Software Quality Journal **5**, (1996) 1–23

Frameworks for quality software process: SEI Capability Maturity Model versus ISO 9000

HOSSEIN SAIEDIAN[1] and LAURA M. MCCLANAHAN[2]

[1]*Department of Computer Science University of Nebraska at Omaha, Omaha, NE 68182, USA*
[2]*Science Applications International Corporation (SAIC), Bellevue, NE 68005, USA*

Received 1 August 1994

With the historical characterization of software development as being costly due to massive schedule delays, incorporation of the ever-changing technology, budget reductions, and missing customer requirements, the trend of the 1990s in establishing a quality improvement or a quality assurance programme has been overwhelming. The two popular models or frameworks for assessment of a quality assurance programme are the US government-sponsored Capability Maturity Model (CMM) and the internationally recognized ISO-9000 quality standards. Both of these two frameworks share a common concern regarding software quality and process management. Since it is not clear which of these two frameworks is most effective in achieving their shared objectives, it is valuable and timely to provide an objective overview of both models and to compare and contrast their features for quality software development. Because there are many legitimate areas for comparison, we have selected the two most important as a basis for comparison: (1) the role of management, and (2) the application of measurements. We also provide a summary of the reported impact of these two models on the organizations adhering to their standards, and include our observations and analysis.

Keywords: SEI Capability Maturity Model, ISO 9000 software quality standards, management roles, application of metrics, industrial impact

1. Introduction

In the 1960s, the cost of hardware was the largest cost for implementing information technology. When the price of hardware began to decrease and the true cost of software development was realized in the 1970s, the focus shifted to planning and control of software projects. Phase-based life-cycle models were introduced and information technology permeated every facet of our institutions and personal lives in the 1980s. The importance of productivity in software development increased significantly as competition in the computer industry became keen, low-cost applications became widely implemented, and the number and value of resources available to organizations faced reductions (Elliot, 1993). The importance of quality was recognized and various cost models were developed and used. In the 1990s, quality has been brought to the centre of the software development process. According to Kan *et al.* (1994) 'from the standpoint of software vendors, quality has become a necessary condition to compete in the marketplace.

Historically, software development has been wrought with cost overruns and schedule delays. A recent *Computing Canada* article reported a UK survey where failure costs for software developers are approximately 20% of revenue for firms with incomes of $5 to $10 million (Inwood, 1994). It

further stated that most expenses were connected to prototyping, but around 50% of rework costs could be salvaged by introducing a quality management system. Kan *et al.* (1994) quote that requirement errors constitute one of the major problems of the software industry, specifically about 15% of defect totals.

To combat the historical problems of software development, increase productivity, and retain competitiveness in the marketplace overall, many institutions have turned to quality assurance programmes of some form. Investigated here is the definition of quality, a look at the US government initiated quality assurance framework, the Software Engineering Institute's (SEI) Capability Maturity Model (CMM), and the International Standard Organization's (ISO) set of quality standards referred to as the ISO 9000 standards. Then exploration of recent developments such as attempts at consolidating the two frameworks is performed.

2. Background

Quality, as commonly defined, is the conformance to customers' expectations and requirements (Kan *et al.*, 1994). In other words, 'quality' has two perspectives. The first perspective implies that the ultimate validation of quality is customer satisfaction. The second implies that in order to achieve quality, the producer must have adhered to the customer's requirements. From this concept several views of software quality have been perceived. One view would have the end-product inspected by the customer to ensure the product's quality. Another perspective is to view each stage of the development process as having a product for an end-customer. In this manner, each stage would produce a product that is checked for meeting the requirements of its intermediate user (the next stage). The end-product thus developed and produced will meet the specified requirements. Another view of process quality is aimed at improving the development processes themselves, i.e. defining a set of ideal processes and measuring the existing processes of organizations against these ideals (Humphrey, 1992).

To improve quality during development, models of the development process are needed. Within the process, specific methods and approaches need to be selected and deployed and proper tools and technologies employed. Measures of the characteristics and quality parameters of the development process and its stages are needed. This requires metrics and quality models to help ensure that the development process is under control to meet the quality objectives of the product (Kan *et al.*, 1994).

Two frameworks that have been proposed to improve quality are the SEI's Capability Maturity Model and ISO's 9000 series of quality standards. Each can be considered an organizational framework that presents guidelines for establishing and institutionalizing a quality assurance programme. Furthermore, each requires an assessment or certification process for an organization to claim adherence to quality.

The SEI Capability Maturity Model is a five-stage process improvement based on repeated assessments of the capability of an organization with regard to a set of key process areas. The approach is based on organizational and quality management maturity models where there are defined key process areas that will improve software development. The goal of the approach is to achieve continuous process improvement (reached when level 5 is attained) via defect prevention, technology innovation, and process change management.

ISO 9000 is a set of generic standards for quality management and assurance, established to facilitate the international exchange of goods and services. Its principal quality standard, ISO 9000, is a top-level document that is an overall guide for the use of four supporting standards. The five parts contain guidelines for selecting and using quality management and quality assurance standards; three quality systems models for design or development, production, installation, and

servicing; and a guidance document describing the quality system elements audited during the certification process.

3. The Capability Maturity Model

The Software Engineering Institute (SEI) was establsihed by an Act of Congress to address the need for improved software for the Department of Defense (DoD). In 1986, in response to a request to provide a method for assessing the capability of contractors to produce software on schedule and within budget, the SEI, assisted by the Mitre Corporation, began developing a process maturity framework. Originally, the SEI delivered a model based on the conceptual framework developed by its first director, Watts Humphrey, and a questionnaire (Humphrey and Sweet, 1987) to aid organizations in improving their software process. The model and questionnaire evolved to what is now known as the Capability Maturity Model (CMM).

