The current issue and full text archive of this journal is available at
www.emeraldinsight.com/0954-478X.htm

Software engineering: a quality management perspective

John McManus
University of Lincoln, Lincoln, UK, and
Trevor Wood-Harper
University of Manchester, Manchester, UK

Abstract

Purpose – The purpose of this paper is to examine the concept of quality related to the context of software development using the ISO, TickIT and CMM frameworks. The paper also seeks to stress the fact that the different perspectives of those involved in software development will influence how quality is seen and measured. In the context of software engineering projects, quality takes on a broad meaning that refers not only to the way in which companies manage software engineering projects, but also to the software development process itself.

Design/methodology/approach – The approach and methodology adopted for this paper were a review of the literature and best practice in software engineering. It is argued that users of software systems are more interested in how easy the software is to use than in the underlying application code that is used to generate the system. Using the body of knowledge that is software quality the basic characteristics of software quality are described and compared in terms of quality standards such as ISO, TickIT and CMM. Each of these standards is decomposed further in order to clarify its usefulness.

Findings – The findings in the paper suggest that, whilst there are many differences in the quality standards used, there are a number of similar characteristics. In essence the underlying philosophies of ISO and CMM have at the core the same goals. Some academics see CMM as being technically over-engineered; a CMM-compliant quality system is in many respects far in advance of ISO.

Research limitations/implications – This paper helps define the strengths and weaknesses within ISO, TickIT and CMM from a software engineering practitioner perspective.

Practical implications – The paper shows that software engineers need to pay more attention to the performance and conformance issues in software projects and to be proactive rather than reactive to quality issues.

Originality/value – It may be argued that the importance of this paper lies in the assertion that those engaged in the software engineering are in need of a multi-perspective view on quality and, with that in mind, this paper should appeal to practitioners and members of the academic community with an interest in software quality.

Keywords Quality, ISO 9000 series, Quality standards

Paper type Conceptual paper

Introduction

According to Juran (1988), 90 per cent of all problems in companies are systematic and beyond the positive influence of staff. Unfortunately, these problems also include issues related to quality. This means that one of the greatest challenges faced by project managers and software professionals alike is to get the process right first time. In turn, this means initiating a rigorous quality regime, which is both defined and measurable. A widely used definition of quality has been supplied by the International Organisation for Standardization (ISO8042, 1988)[1]: "The totality of features and characteristics of a product or service that bear on its ability to satisfy specified or

The TQM Magazine
Vol. 19 No. 4, 2007
pp. 315-327

0954-478X
DOI 10.1108/09544780710756223

implied needs". This view of quality has come to dominate the quality movement in many software companies. Software professionals generally acknowledge that the quality of a product (or software system) is very much influenced by the quality of the processes used to build it. This means in order to build high quality software systems, it is necessary to have high quality processes and people committed to quality (Crosby, 1979).

It is also recognized (Gillies, 1996; and McManus, 2000) that software quality has its own peculiarities. Take the following as an example: in 2002, I arranged a workshop for a specialist group of software developers. Those who attended were asked "why software quality was different from other types of quality". It was suggested that each type of product made had its own quality demands but that software was particularly problematical for the following reasons:

- Software has no physical existence.
- The lack of knowledge of client needs at the start.
- The change of client needs over time.
- The rapid rate of change in both hardware and software.
- The high expectations of customers, particularly with respect to adaptability.

Barker (1992), his observations on quality was that the quality is by large motivated by an individuals expectations and perceived value. Barker's point is that people determine quality, not just procedures, tools or systems. After all, it is people that define the problems and specify the solutions.

