles. Future work in this area will evaluate the support for the principles offered by existing measurement methods as a basis upon which to construct a measurement method that will be acceptable to organisations and encourage increased use of measurement.

REFERENCES

Albrecht,A.J. and Gaffney,J.E. (1983) Software Function, Source Lines of Code, and Development Prediction: A Software Science Validation. In: *IEEE Trans. on Software Eng.* **SE 9** (6) p639-647 1983.

Bainbridge,J. and Whitty,R. (1990) Obtaining structural metrics of Z specifications for system development. In: *Proc. 5th Ann. Z User Meeting* Oxford 17-18 Dec. 1990.

Basili,V.R. and Selby,R.W. (1985) Calculation and use of an environment's characteristic software metric set. In: *Proc. 8th Intl. Conf. on Softw. Eng.* p386-391 1985.

Basili,V.R. and Hutchens,D.H. (1983) An Empirical Study of a Syntactic Complexity Family. In: *IEEE Trans. Softw. Eng.* **SE 9** (6) p664-672 1983.

Basili,V.R. and Rombach,H.D. (1988) The TAME project: Towards Improvement-oriented software environments. In: *IEEE Trans on Softw. Eng.* **SE 14** (6) June p758-771 1988.

Basili,V.R. and Selby,R.W. (1987) Comparing the effectiveness of software testing strategies. In: *IEEE Trans. on Softw. Eng.* **SE 13** (12) p1278-1296. 1987.

Basili,V.R. and Selby,R.W. (1991) Paradigms for experimentation and empirical studies in software engineering. In: *Reliability Engineering and System Safety* **32** p171-191 1991.

Basili,V.R. and Weiss,D.M. (1984) A methodology for collecting valid software engineering data. In: *IEEE Trans. on Softw. Eng.* **SE 10** p728-738 1984.

Belady,L.A. and Evangelisti,C.J. (1981) System partitioning and its measure. In: *J. of Syst. and Softw.* **SE 2** p23-29 1981.

Benyon-Tinker,G. (1979) Complexity measures in an evolving large system. In: *Proc. ACM Workshop on Quantitative Software Models* p117-127 1979.

Boehm, B.W. (1981) *Software Engineering Economics.* Englewood Cliffs NJ Prentice-Hall 1981.

Boehm, B.W. (1984) Software engineering economics. In: *IEEE Trans on Softw. Eng.* **SE 10** (1) p4-21 1984.

Born,G. Controlling software quality. In: *Softw. Eng. J.* 1 (1) Special issue on Controlling software projects January p24-28 1986.

Bourque,P. Cote,V. (1991) An experiment in software sizing with structured analysis metrics. In: *J. of Syst. and Softw.* 15 (2) p159-72 1991.

Bredero, R. Caracoglia,G. Jaggers,C. Kok,P. Tate,G. Verner,J. (1989) Comparative evaluation of existing cost estimation tools. In: *MERMAID Project Report D7.1Y* November 1989.

Bush,M. and Fenton,N.E. (1990) Software measurement: A conceptual framework. In: *J. Syst. Softw.* 12 p223-231 1990.

Card,D.N. and Agresti,W.W. (1988) Measuring software design complexity. In: *J. of Syst. and Softw.* 1988.

Chapin,N. (1979) A measure of software complexity. In: *Proc. AFIPS* p995-1002 1979.

Cherniavsky,J.C. (1991) On Weyuker's axioms for software complexity measures. In: *IEEE Trans. on Softw. Eng.* SE 17 (6) p636-638 1991.

Conte,S.D. Dunsmore,H.E. and Shen,V.Y. (1986) *Software Engineering Metrics and Models.* Benjamin Cummings Memlo Park Ca USA 1986.

Coulter,N.S. (1983) Software Science and Cognitive Psychology. In: *IEEE Trans. Softw. Eng.* SE 9 (2) p166-171 1983.