The primary value of the CMM, which has been advocated as the framework for achieving true 'software engineering' by both government and industrial organizations, lies in its standardization of a model and language for understanding, planning, and measuring process improvement. This framework defines progressively mature software processes into one of five levels of maturity with its associated development process characteristics as shown in Fig. 1.

The five levels are used to gauge an organization's capability by measuring the degree to which processes are defined and managed. Each level is composed of several key process areas organized into one of five sections called common features. For each key process area (KPA), several (usually three or four) goals are defined, and under the common features, more numerous key practices or recommended activities as depicted in Fig. 2.

The common features are attributes utilized in determining whether the effectiveness and repeatability of the implementation or institutionalization of the KPA. The common features are listed below:

- commitment to perform
- ability to perform
- activities performed
- measurement and analysis
- verifying implementation

Commitment to perform typically involves the establishment of policies, senior management sponsorship, and other actions necessary to ensure process establishment and endurance. Ability to perform describes the preconditions, such as resource allocation, training, and organizational infrastructures, necessary for competent implementation of the software process. The roles, procedures, and work activities for the actual implementation are described in the activities performed.

Maturity level	*Characteristics*
Initial	Ad hoc, chaotic. Few processes defined, success depends on individuals
Repeatable	Basic managmeent processes established and process can repeat earlier successes
Defined	Organizational definition of integrated management and engineering process
Managed	Process measured and understood
Optimizing	Continuous improvement fed into process

Fig. 1. CMM levels and characteristics

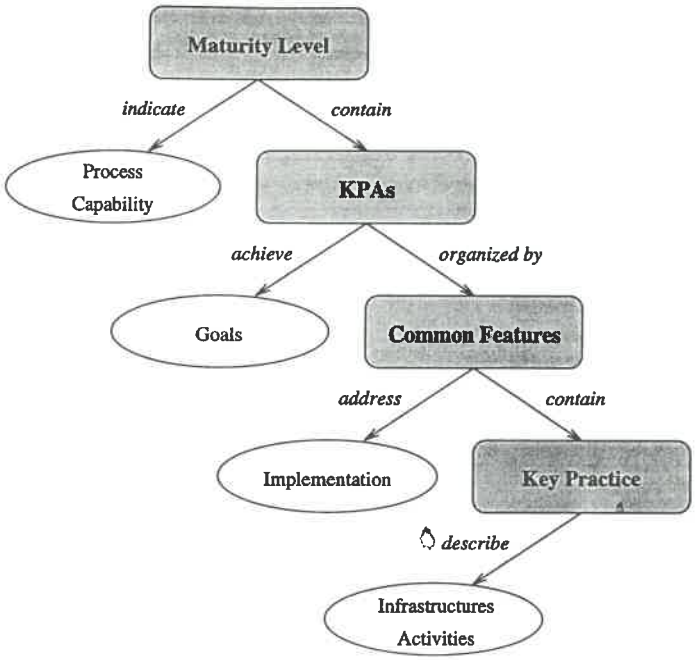

Fig. 2. Organization of the CMM

During the process, measurement and analysis are needed to determine the status and effectiveness of the activities performed. Measurement practices as prescribed by the CMM, i.e. size, effort, staffing, scheduling, costing and planning based on past experience and measurements are described by Baumert and McWhinney (1992). Of course, the implementation must verify that activities are being performed in compliance with the established process.

An organization must satisfy all goals for each KPA to attain the corresponding level of process maturity. The key practices, as further described by Paulk *et al.* (1993a) and shown in Fig. 3, help an organization understand how to achieve the maturity goals; they serve as examples of mechanisms and activities an organization might institutionalize to attain the desired maturity. The specific practices to be executed in each KPA will evolve as the organization achieves higher levels of process maturity. Thus, it must be realized that an organization cannot skip maturity levels. Each level forms the foundation for the next, requiring balanced attention to all KPAs. Good overviews of the model are described by Paulk *et al.* (1993a) in *IEEE Software*.

The levels of maturity are evaluated via self-assessments or evaluations also prescribed by SEI. The SEI has developed two surveys, the Software Process Assessment (SPA) and the Software Capability Evaluation (SCE), which compares and rates the organization against the SEI CMM. The SPA, informally called the self-assessment, is to aid organizations evaluate their software process maturity and identify key areas for improvement. The SCE is an audit or appraisal performed by SEI professionally trained evaluators to either identify qualified contractors or to monitor the state of an existing software effort's software process.

4. ISO 9000 quality standards

The ISO model, developed by the International Standards Organization (ISO) and embodied in the

Maturity level	*Key process areas (KPAs)*
Initial	None
Repeatable	Software Configuration Management Software Quality Assurance Software Subcontract Management Software Project Tracking and Oversight Software Project Planning Requirements Management
Defined	Peer Reviews Intergroup Coordination Software Product Engineering Integrated Software Management Training Programme Organization Process Definition Organization Process Focus
Managed	Software Quality Management Quantitative Process Management
Optimized	Process Change Management Technology Change Management Defect Prevention

Fig. 3. Key process areas for maturity levels

ISO 9000 quality standards, can also be traced to the DoD. In 1959, the agency published its MIL-Q-9858 quality management programme standard. In principle, the programme was adopted nine years later, in 1968, by the North Atlantic Treaty Organization which published the AQAP1, AQAP4, and AQAP9 series of standards. In 1979, the British Standards Institution released commercial versions of these NATO standards as BS 5750 Parts 1–3. With minor changes, the three parts evolved into the 1987 versions of ISO, 9001, ISO 9002, and ISO 9003 (Bamford and Deibler II, 1993).