Quality concepts

Software quality may be described as a paradigm that comprises several concepts (McManus, 2000). The first of these concepts relates to defining the software process to be improved. Defining the process means that all the activities to be performed have to be clearly stated, including the order in which they are to be performed and when they are considered complete. This is normally achieved by expressing exit criteria. The second concept relates to using software processes – to improve a process it needs to be used on many projects. Improvement comes with experience. When the same process is used over different projects, it is always possible to find ways in which the process can be improved. Such improvements are usually small and relatively easy to implement. Over time, such small improvements lead to significant savings and have a positive financial benefit. The third concept is that of metrics, which should be collected to determine if changes incorporated into the process, are really improvements. If so, there is a need to quantify the improvements noted. Measurement is a fundamental aspect of quality, reflected in the proverb: what gets measured gets managed; and what gets managed gets improved. Flood (1993) emphasizes the importance of this by commenting that: "... measurement is crucial in problem solving. Measurement specifications are the basic parameters by which intervention is guided. The choice of specification is therefore a crucial one".

Much of the published literature cites a set of either four or five core metrics for example, Khoshgoftaar and Seliya (2002), and Rangarajan *et al.* (2001). Whilst the names and nature of these metrics varies the data elements commonly identified within

each of the attributes are usually related as shown in Table I. The measures in this table are not the only ones that can be used to describe software products and processes, but they represent a starting point and are practical measures that produce useful information. What is more, they are measures that the project manager should be able to define in ways that promote consistent use. The measures stated in Table I should be accompanied by checklists that an individual project manager can use to specify and obtain supporting data to address management issues that are important to the project organisation.

Qualifying and quantifying software quality

Apart from aesthetic appreciation of quality products, our purpose in examining quality is not only to improve the software process but also to facilitate decision-making. One example of these decisions is in the choice of software tools, where there are several applications that might meet requirements. Another example is the decision of whether to accept, and to pay for, a product that claims to meet a particular need. Many software engineering project managers are often concerned with the issue of quality versus price, that is, what quality is available for a given price, or how much extra better quality would cost. Consequently, another example is in deciding what investment tradeoffs are worth making in order to improve the quality of a given software system or product. In many cases, what we really want to do is predict what our own level of satisfaction with the software will be, before we have had the chance to exercise the software extensively in a particular context (Gentleman, 1996). This comes up both when the software is unfamiliar to us, and when it is not yet complete.

Software quality is often defined in terms of fitness of the product for its purpose. Different people however, have different purposes for the same software. A casual user is probably more concerned about ease of learning and about robustness against

Name of metric	Potential definitions	Characteristics addressed
Size	Counts of physical source lines of code (SLOC)	Size, progress, reuse, rework
	Function points or feature points	Size, progress, reuse, rework
Effort	Number of staff hours expended monthly	Effort, cost, rework, resource allocations
Software quality	Total number of errors opened/closed	Quality, readiness for delivery, improvement trends, rework
	Number of errors opened/closed since last report	Quality, readiness for delivery, improvement trends, rework
	Type of error (testing, action item, document comment)	Quality, readiness for delivery, improvement trends, rework
	Classification and priority	Quality, readiness for delivery, improvement trends, rework
	Product in which error was found	Quality, readiness for delivery, improvement trends, rework
Rework	Number of open software change orders	
	Number of closed software change orders	
	Total number of software change orders	

Table I.
Core metrics

misuse rather than efficiency. On the other hand, a system integrator planning to incorporate the software into some larger system might be more concerned about failure detection and recovery than ease of installation. The point being made is that quality is many-sided, and the importance of the different facets changes with the context, even for the same person at different points in time. As an example, most users would agree that they want secure software, that is, software free from defects, including viruses. Users want applications that provide safe manipulation of data, meaning that the user understands when data is changing due to the actions of the software. However, the consequences of providing absolute software security may cause the product to be deemed less attractive (i.e. usable) to one class of user versus another class of user. The common "are you sure you want to . . . " prompt is source of much derision for the "sophisticated" user but the neophyte user can take great comfort from this extra little bit of security. This may seem like a trivial point, but it is a simple example of how quality standards should be judged within the context of the user population.

