University of Otago
Te Whare Wananga o Otago
Dunedin, New Zealand

Software process engineering for measurement-driven software quality programs — realism and idealism

Stephen G. MacDonell
Andrew R. Gray

The Information Science Discussion Paper Series

Number 96/19
September 1996
ISSN 1172-6024

University of Otago

Department of Information Science

The Department of Information Science is one of six departments that make up the Division of Commerce at the University of Otago. The department offers courses of study leading to a major in Information Science within the BCom, BA and BSc degrees. In addition to undergraduate teaching, the department is also strongly involved in postgraduate research programmes leading to MCom, MA, MSc and PhD degrees. Research projects in software engineering and software development, information engineering and database, software metrics, knowledge-based systems, natural language processing, spatial information systems, and information systems security are particularly well supported.

Discussion Paper Series Editors

Every paper appearing in this Series has undergone editorial review within the Department of Information Science. Current members of the Editorial Board are:

Assoc. Professor George Benwell	Assoc. Professor Nikola Kasabov
Dr Geoffrey Kennedy	Dr Stephen MacDonell
Dr Martin Purvis	Professor Philip Sallis
Dr Henry Wolfe	

The views expressed in this paper are not necessarily the same as those held by members of the editorial board. The accuracy of the information presented in this paper is the sole responsibility of the authors.

Copyright

Copyright remains with the authors. Permission to copy for research or teaching purposes is granted on the condition that the authors and the Series are given due acknowledgment. Reproduction in any form for purposes other than research or teaching is forbidden unless prior written permission has been obtained from the authors.

Correspondence

This paper represents work to date and may not necessarily form the basis for the authors' final conclusions relating to this topic. It is likely, however, that the paper will appear in some form in a journal or in conference proceedings in the near future. The authors would be pleased to receive correspondence in connection with any of the issues raised in this paper, or for subsequent publication details. Please write directly to the authors at the address provided below. (Details of final journal/conference publication venues for these papers are also provided on the Department's publications web pages: `http://divcom.otago.ac.nz:800/COM/INFOSCI/Publctns/home.htm`). Any other correspondence concerning the Series should be sent to the DPS Coordinator.

Department of Information Science
University of Otago
P O Box 56
Dunedin
NEW ZEALAND
Fax: +64 3 479 8311
email: dps@infoscience.otago.ac.nz
www: `http://divcom.otago.ac.nz:800/com/infosci/`

Software Process Engineering for Measurement-Driven Software Quality Programs - Realism and Idealism

Stephen G. MacDonell and Andrew R. Gray

Department of Information Science

University of Otago

PO Box 56, Dunedin, New Zealand

email: stevemac@commerce.otago.ac.nz

Abstract

This paper brings together a set of commonsense recommendations relating to the delivery of software quality, with some emphasis on the adoption of realistic perspectives for software process/product stakeholders in the area of process improvement. The use of software measurement is regarded as an essential component for a quality development program, in terms of prediction, control, and adaptation as well as the communication necessary for stakeholders' realistic perspectives. Some recipes for failure are briefly considered so as to enable some degree of contrast between what is currently perceived to be *good* and *bad* practices. This is followed by an evaluation of the quality-at-all-costs model, including a brief pragmatic investigation of quality in other, more mature, disciplines. Several programs that claim to assist in the pursuit of quality are examined, with some suggestions made as to how they may best be used in practice.

Introduction

Examining any software engineering text is almost certain to result in the reader finding a selection of 'war stories' in the first few pages. These stories point out the huge costs of system development and software failure It is granted here that many of these stories are exaggerations or omit certain crucial facts, such as the often cited study by The US General Accounting Office (1979) which found that about three quarters of expenditure on a set of projects never produced or contributed towards *any* working system. Most citing papers have since failed to mention that these projects were selected on the basis of already being in difficulty. The conclusion that most projects heading towards failure do in fact fail is less significant than some of these authors would have their readers believe.

However, given that there is at least some basis of truth in these project management horror stories it may be expected that organizations would be greatly concerned about the quality of their software development process and of the delivered product. While this concern does seem to exist for many companies, few have actually gone as far as taking a proactive stance towards improving their software quality. This is despite the trend towards maturity accreditation which requires quality development processes. An important question, which will be approached later, is why such companies appear to pay lip service to something so crucial to their survival.

For those companies that do wish to measure and improve quality, a wide range of guides, methodologies, and standards confront them, making the creation of a suitable, and customised, quality-improvement program difficult. At the very least setting up such a program is an expensive exercise in examining the many alternatives, rejecting and altering these to suit the organisation, and finally in the actual implementation of the program.