The ISO 9000 standards, as summarized in Fig. 4, are concerned with the three phases of product development; design, production, and the testing of the final product. The ISO 9001

Standard	*Description*
ISO 9000	Quality Management and Quality Assurance Standards Guidelines for Selection and Use
ISO 9001	Quality Assurance in Design or Development, Production, Installation and Servicing
ISO 9002	Quality Assurance in Production and Installation
ISO 9003	Quality Assurance in Final Inspection and Testing
ISO 9004	Quality Management and Quality System Elements (Aids in the development of internal quality systems that comply with ISO 9000)

Fig. 4. Summary of ISO 9000 standards

standard includes all three phases, the 9002 standard covers production and testing of the final product, and the 9003 standard is aimed only at the final product. The 9004 standard is intended to serve primarily as a guidance document as it describes 20 aspects of a quality programme that are audited during the certification process. The US representative to the ISO is American National Standards Institute (ANSI), a fact why, some (Elliot, 1993; Egan 1993) may say, the ANSI/IEEE standards are compatible with these ISO standards.

ISO 9001 is the most defined of these standards. ISO 9001 emphasizes the implementation of measurement and metrics-such as the use of statistical process control (SPC), to manage the development and delivery process. SPC is utilized to collect data and report metric values on a regular basis, to identify the performance on each level, to take action if metrics levels exceed established target levels, and to establish specific improvement goals in terms of the metrics (Dawood, 1994).

The unique quality of ISO 9000 is the ability to define standards for producing a quality software process. By definition, an ISO 9000 system must be documented, controlled, auditable, monitored, improved and effective. It is designed to create consistency and comprehensibility by requiring documentation in words and pictures (Dawood, 1994). It does not ensure quality but it just reduces the risks of producing poor products. Quality is the goal and management must be committed to quality and empower their workers to continuously improve their development processes.

The following are the seven key characteristics of any ISO 9000 conforming quality system (Schmauch, 1994):

(1) Quality objectives. First and foremost, you must have quality objectives. Your company or organization should have a quality policy that states its quality goals and objectives and the strategy it will use to achieve them.
(2) Commitment, involvement, and attitude. To have an ISO 9000 quality system means that all employees and managers must be committed to the quality objectives and involved in achieving the objectives. Top-level management is responsible for managing quality and must visibly demonstrate its commitment to the quality objectives. All employees must be involved in putting the quality system into practice.
(3) Controlled. Controlled may be the key word in this entire discussion. Every aspect of what is done during the development process must be controlled, or under control.
(4) Effective. Effective may be the second most important word in this discussion. It holds the clue to many of the value judgements made about your development process and your quality systems. This is the means by which you measure whether your quality system is really working for you.
(5) Auditable. The ISO 9000 series of standards requires that systematic internal audits of your quality system be conducted.
(6) Documented quality system. Your quality system, including your processes and procedures, should be documented to the extent that, if you had to replace all of your employees, you could do it and still continue your business.
(7) Continual improvement. The ISO 9000 series of standards requires that your quality system be continually monitored and reviewed for weaknesses and that improvements be identified and implemented.

Control is key to satisfying the ISO 9000 standards. The product parts being developed as well as the processes, procedures, tools, and people that have an effect on the quality of the product must be controlled. ISO 9000 provides a very specific process for quality and has been successfully implemented throughout the world.

Since ISO 9001 confirms process conformance from initial product development through

Section	*Topic*	*Section*	*Topic*
4.1	Management Responsibility	4.2	Quality System
4.3	Contract Review	4.4	Design Control
4.5	Document Control	4.6	Purchasing
4.7	Purchaser-supplied Product	4.8	Product Identification and Tractability
4.9	Process Control	4.10	Inspection and Testing
4.11	Inspection, Measuring, and Test Equipment	4.12	Inspection and Test Status
4.13	Control of Non-conforming Product	4.14	Corrective Action
4.15	Handling, Storage, Packaging, and Delivery	4.16	Quality Records
4.17	Internal Quality Audits	4.18	Training
4.19	Servicing	4.20	Statistical Techniques

Fig. 5. Attributes in the 20 clauses of ISO 9001

production, test, installation, and servicing, it is the quality model most looked at by organizations who are seeking certification. It outlines the minimum attributes, as shown in Fig. 5, a company needs for a generic quality system in two-party contractual situations.

4.1 *The process of achieving certification*

To qualify for ISO 9000 registration, companies must perform internal quality audits, train up to 10% of their workforce in quality assurance and develop a comprehensive quality manual. To be ISO certified, a company must contact a third party certified registrant which then conducts an audit of the company's quality assurance programme to determine compliance with applicable standards. The audit is basically a perusal of the company's documented procedures and inspection via interviews to determine whether the procedures are followed. However, in order to achieve full compliance, all components must be produced by certified manufacturers. This means each development facility and any sub-contractors must be certified. Depending upon the program, the entire certification process may take from a few months to a few years with certification being valid for three years.

ISO certification can only be awarded by a certified registrar. To be a registrar, a company must be accredited and its auditors must be certified. In the USA, only the Registrar Accreditation Board (RAB), founded as a separately incorporated affiliate of the American Society for Quality Control (ASQC), has the authority to accredit. The American National Standards Institute (ANSI), in its Accreditation Program for Quality Registrars, incorporates the RAB accreditation programme. However, some firms have been accredited as registrars by foreign organizations, such as AFAQ (France), NACCB (UK), PVC (Netherlands), and JAS-ANZ (Australia and New Zealand). The International Accreditation Forum (IAF) has been formed to determine the equivalence of accreditation from these boards (Dichter, 1993).