Software quality characteristics and requirements

Many of the above attributes are explicit in various quality standards. As an example, the International Standard ISO/IEC 9126[2] offers a model that lists six characteristics: functionality, reliability, efficiency, usability, portability and maintainability. This decomposition reflects the viewpoint of users and introduces the concept of quality in use: users are mainly interested in using the software product, and evaluate software mostly from the viewpoint of the performance and the service it provides, rather than on the basis of internal aspects or the development process.

Information system requirements are usually categorized as functional and non-functional. As functional requirements address what the software can do, while non-functional requirements are concerned with the overall qualities of the system. The attributes discussed previously are of a non-functional nature. In carrying out system development projects, a number of problems have been identified with non-functional requirements to which the project manager should pay attention (Kontonya and Sommerville, 1998):

- Some non-functional requirements are related to a design solution that is unknown at the requirement stage.
- Some non-functional requirements are highly subjective, especially those associated with human engineering.
- Non-functional requirements have great diversity.
- Non-functional requirements and functional requirements are related methods that separate them out and make it difficult to see the correspondence between them.
- Non-functional requirements tend to conflict and therefore need to be treated as tradeoffs.

Many of the issues listed above can be seen in structured methods, such as SSADM[3], which places great emphasis on capturing functional requirements, supported by techniques such as data flow diagramming, data modelling, and a requirements

catalogue. Although non-functional requirements are dealt with in SSADM there is no systematic approach to capturing them, no method support, and no mechanism for trading off conflicts between them.

Software quality metrics

Not all these software characteristics have equal weighting with respect to metrics. Watts (1987), for example, identified some 40 individual software metrics. Some 34 of these metrics are associated with the criteria of Maintainability (18), Reliability (12) and Usability (4). This uneven distribution of metrics seems arbitrary and calls into question the validity of any given metric. As an example, complexity is used as a handle on both reliability and maintainability, and 13 measures are described as based upon complexity. According to Bevan (1997), in order to specify or measure quality in use, it is necessary to decompose effectiveness, efficiency and satisfaction and the components of the context of use into sub-components with measurable and verifiable attributes. Measures of effectiveness relate the goals, or sub-goals, of the user to the accuracy and completeness with which the goals are achieved. In contrast, measures of efficiency relate the level of effectiveness achieved to the expenditure of resources. Measures of satisfaction describe the comfort and acceptability of the use of the product.

Another ISO standard, ISO 9241-11 (1998)[4], explains how quality in use can be measured in terms of user performance and satisfaction by the extent to which the intended goals of the user are achieved. This standard includes the resources that have to be expended to achieve the stated and intended goals, and the extent to which the user finds the use of the product acceptable. Measures of user performance and satisfaction assess the quality in use of a product in the particular context of use provided by the rest of the working environment.

Quality management frameworks

An attempt to put metrics on a systematic level has been made by the European METKIT project (Fenton, 1991). The use of metrics to measure criteria is part of an overall framework that includes quality control and assurance, quality models, reliability models, and performance evaluation. Over the years, a number of quality frameworks have been developed to enable companies (including suppliers of software services) to measure and improve their product offerings to customers (Sanders and Curran, 1994). At an organisation level the most widely adopted are ISO 9000 (and TickIT at the individual project level) and the Capability Maturity Model (CMM).

The ISO 9000 framework

The International Organisation for Standardization's (ISO) 9000 is a series of quality assurance standards with application for any business, whether in manufacturing, service, retail, or government, in producing a product or service. Popular in Europe, ISO 9000 is rapidly taking hold in the USA and around the globe. Some 60 countries, including the USA, Canada, Japan, and the members of the European Community have adopted ISO 9000 series standards. Table II lists the components of ISO 9000. As shown by the table, the ISO 9000 framework is a deep, vertical quality system. That means that it creates a system that tracks and controls a consistent set of factors