Cowderoy,A.J.C. (1989) New trends in Cost-Estimation. In: *Proc. Centre for Software Reliability Conf. on Measurement for Softw. Control and Assurance* (Eds.) Kitchenham,B. and Littlewood,B. Elsevier Bristol U.K. p63-88 1989.

Curtis,B. Sheppard,S. Milliman,P. Borst,M. and Love,T. (1979b) Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. In: *IEEE Trans. on Softw. Eng.* SE 5 (2) p96-104 1979

de Paoli,F. Morasca, S. (1990) Extending software complexity metrics to concurrent programs. In: *Proc. 14th Annual International Computer Software and Applications Conference* p414- 19 1990.

deMarco,T. (1982) *Controlling Software Project Management, Measurement and Estimation.* New York, Yourdon Press, 1982.

Emerson,T.J. (1984) A discriminant metric for module cohesion. In: *Proc. 7th Int. Conf. on Softw. Eng.* p335-343 1984.

Evangelist,W.M. (1983) Software Complexity Metric Sensitivity to Program structuring rules. In: *J. Syst. and Softw.* 3 (3) p231-243 1983.

Evans,M.W. and Marciniak,J.J. (1987) *Software quality assurance and management.* John Wiley and Sons, Chichester UK 1987.

Fenton,N.E. and Whitty,R.W. (1986) Axiomatic approach to software metrication through program decomposition. In: *Computer J.* 29 (4) p329-339 1986.

Fenton N.E. (1991) *Software Metrics: A Rigorous Approach.* Chapman Hall London 1991.

Frewin,G.D. and Hatton,B.J. (1986) Quality management - procedures and practices. In: *Softw. Eng. J.* 1 (1) Special issue on Controlling software projects January p29-38 1986.

Fuchs,N.and S. Stainer (1991) Language independent definition of axiomatic metrics In: *Proceedings Ist BACS Seminar on Formal Aspects of Measurement* University of the South Bank London 1991

Grady, R.B. and Caswell, D.L. (1987) *Software Metrics: Establishing a company wide program.* Prentice Hall NJ 1987.

Grady,R.B. (1992) *Practical Software Metrics for Project management and Process Improvement.* Prentice Hall Englewood cliffs NJ. 1992

Halstead,M.H. and Zislis,P.M (1973) Experimental Verification of Two Theorems of Software Physics. In: *Computer Science Department Technical Report 97* Purdue University 1973.

Halstead,M.H. (1975) Software Physics: Basic Principles. In: *IBM Research Report RJ1582* 1975.

Halstead,M.H. (1977) *Elements of Software Science.* New York Elsevier North Holland 1977.

Hamer,P.G.and Frewin,G.D. (1982) M.H. Halstead's Software Science - A Critical Examination. In: *Proc. 6th Intl. Conf. Softw. Eng.* p197-206 1982.

Hansen,W.J. (1978) Measurement of program complexity by the pair (Cyclomatic number, operator count) In: *ACM SIGPLAN Notices* 13 (3) p29-33 1978.

Harrison,W. and Magel,K. (1981) A complexity measure based upon nesting level. In: *ACM SIGPLAN Notices* 16 (3) p63-74 1981.

Harrison,W. Magel,K. Kluczny,R. and DeKock,A. (1982) Applying software complexity metrics to program maintenance. In: *Computer* **15** (9) Sept. p65-78 1982.

Henry,S. and Kafura,D. (1981) Software structure metrics based on information flow. In: *IEEE Trans. on Softw. Eng.* **SE 7** (5) Sept. p510-517 1981.

Hihn, J. and Habib-Agahi, H. (1991) Cost estimation of software intensive projects: A survey of current practice. In: *Proc. 13th Intl. Conf. on Softw. Eng.* 13-16 May Austin Texas USA IEEE Computer Society Press p276-287 1991.

Ince,D. (1990) An annotated bibliography of software metrics In: *ACM SIGPLAN Notices* **25** (8) p.15-23 Aug. 1990.

Ince,D.C. and Shepperd,M.J. (1988) System design metrics: A review and perspective. In: *Proc. IEE/BCS Conf. on Softw. Eng.* p23-27 1988.