4.2 *ISO software quality assurance*

While ISO 9001 is meant to apply to all products, services, and industries, it was perceived to be primarily prepared for manufacturing processes. Therefore, it did not clearly show how the model

applied to a specific industry, particularly the software industry. Since the process of development and maintenance of software is different from that of hardware, additional guidance was needed for quality systems involving software. To compensate for this weakness and to ensure a uniform interpretation of the model for software, ISO published in 1991, ISO 9000-3 entitled 'Guidelines for the Application of ISO 9001 to the Development, Supply, and Maintenance of Software'.

In general, the ISO 9000-3 guidelines, established to set international guidelines for software quality assurance, describe the suggested controls and methods for producing software that meets the purchaser's requirements set forth in purchaser specifications. Several issues are addressed in the proposal, including internal audits, purchasing specifications, training and standardized software testing and validation procedures.

More specifically, to be ISO 9000-3 compliant, a software developer must have a formal set of company-defined process procedures in place, documenting the use of a formal life-cycle model governing the design, development, and maintenance processes. These formal procedures are referred to in terms of the development 'framework' (in-house quality assurance programmes), 'life-cycle activities' (the overall software development process), and 'supporting activities' (those necessary to qualify, conform, and confirm that the software product was developed properly). According to Inwood (1993), there are four categories of compliance based on combinations of two factors: critical/non-critical factors, and whether the software components were previously developed.

Essentially, the basic requirement of ISO 9001 still holds true for an effective software quality system. Simply put, this is a system to record and track defects, and ensure their removal from the production process. This means the production process must be known and inspection points at specific breaks in the process must be clearly defined.

5. Management's role

In a survey made by the Standish Group (Johnson, 1995), 365 information technology executive managers stated one of the major factors that cause software projects to fail or be challenged, is the lack of management support. Management support was ranked second to user involvement for necessary criteria in the successfulness of a software project. Paulish and Carleton (1994) state that generally, management support is necessary for software-process improvement. In their initial studies of various Siemens software development organizations in both Germany and the USA, the need for management support buy-in at all levels of the organization seemed to be a more important issue in the USA than in Germany. They also observed that organizations that provided generalized quality training (for example, TQM) were generally more supportive of software-process improvement.

Although both the CMM and the ISO standards actually overlap in their recognition of management roles, they both may suffer from criticism of not recognizing enough management roles. Capers Jones (1995) criticized the CMM of covering only one third of the factors that influence software projects until a recent publication of a new set of SEI factors dealing with personnel skills and human relations was released. Some of the listed factors missing could be interpreted as management responsibilities, i.e. staff hiring practices, appraisal methods, salaries and benefits, and office environments. The following paragraphs address management's role in the CMM and ISO 9000 quality standards.

5.1 *CMM management's role*

The intent of level 2 is to concentrate on the establishment of basic management practices to

achieve an effective disciplined software process. Not every project has to be following the same process at this level, but the organization must have an overall policy in place to guide these projects. Level 3 builds on this foundation because the focus is on management acting to improve its processes before tackling technical and organizational issues at level 3. For instance, Integrated Software Management at level 3 is the evolution of Software Project Planning and Software Project Tracking and Oversight at level 2 as the project is managed using a defined software process. The key process areas at level 3 address both project and organizational issues, as the organization establishes an infrastructure that institutionalizes effective software engineering and management processes across all projects.

Level 2 organizations have installed basic software management controls. The project's process is under the effective control of a project management system, following realistic plans based on the performance of previous projects. Project software managers track software costs, schedules, functionality, and identify problems in meeting commitments. The Requirements Management KPA involves understanding the customer's requirements as the basis for planning and managing in the Software Project Planning and Software Project Tracking and Oversight key practices, respectively. Software Project Planning consists of the establishment of plans for the foundation of management. The intent of the Software Project Tracking and Oversight is to provide adequate visibility of the actual progress so that management can take effective actions when performance deviates from the software plans. The establishment of Software Quality Assurance should provide the visibility by reporting the findings of progress, defects, etc. to management. However, compliance issues are to be addressed first within the software project and resolved there, if possible. Otherwise, the SQA group escalates the issue to an appropriate level of management for resolution. Until then, it is not management's responsibility to perform the actual tasks of SQA. These are to be delegated to someone else, one not under the supervision of the project management.

In Subcontract Management, the concerns of all the aforementioned key process areas are combined in the selection of qualified subcontractors and managing them effectively. The subcontractor performs the software planning, tracking and oversight activities for the subcontracted work, but the prime contractor ensures these activities are performed appropriately. Additionally, the prime contractor ensures the delivered software products satisfy the documented acceptance criteria. Both contractors work together to merge their product and process interfaces.

The CMM specifically differentiates between tracking and taking corrective actions versus managing. In the Software Project Tracking and Oversight KPA at level 2, many of the key practices use the phrase '... is tracked ... corrective actions are taken as appropriate'. In the level 3 KPA of Integrated Software Management, many of the similar key practices use the phrase 'is managed'. This is due to a complete defined software process at level 3 where the relationships between the various products, tasks and activities are well defined. Whereas management at level 2 is likely to be reactive to problems, level 3 management is better able to anticipate problems and proactively prevent their occurrence.