Standard	Description
ISO 9000	Quality management and quality assurance standards – Guidelines for selection and use (1987)
ISO 9000-9001	Revision of ISO 9000 (1991)
ISO 9000-9002	Guidelines for the application of ISO 9001, ISO 9002 and ISO 9003 (1991)
ISO 9000-9003	Guidelines for the application of ISO 9001 to the development, supply and maintenance of software (1991)
ISO 9001	Quality systems – Model for quality assurance in design/development, production, installation and servicing (1987) A more detailed standard, which covers design, development, production, installation, and servicing, this applies to the software industry
ISO 9002	Quality systems – Model for quality assurance in production and installation (1987) Assesses the production and installation processes
ISO 9003	Quality systems – Model for quality assurance in final inspection and test (1987) Evaluates the final inspection and test phase
ISO 9004	Quality management and quality system elements – Guidelines (1987) Defines the 20 fundamental quality system concepts included in the three models

Table II.
Components of ISO 9000

involved in quality and service to the customer. ISO 9001 is applicable in situations in which there is a substantial element of design. In situations in which design is predefined then ISO 9002 provides a focus on production. Where there is little or no production, then ISO 9003 is applicable. The ISO 9000 standard family requires that whatever process is chosen for development should be understood and documented and should be monitored to ensure it is actually used.

Companies using this framework should have gone through a three-tier accreditation process that involves self-assessment, customer assessment and a third party assessment by an independent standards body. ISO 9000 is a powerful incentive for companies to get their quality procedures right. Accreditation is a powerful evidence of this fact. From the information contained in Table II it is clear that ISO 9000 is intended to be generic so that it can serve a broad range of companies. However, ISO 9001 is intended for applications where there is a significant design element. Since most software applications require significant design input, ISO 9001 is generally the standard applied within the software development industry (McManus, 2000).

Acceptance

The primary purpose of applying the ISO 9000 framework is the confidence it will afford prospective clients, management, and development personnel that the company system for managing software quality is efficient, effective and measurable. According to Gillies (1996), however, one of the biggest barriers to acceptance of ISO (particularly ISO 9001) amongst information technology practitioners is its generic nature and its origins as a manufacturing standard. Although ISO 9001 has been applied in many service industries, information technology people still feel it is inappropriate and difficult to apply (Shelley, 1994, and McManus, 2004). The response to this from the standards bodies is to issue notes for guidance on the application of the standard to

software development (an example of such guidance was given in Table II). It should be stressed that these do not supersede the standard, but rather amplify its contents with the aim of explaining how the standard should be applied in a software context.

The UK TickIT system

In 1991 the British Standards Institute of the United Kingdom (BSI-UK) introduced TickIT into the quality vocabulary. TickIT uses the checklist approach to gather detailed information on quality related processes. The aim of the TickIT system is to assist the take-up of ISO 9001 by the information technology sector and ensure that suppliers, purchasers and assessors have an overlapping understanding of the requirements in the IT sector. TickIT (1992), is a certification scheme developed to apply ISO 9000, but with the advantage of having been tuned to deal with special requirements of software development. Its main principles and objectives are:

- The interpretation of ISO 9001 for the information technology sector;
- The need to ensure continuing conformity for certified suppliers;
- The necessity to perform assessments with experienced and skilled assessors as witnessed by their ability to satisfy examiners; and
- The benefit of accredited training and examination for entrants on the assessor register.

The TickIT Guide emphasizes the establishment of a "delivery chain" from supplier to customer by means of a quality management system, which is documented, implemented and audited. The TickIT guide includes the following information:

- Purchaser's guide.
- Supplier's guide.
- Auditor's guide.

The Supplier's guide is aimed at software sector companies requiring ISO 9001 certification and also includes guidance on the standard's application to support and service activities. The TickIT guide provides background to the TickIT scheme, including its origins and objectives, how to implement a quality system and the expected structure and content relevant to software activities.