Ince,D.C. Shepperd,M.J. (1989) An empirical and theoretical analysis of an information flow-based system design metric. In: *Proc. ESEC '89. 2nd European Softw. Eng. Conf.* p86- 99 (Eds.) Ghezzi,C. McDermid,J.A. Springer-Verlag Berlin West Germany 1989.

Johnston,D.B. and Lister,A.M. (1981) A Note on the Software Science Length Equation. In: *Softw. Practice and Experience* **11** (8) p875-879 1981.

Kafura,D. and Canning,J. (1985) A validation of software metrics using many metrics and two resources. In: *Proc. 8th Intl. Conf. on Softw. Eng.* p378-385 1985.

Kemerer,C.F. (1987) An empirical validation of software cost estimation models. In: *CACM* **30** (5) May p416-429 1987.

Kemerer,C.F. (1993) Reliability of Function Points Measurement. In: CACM **36** (2) February p85-97 1993.

Kitchenham,B. and Pickard,L (1987a) Towards a constructive quality model Part I : Software quality modelling, measurement and prediction. In: *Softw. Eng. J.* p 105-113 July 1987.

Kitchenham, B. and Pickard,L (1987b) Towards a constructive quality model Part II : statistical techniques for modelling software quality in the ESPRIT REQUEST project. In: *Softw. Eng. J.* p 114 - 126 July 1987.

Kitchenham, B.A. and Walker,J.G. (1989a) A quantitative approach to monitoring software development. In: *Softw. Eng. J.* January p2-13 1989.

Kitchenham,B.A. and Linkman,S.J. (1990) Design metrics in practice. In: *Info. and Softw. Tech.* **32** (4) May p304-309 1990.

Kitchenham,B.A. and McDermid,J.A. (1986) Software metrics and integrated project support environments. In: *Softw. Eng. J.* **1**(1) p58-64 1986.

Kitchenham,B.A. Pickard,L.M. and Linkman,S.J. (1990a) An evaluation of some design metrics. In: *Softw. Eng. J.* **5** (1) Special issue on software reliability and metrics January p50-58 1990.

Kitchenham,B.A. (1991) Empirical studies of the assumptions underlying software cost estimation models. In: *Info. and Softw. Tech.* **34** (4) p211-218 1991.

Kitchenham,B.A. (1993) Inter-item correlations among function points. In: *Proc. 1st Intl. Software Metrics Symposium* Baltimore USA May 21-22 p11-15 1993.

Kok,P.A.M. Kitchenham,B.A. and Kirakowski,J. (1990) The Mermaid approach to software cost estimation. In: *Proc. ESPRIT '90 Conf.* Kluwer Academic Press 1990

Lassez,D. Van de Knijff,and Shepherd, (1981) A Critical Examination of Software Science. In: *J. Syst. and Softw.* **2** p105-112 1981.

Lehman, M.M. (1991) Software engineering - theory and practice. In: *Softw. Eng. J.* **6** (5) Special issue on Software process and its support Sept. p243-258 1991.

Lewis,J.A. and Henry,S.M. (1990) On the benefits and difficulties of a maintainability via metrics methodology. In: *Softw. Maintenance: Research and Practice* **2** p113-131 1990.

Li,H.F. Cheung,W.K. (1987) An empirical study of software metrics. In: *IEEE Trans. on Softw. Eng.* **SE 13** (6) 1987.

Londeix,B. (1988) Aspects of estimation practice in software development. In: *Proc. IEE/BCS Conf. on Softw. Eng. '88* p75-79 1988.

Low,G.C. and Jeffery,D.R. (1990) Function Points in the estimation and evaluation of the software process. In: *IEEE Trans. on Softw. Eng.* **SE 16** (1) p64-71 1990.

McCabe,T.J. (1976) A complexity measure. In: *IEEE Trans. on Softw. Eng.* **SE 2** (4) Dec. p308-320 1976.

McClure,C. (1978) A model for program complexity analysis. In: *Proc. 3rd Intl. Conf. on Softw, Eng.* p149-157 1978.