Even once such a program has been put in place, it requires support from a measurement program. This measurement focus can be a subset of the quality framework or standalone as providing services to the remainder of the organisation. Either way, the integration of measurement and quality improvement is a critical, and delicate, task.

This paper brings together a set of commonsense recommendations relating to the delivery of 'software quality', with an emphasis on measurement-driven quality improvement. The need for realism in such a metrics program is also emphasised as necessary to avoid an overly theoretical, potentially counter-productive, and certainly costly exercise. The goal of this paper is to attempt the formulation of a set of fundamental characteristics for successful quality measurement programs. This set of characteristics is based on experience reports relating to successes and failures in quality improvement, as well as on recent research into the various determinants of success in software metrics programs. This could form a baseline, a minimum set of characteristics that must exist if a metrics-driven quality improvement program is to succeed.

Drivers of Quality

There are many aspects of the software development process that affect quality in some way. These include the tools, methods, and development staff used. While there have been found to exist positive relationships between the sophistication of tool and methodology support and the quality of the eventual system, simply 'throwing' these costly resources at the development process is not sufficient. This 'shotgun' approach may result in isolated gains in productivity, or one-off project successes, but without a disciplined, well-measured, software process these gains will most likely be sporadic at best. It is suggested here that a 'quality software process' is one that is defined, leveled, accepted, used, monitored, controlled, reused and improved. Several of these steps require some form of measurement as will be discussed later.

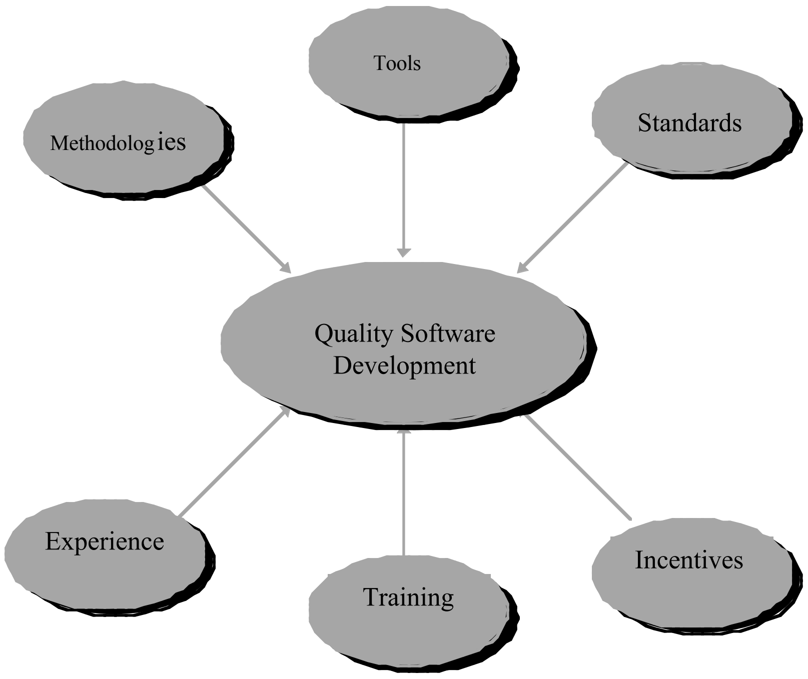

Figure 1 A Sample of Quality-Drivers

Software Processes for Quality

A defined software process contains a series of steps towards goals, and therefore requires measurement to determine the current stage of the process, the performance for each stage, and also to identify when the process is ready to advance to a subsequent stage. When the process is complete measurements are crucial for analysing the strengths and weaknesses of development. This is obviously an important aspect of quality-improvement. The leveling of the process allows for a hierarchy of measurements, so that the process can be assessed at any desired magnification.

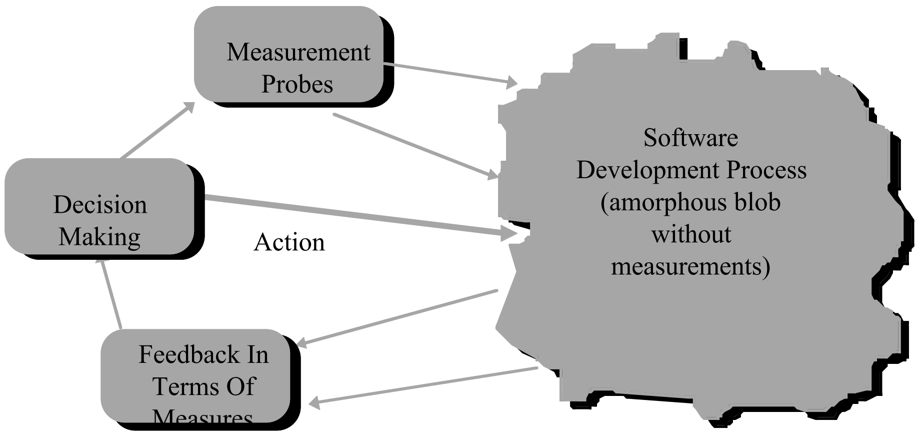
Measurement Probes
Decision Making
Action
Software Development Process (amorphous blob without measurements)
Feedback In Terms Of Measures