At level 3, all processes, including both software engineering and management processes, are documented and integrated into a coherent whole. Basic management involvement lies within an organizational focus and a project focus. The three KPAs, Organization Process Focus, Organization Process Definition, and Training Program, all involve senior management. In the Organization Process Focus the organization established the responsibility for all software process activities. The Organization Process Definition develops and maintains a useable set of software process assets that improve process performance across the projects and provides a basis for cumulative, long-term benefits to the organization. These assets provide a stable foundation that can be institutionalized via mechanisms, i.e. the Training Programme. The Training Programme's purpose is

to develop the skills and knowledge of all individuals so that roles are performed effectively and efficiently.

The Integrated Software Management KPA defines the integrated software process tailored from the organization's standard software process and related process assets described in the Organization Process Definition KPA. The Intergroup Coordination KPA is the interdisciplinary aspect of the Integrated Software Management KPA where software engineering groups' interactions with other groups is coordinated and controlled.

The CMM specifies the level of management involvement in all its activities and key practices. To aid in the distinction of these levels of management, the CMM also provides definitions of the various levels of management so that its intended audience is not misinterpreted. In general, senior management is considered a role at a relatively high level in an organization with the primary focus being long-term vitality of the organization, rather than the short-term project and contractual concerns and pressures. Primary functions include providing and protecting the resources necessary for long-term improvement of the software process. Typically the CMM's Commitment to Perform feature looks for senior management roles in:

- demonstration of commitment to software process activities (especially reviews);
- establishing long-term plans and commitments for funding, staffing, other resources, and process development and improvement;
- ensuring standard software process supports business goals and strategies.

The Verifying Implementation common feature generally contains key practices that relate to oversight by senior management and project management, as well as specific SQA activities for verifying key practices are being performed properly. Oversight by management is performed via periodic reviews to proved awareness of, and insight into software process activities at the appropriate level of abstraction and in a timely manner. At a project level, management oversight is expected to be at a more detailed level, reflecting the role of more active involvement in the operational aspects of a project. Examples include participation in formal reviews, such as critical design reviews, as well as reviews which encompass process issues such as status of process improvement planning and resolution of noncompliance issues.

5.2 *ISO management's role*

ISO 9001 states that a quality policy shall be defined and documented along with objectives and commitment to quality. Along with the policy, it is management's responsibility to ensure the policy is understood and maintained at all levels in the organization. This is accomplished by defining the responsibility and authority for personnel who manage, perform, and verify work affecting quality and identifying and providing verification resources. A designated manager ensures that the quality programme is implemented and maintained. Management, per clause 5.5.1, must periodically review the quality system to ensure its continuing effectiveness.

Clause 5.4.1 of ISO 9000-3 calls for a development plan for a project. Contained within it is the project's organization describing the team's structure and responsibilities, including management's roles. A section calls for how the project will be managed, how progress will be monitored and measured, organizational responsibilities, work assignments, and how interfacing will take place between different groups. The plan is also to show project schedules, tasks to be performed, by whom, when, and the resources that will be applied to each task during the project. All of this could be considered to fall within the realm of a manager's role.

ISO 9000-3, exceeds the CMM's management role in that it not only reflects the supplier's

management responsibility (organization's) as outlined in ISO 9001 but also outlines management roles for the purchaser. The purchaser is required to cooperate with the supplier to provide, in a timely manner, all necessary information for defining requirements, answering questions, approving proposals, defining acceptance criteria and procedures, resolving pending items, and holding joint reviews with the supplier. The standard recommends the purchaser assign a representative the responsibility for dealing with most of these listed items as contractual matters along with concluding agreements, ensuring their organization observes agreements made with the supplier and dealing with the unsuitability of software items supplied. The joint reviews, to be documented, are for covering aspects of conformance to requirements specification, verification results, and acceptance test results.

6. Measurements

One of the quality concepts the ISO 9000 standards address is that an organization shall provide confidence to its own management that the intended quality is being achieved and sustained. This type of reporting typically implies measures but, as Paulk (1995) states, 'ISO-9001 is somewhat ambiguous about the role of measurement in the quality management system'. This is based on the fact that the quality policy described in Software Quality Management at level 4 of the CMM is quantitative versus ISO 9001's clause 4.20, statistical techniques, which requires the quality objectives to be defined and documented, but not necessarily be quantified. On the other hand, this does not imply that the CMM addresses measurement only when an organization has attained level 4. Rather, the CMM requires measurement and analysis as a common feature on all levels for process measurement whereas product measurement is typically incorporated into the various practices within the Activities Performed common feature. Additionally, level 3 describes the establishment of an organization-wide process database for collecting process and product data.

This section explores the first documented recommended measurements which Baumert and McWhinney (1992) released in the form of indicators compatible with measurement practices of the CMM. Discussion then returns to the measurement requirements described in the ISO-9000 standards.

6.1 *CMM's initial core of measurements*

The Software Engineering Institute was tasked by the Defense Advanced Research Projects Agency (DARPA) to develop definition frameworks for a core set of software measures to be implemented with all DoD systems as one of the 17 initiatives outlined in the DoD's Software Action Plan (SWAP). The SWAP outlined the DoD's objectives of reducing equivalent software life-cycle costs by a factor of 2, reducing software problem rates by a factor of 10, and achieving new levels of DoD mission capability and interoperability via software. Thus, SEI's response in 1992 was to provide a standard method for measuring and reporting software products and processes so comparisons could be made across domains and the nation. The core set of measures, recommended for initial measurement, focused on management measures of size, effort, schedule, and quality. These measures are considered crucial to managing project commitments not only for project tracking and reporting, but also for planning and improving future products via process improvement (Carleton *et al.*, 1992).