Independent assessment of a company's quality system against ISO 9001 provides confirmation that it has achieved a base line level of performance for its quality-related processes and practices. In the UK, all recognized (accredited) certification bodies are required to perform an ISO 9001 assessment in the software sector under the TickIT scheme, which ensures the use of trained and experienced IT auditors and recognizes the guidance provided by ISO 9000-9003 (1997) and The TickIT Guide.

Maximizing the benefits from TickIT

Although generally viewed as a certification scheme, this is not its primary purpose. The main objectives are to stimulate information technology companies to think about how they can benefit from a quality system and how quality performance may be achieved.

Certification is an end process, which seeks to confirm that whatever the organisation declares as necessary to their quality system is put into use and is effective. In doing so, certification assures that the appropriate parts of the quality system standard are addressed satisfactorily. The quality system provides a means of ensuring that quality is delivered. It determines how requirements are processed as input and how these are transformed into products and services. By its presence or absence, the quality system has a direct influence on the quality of product and service. Although, it is easy to measure the effectiveness of such programmes in terms of the number of firms achieving certification, it is sometimes less easy to quantify the overall effect upon software quality. ISO and TickIT share a common goal with quality. Each is driven by similar concerns and intuitively correlated.

The capability maturity model

The Software Engineering's Institute's (SEI) Capability Maturity Model (CMM) has evolved to focus on product quality through a maturity framework that includes five basic steps (Paulk, 1993). Like ISO 9000, these steps must have full management sanction and commitment to:

(1) Establish basic management control.

(2) Set quality standards.

(3) Define the process.

(4) Measure and evaluate.

(5) Institute continuous process improvement.

The CMM is a five-level model that attempts to quantify a software organization's capability to constantly and predictably produce high-quality software products. The model is designed so that capabilities at lower stages provide progressively stronger foundations for higher stages. Each development stage or maturity level distinguishes an organisation's software capability. For each maturity level there are associated key process areas (KPAs). The KPAs identify the requirements for achieving each maturity level. Level 1 does not include KPAs since it is the starting point. Table III shows the maturity levels and their associated KPAs (McManus, 1999a).

In essence, each of the maturity levels outlined in Table III is a well-defined step towards achieving a mature software process. Each maturity level provides a layer in the foundation for continuous process improvement. Each level comprises a set of goals that, when satisfied, will stabilize an important component of the software process. Achieving each level of the maturity framework establishes a different component in the software process, resulting in an increase in the process capability of the organisation. What is important here is the organisation's understanding of the level of commitment and focus required for achieving levels of maturity. This commitment must be supported and encouraged by both managers, and software professionals. Otherwise, the ability to realize process maturity goals will be sporadic at best and probably unattainable. Those characteristics identified with immature and mature companies are summarized in Table IV.

SEI CMM definition	KPAs
Initial The processes are special and mostly defined. Success depends upon the individual effort	None
Repeatable Basic project management processes to track cost, schedule and functionality Tools are in place to repeat success achieved on analogous programmes	Requirements management Software project planning Software project tracking and oversight Software subcontract management Software quality assurance Software configuration management
Defined The software process is organisation-wide and is employed by both management and engineering. The process is documented, standardized and integrated	Organisation process focused Organisation process definition Training programme Integrated software management Software product engineering Inter-group coordination Peer reviews
Managed The detailed measures of the software process are collected, managed, quantified, understood and controlled	Quantitative process management Software quality management
Optimized The software process continuously improves by quantified feedback from the process and testing new and creative ideas and technologies	Defect prevention Technology change management Process change management

Table III.
CMM level framework

Immature	Mature
Over budget	Proactive disciplined/consistent
Late delivery	Defined processes
Undefined processes	Defined roles
Reactive	Consistent monitoring
Crisis management	Predictive results
Poor quality	On time/within budget
Overworked/confused staff	Enabled staff
Unsatisfied customers	Satisfied customers
	Good communications
	Processes institutionalized
	Information systems viewed as strategic

Table IV.
Characteristics of mature and immature companies

Key process areas

A KPA contains the goals that must be reached in order to improve a software process. A KPA is said to be satisfied when procedures are in place to reach the corresponding goals. These key indicators offer an insight into whether the goals have been satisfied. When an organisation collectively performs the activities

defined by the KPAs, it can achieve goals considered important for enhancing process capability (McManus, 1999b).