Figure 2 The Role of Measurement in Software Development

One of the first stages of designing such a process is to determine what is meant by the term quality. This is difficult since as observed by Juran (1979), quality has many meanings and its ambiguity can lead to many problems and disagreements. The Software Engineering Technical Committee of the IEEE Computer Society (1983) is of little further help when it defines software quality as “The degree to which software possesses a desired combination of attributes.” Other definitions from the same source add at least some tangible meaning, “Totality of features and characteristics of a software product that bear on its ability to satisfy given needs; for example, conform to specifications” and “The degree to which a customer or user perceives that software meets his or her composite expectations.” Still, the ambiguity inherent in these definitions is troublesome. While the idea of a single definition of quality is unrealistic, there needs to be some idea of what goes towards making up quality for a given organisation. It is these components and sub-components of quality that should form the goals of any software process.

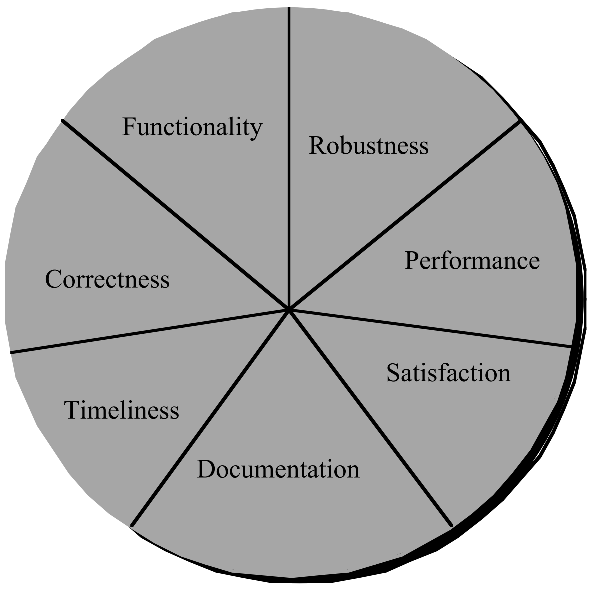

Figure 3 Some Slices of the Quality Pie

Software Metrics as one Aspect of Quality Software Processes

It has been observed by Jones (1995) that “software progress monitoring is so poor that several well-known software disasters were not anticipated until the very day of expected deployment!". Measures are therefore vital to assess the development on a continuous basis throughout its life-cycle. Similarly, process improvement requires longitudinal measurement to determine that some improvement has in fact occurred. In many cases there exists no objective basis on which to judge product quality. Thus it then becomes impossible to measure or predict. Monitoring can be carried out incorporating both objective and subjective measures and a single measure should never be used to answer a question concerning performance (Debou et al 1994). Moreover, measures need to be understood and collected *across* the organisation if a comprehensive understanding of status and progress is to be attained (Krasner 1994).

Miscommunication, or "loose language", makes a significant contribution to development problems. A good measurement program provides unambiguous results that allow for and encourage communication between stakeholders (ami 1995). Blame attributing should not be the result of a measurement program. Both understanding and a willingness to adhere to decisions resulting from the measurement program are required from all process stakeholders. The program must be supported, both monetarily and personally and its importance must be understood by all participants. While many of these statements may appear to be nothing more than ‘common-knowledge’ or truisms the difficulty comes with the actual implementation.

Given a monitoring program for a development process, it is then possible to assert control. Juran (1979) explains that software quality control is the process of measuring actual quality, comparing this to some standard, and then acting on the discrepancy. As DeMarco stated, the ability to measure is a requisite for control (DeMarco 1982). This control can consist of concentrating development effort on weaknesses in the quality of the product and process.

Organizations must collect measures of both process and product quality. Data collection should be automated wherever possible, and reporting should be focused on exceptions (Arthur et al. 1993) rather than continuation of norms. Assessment should be a continuous process (Bootstrap Project Team 1993) throughout development, not at a small number of pre-specified points. The early identification of anomalies is crucial for minimsing the costs of corrective action.