The SEI stresses establishing a cyclic process for software measurement within an organization by identifying the current situation, primary goals, and primitive measures; defining how the

Units of measure	*Characteristics addressed*
Counts of source statements	Size, progress, reuse
Counts of staff hours	Effort, cost, resource allocation
Calendar dates	Schedule
Counts of software problems, failures, and faults	Quality, readiness for delivery, improvement trends

Fig. 6. CMM levels and characteristics

identified measures are to be collected and used; collecting measurement data; evaluating the effectiveness of the efforts at each stage in addressing the organization's goals; and repeating the cycle for process improvmeent. The SEI suggests making use of Basili's Goal/Question/Metric (GQM) paradigm for the identification stage and states the cyclic process for establishing software measurement is akin to Shewhart and Deming's quality methods of Plan/Do/Check/Act.

The focus of measurements for each CMM level are as follows:

(1) Initial establishing baselines for planning and estimating.
(2) Repeatable project tracking and control.
(3) Defined definition and quantification of intermediate products and processes.
(4) Managed definition, quantification, and control of subprocesses and elements whereby characteristics of the development process itself are captured to allow control of individual activities of the process
(5) Optimizing dynamic optimization and improvement across projects where measurement provides feedback from mature processes for dynamically changing processes across multiple projects

The SEI's core set of measurements, are described in three documents (Park, 1992; Goethert *et al.*, 1992; and Florac, 1992). The central theme is the use of checklists to create and record structural descriptions of measurement specifications and measurement results. The framework documents include example definitions; checklists for constructing alternative definitions and data specifications; instructions for using the checklists to collect, record, and report measurement data; and examples of how the results can be used to improve the planning and management of software projects. The recommended measurements are defined with structured rules that state explicitly what is included and excluded in each measure. Each measure provides for collecting data on multiple attributes. SEI admits these are not the only measures that can be used but chose four basic measures shown in Fig. 6 that address important product and process characteristics vital for planning, tracking and process improvement.

6.1.1 *CMM's measurement of size*

Size measures have a direct application to planning, tracking, estimating in their use of computing productivities, normalizing quality indicators and deriving memory utilization and test coverage. The SEI's core recommended size measure is physical source lines of code (SLOC) reported as the count of lines of code (drawn heavily from the draft 1992 IEEE Standard for Software Productivity Metrics). Subsequently, counts of logical source statements or even other size measures may be added, but only after the requisite language-specific rules have been defined and automated counters have been built. The SEI does not maintain the belief that one size measure is more useful or

more informative than another. Rather, judgement should sometimes be made as to which measures will be easiest to implement and which will be easiest to automate. The recommendations do not discourage current use of any measure being used to good effect, nor suggest SLOC is the best measure of size. The recommendation is based solely on simplicity and the fact that historically, cost models used for project estimating are based on physical measures of source code size. If other measurements were to be recommended, the SEI's order of preference would be SLOC, counts of software units, logical source statements, and, where appropriate, function points.

Park (1992) provides guidelines for defining, recording, and reporting physical source lines and logical source statements. For physical source lines, the definitions include recording attributes of programming language, statement type, i.e. noncomment/nonblank source statements, development status, origin, and production method. The guidelines do not address measures of difficulty, complexity, or other product and environmental characteristics often used when interpreting measures of size. The emphasis is that consistent definitions of software size measures be maintained to satisfy the data needs to different users.

Counts of logical functions or computer software units (CSUs) are considered to be useful measures but the SEI has not been able to find work helpful in the contstruction and communication of formal definitions for these measures. Logical source statement counts (or instruction counts) are often attempts for measuring size of software independent of the physical form in which instructions appear. The theory is that the number of logical instructions is proportional to the amount of logic in the code or the effort and cost are proportional to the total logic that must be specified, designed, coded and tested. The advantage of this measure is that is less dependent on programming style than counts of physical source statements and unaffected by the passing of code through a source code formatter or pretty printer. The disadvantage is that interpretations are strongly dependent on rules for deciding what constitutes a statement for identifying statement delimiters as these vary from language to language. SEI's conclusion is that physical source lines are both simpler to count and less subject to unstated assumptions than logical source statements as detailed lists of language-specific counting rules must be identified and recorded for each programming language before use.

Capers Jones (1995) criticizes CMM's measurements based on LOC, stating they are obsolete and were proven not to work years ago. His belief is that less than a quarter of software work is coding, so LOC metrics tend to conceal the high costs associated with paper document production including plans and specifications. Capers also states that SEI's CMM does not provide rules or guidance for converting between physical lines of code and logical statements or address conversion of LOC data into equivalent function point data (an available technological advance called 'backfiring').

SEI recognizes the advantages of function point measures over source statement counts because of their language independence, often solution independence, and their early computability in development life cycles. However, three properties cause hesitation for recommending implementation in every organization:

(1) Nonsupport of project-tracking. Function points are not contained in units so it is difficult to track the percentage of total function point count coded, tested, integrated, etc.
(2) Focus is not on employed design. Function point and feature point counts are oriented toward the customer's view of what the software does rather than on the producer's view of how it is done. Therefore, the focus is on the value received.
(3) Non-applicability to all kinds of software (i.e. embedded, real-time systems, continuously operating systems, or systems spending large amouns of computational effort to generate a small number of numerical results).

6.1.2 *CMM's measurement of effort and schedule*

The recommended core measurement for effort is staff hours as used by Institute of Electrical and Electronics Engineers, Inc (IEEE). This is a reliable measure for software cost and is classified by labour class and the type of work performed, i.e. requirements analysis, design, coding, unit testing, integration, quality assurance, and configuration management. Staff hours was selected versus labour months as there is no standard for the number of hours in a month. A staff week has similar problems in that organizations may vary in the length of a standard working day. Each of these counts are either not granule enough or present other problems such as accounting for overtime and holidays that occur during the period. The full definition framework is described in Goethert *et al.* (1992) and is tailorable to meet specific organizational needs.