A software organisation can only claim to have reached a given maturity once all corresponding KPAs are satisfied.

CMM assessments

The CMM provides both internal and external assessments. Key indicators form the basis for the Software Engineering Institute's maturity questionnaire used to assess the capability of software companies' internal processes. This assessment questionnaire contains 120 questions, where repeatable and defined levels contain about 40 questions each and managed and optimized levels contain about 20 questions each. Furthermore, a profile template is used which lists each KPA, such that they can be checked as not satisfied, partially satisfied, or fully satisfied. As stated earlier, an organisation maturity level is set at the highest level at which it satisfies all KPAs.

In essence, CMM's capability evaluation has the same objective as ISO 9000s third party audits. Both have been developed to check the overall capability of a software organisation to produce software in timely, repeatable fashion. The only difference is that in a CMM capability evaluation, a software organisation is ranked according to the five levels, and in an ISO 9000 audit, a software organisation is checked to see that it follows a given set of standards. Although some quality managers see the CMM as technically over-engineered, a CMM-compliant quality system is in many respects much more advanced than an ISO 9000-compliant system. ISO 9000 establishes a minimum quality programmes for a software organisation. The CMM establishes a continuous improvement focus. Whereas ISO 9000 deals with issues of quality control and quality assurance, the CMM talks about the maturity of the process. Table V maps the relationship between CMM and ISO 9001.

Conclusion

In conclusion it is generally accepted that higher CMM levels lead to better quality software products and therefore a better company reputation. CMM compliance may also change the manner in which a company interacts with its customers because there are stringent requirements for maintaining a high maturity level. Highly rated companies are more adept at handling quick demands by the customer. Fortunately, compliance leads to higher quality software at lower cost (Budlong and Peterson, 1996). Also compliance improves a company's reputation, which should be a very potent ingredient for winning and maintaining contracts. Companies participating in CMM are looking at meeting their software quality goals, meeting their requirements, building a maintainable product, and seeking better and improved quality as well as stabilizing schedule, meeting commitments, and accelerating or reducing schedule. Several software companies have experienced a reduction in defects that ranged from as low as 10 per cent to as high as 80 per cent. One organisation reported a 45 per cent decrease in its reduction error rate, while two more companies' product error rates decreased from 2.0 to 0.11 per thousand source lines of code and from 0.72 to 0.13 per thousand non-commented source statements (Brodman and Johnson, 1996).

ISO 9001	CMM relationship	CMM judgemental relationship
Management responsibility	Commitment to perform Software project planning Software project tracking Software quality assurance	Ability to perform Verifying implementation Software quality management
Quality system	Verifying implementation software project planning Software quality assurance Software product engineering	Organisation process definitions
Contract review	Requirements management Software project planning	Software subcontract management
Design control	Software project planning Software project tracking Software configuration management Software product engineering	Software quality management
Document control	Software configuration management	
Purchasing	Software subcontract management	
Control of consumer-supplied product		Software subcontract management
Product identification and tractability	Software configuration management Software product engineering	
Process control	Software project planning Software quality assurance Software quality engineering	Quantitative process management Technology change management
Inspection and testing	Software product engineering Peer reviews	
Control of inspection measuring, and test equipment	Software product engineering	
Inspection and test status	Software product engineering	
Control of non-conforming product	Software configuration management Software product engineering	
Corrective and preventive actions	Software quality assurance Software configuration management	Defect prevention
Handling, storage, packaging, preservation, and delivery		Software configuration management and software product engineering
Control of quality records	Software configuration management Software product engineering Peer reviews	
Internal quality audits	Verifying implementation Software quality assurance	
Training	Ability to perform training programme	
Servicing		
Statistical techniques	Measuring and analysis	Organisation process definition; quantitative process management; software quality management