Myrvold,A. (1990) Data analysis for software metrics. In: *J. Syst. and Softw.* **12** p271-275 1990.

Oviedo,E. (1980) Control flow, data flow and program complexity. In: *Proc. IEEE Computer Software and Applications Conference* Chicago USA p146-152 1980.

Pickard,L.M. (1989) Analysis of software metrics. In: *Proc. Centre for Software Reliability Conf. on Measurement for Softw. Control and Assurance* (Eds) Kitchenham,B. and Littlewood, B. Bristol U.K. Elsevier p155-180. 1989.

Putnam,L.H. (1978) A general empirical solution to the macro software sizing and estimating problem. In: *IEEE Trans. on Softw. Eng.* **SE 4** p345-361 1978.

Ramamurthy,B. Melton,A. (1988) A synthesis of software science measures and the cyclomatic number. In: *IEEE Trans. on Softw. Eng.* **SE 14** (8) August p1116-1121 1988.

Ratcliffe,B. and Rollo,A. (1990) Adapting Function Point analysis to Jackson system development. In: *Softw. Eng. J.* **5** (1) Special issue on software reliability and metrics p79-84 1990.

Reynolds,R.G. (1984) Metrics to measure the complexity of partial programs. In: *J. of Syst. and Softw.* **4** (1) p75-92 1984.

Ross,N. (1989) The collection and use of data for monitoring software projects. In: *Proc. Centre for Software Reliability Conf. on Measurement for Softw. Control and Assurance* (Eds Kitchenham,B. and Littlewood, B. Bristol U.K. Elsevier p125-153 1989.

Samson,W.B. Nevill,D.G. and Dugard,P.I. (1987) Predictive software metrics based upon a formal specification. In: *Info. and Softw. Tech.* **29** (5) p242-248 1987.

Shen,V.Y. Conte,S.D. Dunsmore,H.E. (1983) Software science revisited: A critical analysis of the theory and its empirical support. In: *IEEE Trans. on Softw. Eng.* **SE 9** (2) March p155-165 1983.

Shepperd,M.J. (1988) An Evaluation of Software Product Metrics. In: *Info. and Softw. Tech.* **30** (3) p177-188 1988.

Shepperd,M.J. (1991) System architecture metrics: An evaluation. In: *Unpublished PhD thesis* March 1991.

Symons,C.R. (1988) Function Point Analysis: Difficulties and Improvements. In: *IEEE Trans. on Softw. Eng.* **SE 14** (1) p 2-11 1988.

Szulewski,P.A. Whitworth,M.H. Buchan,P. Dewolf,J.B. (1981) The measurement of software science parameters in software designs. In: *ACM SIGMETRICS P.E.R.* p89-94 1981.

Weyuker, E.J. (1988) Evaluating software complexity measures. In: *IEEE Trans. on Softw. Eng.* **SE 14** (9) Sept. p 1357-1365 1988.

Whale,G. (1990) Software metrics and plagiarism detection. In: *J. Syst. and Softw.* **13** (2) 131- 138 1990.

Whitty,R. (1989) Structural metrics for Z specifications. In: *Proc. 4th Ann. Z Technical and Users Meeting* Oxford 14-15 December 1989.

Wingrove,A. (1986) The problems of managing software projects. In: *Softw. Eng. J.* **1** (1) Special issue on Controlling software projects January p3-6 1986.

Woodfield,S. (1980) Enhanced effort estimation by extending basic programming models to include modularity factors. In: *Ph.D. Dissertation Computer Science Dept. Purdue University.* 1980.

Yau,S.S. Collofello,J.S. and MacGregor,T.M. (1978) Ripple effect analysis of software maintenance.In: *Proc. COMPSAC '78* p60-65 1978.

Yin and Winchester (1978) The establishment and use of measures to evaluate the quality of software designs. In: *Proc. of the Softw. Quality Assurance Workshop* San Diego Calif November 15-17 p45-52 1978.