With the focus on cost minimisation, positive aspects of software quality metric processes should be reusable for other projects. This is part of the “maximising lessons learned” principle. The iterative improvement of the measurement program should be a stated goal.

Finally, the measurements extracted during the process can be used to improve future development as part of a lessons learned philosophy. Three major principles in software quality improvement are, firstly, to understand your baseline - an organisation needs to be aware of current position in terms of products, processes and goals. Secondly, not all software is the same - optimal software process for an organisation, or even a project, may not be optimal for another organisation/project. Thirdly, let experience drive change - all changes are experiments and should be treated as such (McGarry 1995).

Recipes for Failure, and Even Some for Success

Failure of Quality Programs

Based on practical experience it is possible to identify a number of common mistakes and misconceptions that can lead to an increased risk of failure for a software quality program. Some of these have already been mentioned in sections above.

Some fairly obvious ways to increase the chance of failure are to fail to communicate requirements to stakeholders. Ignore users’ requests for what they need, don’t let developers know what is expected of them, and assure management that this exact amount of funds are necessary and the project will be delivered on this date.

Johnson (1995) reports the findings of a survey of 365 IT executive managers in the USA concerning the development of more than 8000 applications. When asked why projects failed, the following breakdown of responses was provided:

Incomplete requirements	13%
Lack of user involvement	12%
Lack of resources	11%
Unrealistic expectations	*10%*
Lack of management support	9%
Changing requirements	9%
Lack of planning	*8%*

Those factors emphasised (in italics) are directly influenced by the adequacy and performance of an organisation's measurement program and serve to illustrate the consequences of an absence of effective project management. Furthermore, lack of resources and management support are as significant in determining the success of a measurement and process improvement program as they are for development projects themselves.

Keil (1995) suggests that a failure to carry out early and frequent risk assessment is a common error that leads to project escalation. A further common error in terms of process measurement, even under a goal-oriented framework, is to measure everything from the outset. This can result in an overwhelming volume of information without the infrastructure required to analyse and use it effectively (SPC 1994; Debou et al. 1994).

Quality Program Success

In contrast, successful projects exhibit the following characteristics (according to Johnson's survey (1995)):

User involvement	16%
Management support	14%
Clear requirements	13%
Proper planning	*10%*
Realistic expectations	*8%*
Smaller project milestones	*8%*

Again the emphasised factors are indicative of the influence of effective project management, which is inherently based on measurement, for successful software development. Johnson (1995) goes on to suggest a 'Success Points' grading scheme which enables organisations to pre-determine whether they are likely to successfully develop quality software systems.

The Metricate framework (SPC 1994) provides the following definition of a metrics program: "A metrics program is the formalization of procedures to collect and interpret software metrics within an organization. A successful metrics program will have well-defined goals, and provide feedback on how the software development process can be improved."

Recommendations

In order of frequency of occurrence in the literature (see the list of references at the end of the paper), the following factors are recommended as those that are more likely to lead to the development of quality software through the application of measurement-driven software assessment and improvement programs:

- Executive management support - contemporary opinion clearly rates this factor as the most important in implementing and using a measurement program as part of a quality framework. Moreover, middle management support is also essential if the programs are to succeed at the operational level.
- Adequate resources/funding for assessment and improvement - too often measurement and improvement are expected to happen in addition to the 'real work' of development, but experience clearly shows that this simply does not happen. Separate and adequate funding is essential if these programs are to be of real effect.
- An appropriate corporate culture - the measurement function, and the results of measurement activities, must be seen to have value by all stakeholders. This may involve the breaking down of barriers around information sources, barriers based on political rather than organisational motivation. Actions resulting from measurement outcomes also need to be monitored for impact.
- Realistic expectations of 'reward' - too often organisations expect immediate and significant payback from the measurement function, but these expectations are seldom achieved. More appropriate levels of anticipated benefits need to be set out and communicated to all stakeholders.
- The appointment of a (preferably voluntary) measurement sponsor - an individual in middle management to act as measurement sponsor can be influential regarding the success of the program, particularly in terms of maintaining momentum after an initial concentration of activity at the beginning of program use.
- Clearly defined and communicated measurement goals - *ad hoc* measurement is little better than no measurement at all in terms of long-term process quality improvement. Much measurement literature therefore emphasises the need to specify in clearly defined terms the goals and objectives of the measurement program.
- Verification methods - measurement will only succeed if the measures as defined are collected consistently and effectively according to the specified definitions.