In Paulish and Carleton's (1994) initial data collection of Siemens, effort is assessed using the organization's existing accounting and time-reporting procedures. It includes the work performed by all staff involved with software development, including test, quality assurance, and documentation generation. Effort is expected to be counted for the duration of the phases in product development from the beginning of high-level design to one year after first customer delivery.

SEI recommended that DoD projects adopt structured methods for defining schedule. Their recommendation is twofold. First, track both the planned and actual calendar dates of milestones, reviews, audits, and deliverables. Second, track the exit or completion criteria associated with each date. These dates are recommended versus dates of project activities or phases i.e. design, analysis, etc. due to the ambiguity of when these activities start or stop. This recognizes the fact that most activities are continued to some extent throughout the project. Schedule adherence at Siemens (Paulish and Carleton, 1994) is being calculated as a percentage of the difference between the estimated schedule duration time and the actual schedule duration time, divided by the schedule duration time. A negative number indicates a schedule slip; a positive number indicates that the development was done more quickly than estimated.

6.1.3 *CMM's measurement of quality*

Counts of software problems and defects are recommended for aiding planning and tracking of software development and system support. SEI recommends using these to determine when products are ready for delivery to customers and for providing fundamental data for process and product improvement (Carleton *et al.*, 1992). However, it is not recommended to attempt to use the definition framework described in Florac's (1992) work above the organizational level as the definitions are process dependent. A software defect is defined as any flaw or imperfection in a software work product or process. The counts of problems and defects are classified by status, type, severity, and priority.

The number of defects found per phase and the total are being collected in Paulish and Carleton's (1994) Siemens study as performance measures by determining the defect detection distribution, defect rate, and the defect insertion rate. The defect detection distribution is the percentage of faults found in each development phase. Defect rate is the number of defects for each development phase divided by the product size in thousands of lines of code (KLOC). The defect insertion rate is the sum of the defect rates over all the phases.

6.1.4 *Other CMM measurements*

As stated previously, the SEI recognizes other metrics. Paulish and Carleton (1994) include development cost, which can be calculated as the effort multiplied by the average hourly labour

rate for the project. They also report profitability and customer satisfaction are sometimes used to measure organization performance. The SEI's technical report on size measurement (Park, 1992) lists, in its appendix B, observable inputs and outputs that might be used to quantify size of activities representative of those found in references like DoD-STD-2167A and IEEE Std 610.12-1990. This list was compiled during the SEI's initial work in defining a size measurement. It includes major life-cycle and testing phases, peer reviews, documentation, quality assurance, configuration management, and CCB meetings associated with production and supporting software systems.

Concepts of software measurement for acquisition programme managers are reported in the SEI's Software Acquisition Metrics Working Group's work (1992). The basic measurements discussed are software size, effort, staff, milestone performance, development progress, software defects, and computer resource utilization. Included are methods for analyzing the data once collected, i.e. trend analysis, multiple metric relationship analysis, modelling input data analysis, and analysis of thresholds and warning limits.

Baumert and McWhinney (1992) describe additional measures compatible with the CMM in the form of software indicators, covering 13 different categories that include progress, effort, cost, and quality. The indicators are presented with the following format:

- Objective
- Listing of indicators within an indicator category
- Key process area goals addressed
- Life-cycle stage where indicator should be used
- Users of the indicators
- User's questions answered with use of an indicator
- Input needed to derive the indicators
- Interpretation of the indicators
- Sources of reference material(s) for further description or explanation

The categories of indicators do not include risk management because of its immaturity in practice but include progress, effort, cost, five categories of quality, three categories of stability, computer resource utilization, and training.

6.1.5 *Observations of CMM's measurements*

Capers Jones' criticism of the SEI assessment approach is that the SEI has not included quantitative benchmarks of either productivity or quality data nor does the institution show average results or performance ranges of the five maturity levels. He believes that large-scale studies, with statistically valid sample sizes, are required to collect empirical data on quality levels, productivity levels, and the investments associated with each level (Jones, 1995). The primary date collected at the various Siemens software development organizations in Paulish and Carleton's (1994) work is similar to the SEI-recommended core measurements, but may not be considered to be empirical in nature. However, this study seems to address some of Jones' concerns of missing environmental data in the SEI's work on measurement. The study is tracking staff size and turnover, maturity level, and staff morale. Also worthy to note is the fact that half the case study sites obtained ISO 9001 certification during the first year of the study.

Jones also states SEI's work is far behind the best civilian practices as it tends to reflect military topics of interest. With the government having the largest bases of data, he believes it would be highly useful for the SEI and the government to explore the topic of data metrics. No effective metrics exist for measuring the quality of data used by applications, the costs of creating the data,

and the costs of repairing the data when it is incorrect. He also encourages others as well as the SEI to explore activity-based costing and the sizing, estimating, and measuring of hybrid systems including 'hardware components, software components, electronic components, and microcode components'.

6.2 *ISO measurements*

ISO 9001 states that organizations must identify adequate statistical techniques and use them to verify the acceptability of the process capability and product characteristics. ISO 9000-3 simply characterizes this clause as measurement (6.4). But to quote ISO 9000-3, 'there are currently no universally accepted measures of software quality'. However, it does state that at a minimum, product measurement should report field failures and/or defects from the customer's viewpoint and that the selected metrics should be described such that results are comparable. ISO 9000-3 states quantitative measures of product quality are collected for the purposes of:

- collecting and reporting on a regular basis;
- identifying the current level of performance on each metric;
- taking remedial action if metric levels grow worse or exceed established target levels;
- establishing specific improvement goals in terms of the metrics.