Table V.
Summary of mapping between ISO 9001 and CMM

Notes

1. International Organisation for Standardisation: Published Standard 1988.
2. Software Quality Characteristics and Metrics.
3. Structured Systems Analysis Design Method.
4. Ergonomic requirements for office work with visual display terminals (VDTs) Part 11: Guidance on usability.

References

Barker, J. (1992), *Future Edge: Discovering the Paradigms of Success*, William Morrow and Company Inc., New York, NY.

Bevan, N. (1997), *Quality and Usability: A New Framework*, Tutein-Nolthenius, Delft.

Brodman, J. and Johnson, D. (1996), "Return on investment from software process improvement as measured by US industry", *Cross-Talk*, April, pp. 23-9.

Budlong, F. and Peterson, J. (1996), "Software metrics, capability evaluation methodology and implementation", *Cross-Talk*, January, pp. 15-19.

Crosby, P.B. (1979), *Quality Is Free*, McGraw-Hill Publishing Company, New York, NY.

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

Flood, R.L. (1993), *Beyond TQM*, John Wiley & Sons, London.

Gentleman, W. (1996), "The quality of numerical software: assessment and enhancement", in Boisvert, R. (Ed.), *The Proceedings of IFIP WG2.5 Working Conference 7, Oxford, 7-12 July*, pp. 32-43.

Gillies, A. (1996), *Case Studies in Software Engineering*, SUBSL.

ISO 9000-9003 (1997), *Guidelines for the Application of ISO 9001:1994 to the Development, Supply, Installation and Maintenance of Computer Software*, 2nd ed., 15 December.

Juran, J. (1988), *The Quality Control Handbook*, 4th ed., McGraw-Hill, New York, NY.

Khoshgoftaar, T.M. and Seliya, N. (2002), "Tree-based software quality estimation models for fault prediction", *Proceedings of the 8th IEEE Symposium on Software Metrics (METRICS 2002), Ottawa, June 4-7*, pp. 203-15.

Kontonya, G. and Sommerville, I. (1998), *Requirements Engineering, Process and Techniques*, John Wiley & Sons, New York, NY.

McManus, J. (1999a), "Climbing the maturity ladder part I", *Project Manager Today*, March, pp. 30-1.

McManus, J. (1999b), "Climbing the maturity ladder part II", *Project Manager Today*, April, pp. 4-5.

McManus, J. (2000), "Quality meets process improvement", *Management Services Journal*, No. 5, pp. 14-16.

McManus, J. (2004), *Risk Management in Software Development Projects*, Elsevier, Butterworth-Heinemann, Oxford.

Paulk, M. (1993), *Capability Maturity Model, Version 1.1*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Rangarajan, K., Swaminathan, N., Hedge, V. and Jacob, J. (2001), "Product quality framework: a vehicle for focusing on product quality goals", *Software Engineering Notes*, Vol. 26 No. 4, pp. 77-82.

Sanders, J. and Curran, E. (1994), *Software Quality: A Framework for Success in Software Development and Support*, Addison-Wesley, Reading, MA.

Shelley, C.C. (1994), "Practical experience of implementing software measurement programmes in industry", *Software Quality Management*, pp. 95-106.

TickIT (1992), *A Guide to Software Quality Management System Construction and Certification using EN29001 (ISO 9001)*, Issue 2.0, 28 February, UK Department of Trade and Industry, London.

Watts, R. (1987), *Measuring Software Quality*, NCC, Blackwell, Oxford.

Corresponding author
John McManus can be contacted at: jmcmanus@lincoln.ac.uk