- Adequate analysis methods - data collection is one step in the establishment of a successful measurement program, but the determination and use of *appropriate* analysis techniques is also necessary if progress is to be made.
- Trade-off awareness - there is a clear trade-off between the accuracy and granularity of the data collected in a metrics program and the effort required for that collection. Again, unrealistic expectations of the value of a cheap metrics program can only result in disillusionment with the measurement function.
- Feedback mechanisms - process and product measurement can be viewed as unnecessary overhead by developers and operations managers if they fail to receive adequate feedback on their data collection efforts. Facilities for communication in both directions between the measurement group or sponsor and developers and managers is needed to ensure the continuing worth and relevance of the measurement function.
- Customer orientation - the almost obsessive nature of some quality frameworks in terms of measuring product and/or process quality can mean a lack of attention on the ultimate consumers of software, the users and customers. Measures of customer and user satisfaction are required if organisations are to maintain process improvement in a direction that can help to ensure the longer term viability of the enterprise.
- A dynamic nature - the measurement function needs to be sufficiently flexible to adapt as the needs and goals of the organisation change over time.
- Adequate training (including resources) - those responsible for measurement data collection and analysis need to be trained in the discipline as well as in the rationale for the procedures used. Again, funding for such activities is a further requirement of a successful program.
- Tool support - software developers can be resistant to change, especially if they perceive the change involves extra workload for themselves. Implementation of a measurement program can be viewed as such a change. Moreover, self-reporting of metrics data may not provide sufficiently accurate or reliable data to enable an organisation to determine current status and levels of improvement. Thus the availability of automated tool support for non-intrusive data collection is another pre-requisite for successful metrics programs. Management-oriented monitoring and control tools are also needed to support the administrative tasks associated with measurement programs.
- Non-chaotic development environment - it almost goes without saying that a managed software process is a necessary foundation on which a measurement program can be built. Chaotic development by its very nature does not enable measurement as part of a long term improvement framework.

Quality in other Disciplines

An interesting question to pose at this stage is "How does the management of quality in software development differ from other disciplines". A reason for raising this question here are that there is always more to be learned from other fields. If software engineering is to live up to the second half of its name, then much must be borrowed from the engineering field.

When a building is to be constructed a multitude of activities are performed before the foundation is laid to ensure the correctness of the structure. Surveying is performed to determine the exact location for the structure. Equations are used to verify the soundness of the structure, possibly simulations will be used for the same purpose. The building will be tested against both natural hazards such as earthquakes, and also man-made such as fires. Standards must be followed for plumbing and electrical work. All of these drivers of quality are used as a matter of course. No builder would claim to be more mature than another simply because he employed a certified electrician.

Why is it then that measurement and quality-improvement programs for software development are often regarded as significant and a sign of process maturity in both practice and much research? Surely, it should be the lack of such basic engineering requirements that attract attention.

Standards and Programs for Quality

An important aspect of a measurement-driven quality improvement program is the use of quality standards and programs, such as:

- ISO9001 and its guide for software ISO9000-3 (Huyink and Westover 1994);
- Software Process Improvement and Capability Determination (SPICE) (Dorling 1993);
- Bootstrap (Bootstrap Project Team 1993);
- the Software Engineering Institute's (SEI) Capability Maturity Model (CMM) (Paulk et al. 1993a; 1993b).

In some of these, measurement is binary (in other words no partial achievements are recognised). This is obviously not adequate in rewarding an organisation in the early stages of quality improvement when many significant building blocks may be in place, but none are entirely completed. The use of such programs is often politically motivated, rather than installed by a genuine desire to improve quality through the use of the program, the accreditation or implementation being used as a marketing tool. For example, Dion (1995, p.2) remarks that "One of the reasons that companies were (and still are) driven by the desire to achieve a particular [maturity] level by a specific date is the knowledge that some government contracts are being awarded using SEI level as a selection criteria." Thus the motivation is really not one of quality *per se* but of competitiveness (Krasner 1994). Whilst there can be no argument that all organisations should wish to maintain and extend their competitiveness, quality may begin to take a less important role if the often significant costs of a quality program are seen as dispensable as contract bids become more and more cut-throat.

An important determinant of whether a standard or program is suitable for a given organisation is the type of development. There are several different types of software as described by McManus (1992), namely, operating systems, mission-critical, real-time, interactive, and business. Each of these has its own quality requirements and any standard or program adopted should provide this level of quality.

The implication of adherence to a standard is that measurement and software quality will follow: "If they are certified for ISO9001 or have a well defined process, organisations will normally have procedures in place that can be used in measurement data analysis." (ami 1995). This is by no means certain, however, particularly as the existence of procedures does not guarantee appropriate use or subsequent analysis and improvement.

Campbell (1995) also suggests that, since these quality assessment frameworks are centred around the interview/audit approach, realistic outcomes are far from certain for a number of reasons:

- the approach fosters a lack of trust between the participants
- organisations tend to hide problems from assessors
- organisations may be subjected to several customer-sponsored audits in a relatively short period, using different instruments and producing different outcomes
- the processes are subject to wide interpretation
- the results are not optimised to business goals or organisational needs
- the costs of assessment and improvement can be prohibitive.

It is an indication of the inherent link between measurement programs and general quality improvement frameworks that an almost identical list could be drawn up as describing those factors that result in measurement program failure.

The inconsistencies between the various quality programs can also be disconcerting for organisations wishing to improve their processes via measurement. The two most widely used frameworks, the SEI's CMM and the ISO9000 set of standards, are substantially different in terms of expectations of the measurement function. This is to be expected, given the difference in focus of the two approaches - the CMM is an evaluation of the 'maturity' of an organisation's software function, whereas the ISO standards are concerned with organisational quality management systems. The fact remains, however, that software organisations need to be aware of both approaches, and some means of matching between the two is useful. Paulk (1994, p.11) provides the following assertion regarding measurement in the ISO standards: "ISO 9001 is somewhat ambiguous about the role of measurement in the quality management system... ISO 9001 requires that quality objectives be defined and documented, not that they be quantitative." Although this may be perceived as a somewhat biased view of the ISO framework - Paulk was one of the main proponents of the CMM - there is certainly some evidence that the ISO standards are more static and binary in their assessment.

To illustrate the problems of inconsistency between quality frameworks, Paulk (1994) states that there are organisations with ISO certification (implying quality) but that are at just Level 1 (the ad hoc or chaotic level) of the CMM capability framework. Furthermore, Paulk et al. (1995, p.11) remark that “Although the SEI is working with ISO’s SPICE project to build the best possible international standard, our participation does not imply a commitment to use the standards eventually approved.” This may be bewildering to software development organisations looking to adopt the ‘best’ quality framework. On the other hand, the CMM approach has been criticised for its absence of attention to quality from a customer perspective - this is surely equally important for an organisation’s long term prospects as the adequacy and improvement of software processes (Denning 1992). Supporters of the CMM will suggest that a quality process will inevitably lead to satisfied customers, but the determinants of quality from each perspective are quite distinct. It is this absence of a customer focus in the CMM that has been addressed, for instance, in the Trillium model for telecommunications software (Bell Canada 1994). Denning (1992) provides further comment in this regard, suggesting that there is now *too much* emphasis on quality from the developer perspective, to the detriment of customer-centred quality. The assertion is that, since it is the *customer* that evaluates product quality based on the work they do with software, measurement should be focused on assuring customer oriented satisfaction. In a similar vein, Dion (1995) and Hon (1990) suggest that, particularly for organisations at the lower levels of software process maturity, the focus of improvement should be on customer- and/or shareholder-oriented activities. Dion acknowledges that these are difficult to measure, but this can be achieved through consistent and objective definition.

The existence of quality frameworks has yet to have the expected impact. The ISO standards have been long established, with assessment and certification becoming particularly popular in the last eight years. The CMM framework has been evolving for a similar period. However, Dion (1995) reports that in 1994 the SEI suggested that 73% of organisations involved in software development in the USA (the market most influenced by the CMM) were still at Level 1 (the ad hoc level) maturity. The reasons for the poor extent of infiltration of such frameworks, that may be equally applied to the use of measurement programs, are as follows:

- Cost - there is still little empirical evidence of the costs incurred as a result of adopting a measurement-driven quality approach (Jones 1996), although some information has been made available through the SEI (Goldenson and Herbsleb 1995). They report that two thirds of 138 respondents who had undergone process appraisals had found the costs of process improvement to be greater than they had expected. Moreover, this does not consider the costs of assessment preparation or those associated with the appraisal itself. In this area, Herbsleb et al. (1994) report a median cost of US$1375 per software engineer per year of process improvement.

- Scale of assessment - the CMM is more than 500 pages long, with potentially 316 clauses to consider; the Trillium model developed by Bell Canada includes more than 500 clauses for assessment at various levels; the Bootstrap framework is applied over a period of a week; the Basic Practices Guide of the SPICE framework includes potentially more than 900 issues to be considered; and quality management systems due for ISO certification must be assessed across the entire organisation. These factors tend to illustrate the significant scale of each quality assessment framework, making their applicability for small projects and small organisations (common in Australasia) less certain. The People-CMM (Curtis et al. 1995) and the Personal Software Process (Humphrey 1996), which have been developed to in part address this issue, are even less prominent in industry. Analysis of organisations undergoing Bootstrap assessment also suggests that there is a positive relationship between organisation size and capability level (Lebsanft 1996), providing seemingly little encouragement for small software organisations. It seems equally plausible, however, that small organisations are equally likely to achieve quality as their larger counterparts - a different method of assessment may be required in these cases.
- Return - the payback has been longer in coming than expected for those involved in CMM appraisal - approximately two to three years pass before benefits are obtained (Curtis et al 1992; Hayes and Zubrow 1995). Given the scale of investment, this may be a significant deterrent for those considering program adoption. Many felt that assistance on *how* to improve was needed after appraisal. Similarly, many of those who had undergone assessment felt that the recommendations that were made as a result were too ambitious to achieve in a reasonable time period.
- Inadequate coverage - discussion relating to some of the inadequacies of the ISO standards in terms of measurement (from a CMM perspective) appeared above. Thirty-eight percent of CMM survey respondents, however, felt that the assessment missed important operational areas.

Realism in the Drive for Quality

The goal of this paper is not to suggest that quality should not be a goal. It should in fact be paramount amongst the targets of any software development organisation. However, the organisation needs to keep a larger perspective - realistic expectations form the basis for realistic plans and estimates, leading to real satisfaction from quality software development. Schedules and budgets are indeed routinely exceeded, but often because they are based on unrealistic measurements and estimates. Moreover, for all the attention we can place on the software process, there is still extensive uncertainty in large scale software development that we simply cannot always control (Kraut and Streeter 1995; Mackey 1996). Human nature means that there is a focus on failure - success is expected and generally not so widely reported. But there have been many successes in software development, particularly when it is considered that software systems are so pervasive in spite of the fact that this is an extremely young industry. Finally, software development is at least in part a social activity that cannot always be modelled and constrained as we would prefer - thus even the most comprehensive measurement-driven software quality framework cannot *ensure* success (King and Galliers 1994).

Conclusions

This paper has considered the many available means of trying to improve the quality of a software development process. While many different tools and methods are available, along with a multitude of standards and quality-improvement programs, one essential element is that of measurement. It is simply unfeasible and unrealistic to expect a processes quality to improve without some form of assessment and feedback.

Bibliography

ami (ami Consortium) (1995). ami - Application of Metrics in Industry, ami Consortium, London.

Arthur, J.D., Nance, R.E., and Balci, O. (1993). Establishing Software Development Process Control: Technical Objectives, Operational Requirements, and the Foundational Framework. *Journal of Systems and Software* 22: 117-128.

Bell Canada (1994). Trillium: Model for Telecom Product Development and Support Process Capability. Bell Canada, Quebec, 1994.

Bootstrap Project Team (1993). Bootstrap: Europe's Assessment Method. *IEEE Software* May: 93-95.

Campbell, M. (1995). Tool Support for Software Process Improvement and Capability Determination: Changing the Paradigm of Assessment. *Software Process Newsletter* 4: 12-15.

Curtis, B., Kellner, M.I., and Over, J. (1992). Process Modeling. *Communications of the ACM* 35(9): 75-90.

Curtis, B., Hefley, W.E., Miller, S., and Konrad, M. (1995). The People-CMM. *Software Process Newsletter* 4: 7-10.

DeMarco, T. (1982). Controlling Software Projects. Yourdon, New York.

Debou, C., Liptak, J., and Schippers, H. (1994). Decision Making for Software Process Improvement: A Quantitative Approach. *Journal of Systems and Software* 26: 43-52.

Denning, P.J. (1992). Editorial - What is Software Quality? *Communications of the ACM* 35(1): 13-15.

Dion, R. (1995). Starting the Climb Towards the CMM Level 2 Plateau. *Software Process Newsletter* 4: 1-2.

Dorling, A. (1993). SPICE: Software Process Improvement and Capability Determination. *Information and Software Technology* 35(6/7): 404-406.

Goldenson, D.R., and Herbsleb, J.D. (1995). After the Appraisal: A Systematic Survey of Process Improvement, its Benefits, and Factors that Influence Success. Technical Report CMU/SEI-95-TR-009, Software Engineering Institute, Pittsburgh.

Gottesdiener, E. (1996). What Is Your Development Maturity? *Application Development Trends* March: 60-73.

Hayes, W., and Zubrow, D. (1995). Moving On Up: Data and Experience Doing CMM-Based Process Improvement. Technical Report CMU/SEI-95-TR-008, Software Engineering Institute, Pittsburgh.

Herbsleb, J., Carleton, A., Rozum, J., Siegel, J., and Zubrow, D. (1994). Benefits of CMM-Based Software Process Improvement: Initial Results. Technical Report CMU/SEI-94-TR-14, Software Engineering Institute, Pittsburgh.

Hollom, J.H., and Pulford, K.J. Experience of Software Measurement Programmes and Application of the ami Method Within GEC. *GEC Journal of Research* 12(1): 17-25.

Hon, S.E. III, (1990). Assuring Software Quality through Measurements: A Buyer's Perspective. *Journal of Systems and Software* 13: 117-130.

Humphrey, W.S. (1996). Using a Defined and Measured Personal Software Process. *IEEE Software* May: 77-88.

Huyink, D., and Westover, C. (1994). *ISO 9000*. Irwin, New York.

Jeffery, R., and Berry, M. (1993). A Framework for Evaluation and Prediction of Metrics Program Success. In Proceedings of the First International Software Metrics Symposium, Baltimore MA, IEEE Computer Society Press, pp. 28-39.

Johnson, J. (1995). Chaos: The Dollar Drain of IT Project Failures. *Application Development Trends* January: 41-47.

Jones, C. (1995). Patterns of Large Software Systems: Failure and Success. *Computer* March: 86-87.

Jones, C. (1996). The Pragmatics of Software Process Improvements. *Software Process Newsletter* 5: 1-4.

Juran, J.M. (1979). "Basic Concepts" in Quality Control Handbook, ed. Juran, J.M., Gryna, F.M., and Bingham, F.M., 3rd Edition, New York: McGraw-Hill, pp. 2-5.

Keil, M. (1995). Pulling the Plug: Software Project Management and the Problem of Project Escalation. *MIS Quarterly* December: 421-447.

King, S., and Galliers, R. (1994). Modelling the CASE Process: Empirical Issues and Future Directions. *Information and Software Technology* 36(10): 587-596.

Koch, G.R. (1993). Process Assessment: the Bootstrap Approach. *Information and Software Technology* 35(6/7): 387-403.

Krasner, H. (1994). The Payoff for Software Process Improvement (SPI): What it is and How to get it. *Software Process Newsletter* 1: 3-8.

Kraut, R.E., and Streeter, L.A. (1995). Coordination in Software Development. *Communications of the ACM* 38(3): 69-81.

Lebsanft, E. (1996). BOOTSTRAP: Experiences with Europe's Software Process Assessment and Improvement Method. Krasner, H. (1994). *Software Process Newsletter* 5: 6-10.

Mackey, K. (1996). Why Bad Things Happen to Good Projects. *IEEE Software* May: 27-32.

McGarry, F. E. (1995). Product-Driven Process Improvement. *Software Process Newsletter* 3:1-3.

McManus, J.I. (1992), "How Does Software Quality Assurance Fit In?", in Handbook of Software Quality Assurance, ed. Schulmeyer, G.G., and McManus, J.I., New York: Van Nostrand Reinhold, pp. 14-24.

Paulk, M.C., Curtis, B., Chrissis, M., and Weber, C.V. (1993a), Capability Maturity Model for Software (Ver 1.1), Technical Report CMU/SEI-93-TR-24, Software Engineering Institute, Pittsburgh.

Paulk, M.C., Weber, C.V., Garcia, S.M., Chrissis, M., and Bush, M. (1993b). Key Practices of the Capability Maturity Model (Ver 1.1), Technical Report CMU/SEI-93-TR-25, Software Engineering Institute, Pittsburgh.

Paulk, M.C. (1994). A Comparison of ISO 9001 and the Capability Maturity Model for Software. Technical Report CMU/SEI-94-TR-12, Software Engineering Institute, Pittsburgh.

Paulk, M.C., Konrad, M.D., and Garcia, S.M. (1995). CMM Versus SPICE Architectures. *Software Process Newsletter* 3: 7-11.

Rada, R. (1996). ISO 9000 Reflects the Best in Standards. *Communications of the ACM* 39(3): 17-20.

Schwaber, K. (1996). Defining Process vs. Problem-Solving. *Application Development Trends* March: 76-81.

Software Engineering Technical Committee of the IEEE Computer Society (1983). IEEE Standard Glossary of Software Engineering Terminology, IEEE-STD-729-1983, New York: IEEE, p.32.

SPC (Software Productivity Centre) (1994). Metricate: Metrics Implementation Guide for Software Quality Professionals, Software Productivity Centre, Vancouver.

US Government Accounting Office (1979). *Contracting for Computer Software Development-Serious Problems Require Management Attention to Avoid Wasting Additional Millions.* Report FGMSD-80-4. November.