Quantitative measure of process quality are to reflect:

- how well development process is being conducted in terms of milestones and quality objectives being met on schedule;
- effectiveness of development process at reducing the probability of introduction of faults or that any faults introduced go undetected.

The important feature to note in the ISO 9000 standards is that any statistical techniques used must be proven correct and accurate. In addition to validating the metrics, measurements, etc., Schmauch (1994) states one must be able to show validity of any predictive algorithms used (e.g. for predicting number of remaining problems) and methods used to collect data.

7. Impacts of the Two Models

It is difficult to know the exact impacts of these two models, perhaps because of the number of documented cases available. In his tracking the field of measurement for 20 years, Rubin (1995) has been able to find evidence of almost 600 organizations that have attempted to implement measurement programmes but has only seen about 80 documented successes. As with many ISO standards, 9000-3 has gained more momentum in Europe than North America. By late 1992, 530 US companies and about 25 000 European companies were registered with ISO (Dichter, 1993). ISO 9000 has been implemented by more than 1800 European companies and over 90 countries. Most of these ISO-registered companies now buy only from ISO 9000-certified firms. Additionally, NATO now requires ISO certification for its contractors (Dichter, 1993). CMM has more documented case studies than ISO (albeit they may be prejudiced), but perhaps elaboration on the number of companies/countries ISO registered/certified or reported to be following CMM (although not strictly empirical analysis) may show some indicative trends.

With respect to the CMM, industry has reacted with both favourable and unfavourable opinions (Saiedian and Kuzara, 1995). A number of studies interested in the impact of the CMM purport to

demonstrate software productivity improvement and cost savings. The most detailed one is related to the assessment, actions, and re-assessment at Hughes Aircraft Software Engineering Division (Humphrey *et al.*, 1991). According to this study, the assessment and re-assessment cost near $550 000 but Hughes estimates the resulting annual savings to be $2 000 000. Other studies question CMM's effectiveness and doubt if there is any meaningful correlation between its rating and the actual ability of an organization to produce quality software on time and within budget (Bollinger and McGowan, 1991). Nevertheless, with strong DoD sponsorship, more US companies will probably base their software improvement efforts on CMM.

CMM collected and analysed data from 13 early adopting organizations engaged in CMM-based software process improvement (Herbsleb *et al.*, 1994). The diverse group of participants included DoD contractors, commercial organizations, and military organizations representing the whole range of maturity levels and various application domains. SEI requested three categories of information: descriptive information about the organizations, information about the process improvement and measurement programs, and data about the results of the improvement efforts. The study is not considered to be without limitations, but hope is that it identified needs for future samplings so that all be addressed more systematically.

The results of the study are summarized in Table 1. Additionally, six companies describe their software process improvement in separate case studies in the back of the report. Interestingly, some of these companies were not simply attempting to achieve higher maturity levels based on the CMM, but were also either ISO 9000 certified or attempting to become certified.

According to Schmauch (1994), fulfilling the requirements of ISO 9000 can be costly in time, effort, and money. The cost primarily depends on how close a current quality system conforms to the ISO 9000 standards. Making significant changes to an existing process can be disruptive and simple documentation of existing procedures of a currently conforming process can be expensive and time-consuming. To register, an organization must be able to demonstrate a working (i.e. mature) ISO 9000 conforming quality system. To do this, he states will take a minimum of three months of actually using the quality system and keeping records prior to a third-party registration audit. Unless the quality system is already close to conforming and ready for registration assessment, one can expect ISO 9000 registration to take at least one year to 18 months. If one is starting with no system or a poorly documented system, it could take 18 to 24 months or more.

Despite the fact many organizations have implemented the ISO standards and many others are considering ISO certification, the authors have been unable to locate any quantitative data demonstrating the payoff of adhering to its standards. According to Caldiera (1995), an expert on ISO, '... the main reason is that many companies that initiate an ISO 9000 certification effort do not have current cost of quality figures. Therefore [it] is impossible to calculate a cost/savings ratio'. ISO 9000's www home page (www.iso.ch/welcome.html), which provides a rich and comprehensive

Table 1. Summary of the overall results (Herbsleb *et al.*, 1994)

Category	*Range*	*Median*
Total yearly cost of SPI activities	$49 000–$1 202 000	$245 000
Years engaged in SPI	1–9	3.5
Cost of SPI per software engineer	$490–$2004	$1375
Productivity gain per year	9%–67%	35%
Early detection gain per year (defects discovered pre-test)	6%–25%	22%
Yearly reduction in time to market	15%–23%	19%
Yearly reduction in post-release defect reports	10%–94%	39%
Return-on-investment in SPI	4.0–8.8	5.0

hypertext knowledge on its standards, does not provide a pointer to any study showing ROI or 'payoff' figures.

8. Observations

According to Dichter (1993), the ISO and SEI criteria have many areas in common. For ISO certification, it is stated all SEI level 2 and, some of the SEI level 3 Key Process Areas are required to be under control. A software organization hoping to become ISO compliant therefore needs to be close to level 3 in maturity. Furthermore, the recommendation includes criteria for the Malcolm Baldrige National Quality Award when an organization approaches the higher levels (3 to 5) because the Baldrige criteria are reportedly similar to SEI levels 4 and 5. Likewise, Egan (1993) also states that conforming suppliers of the Dod-Std-2167A/2168, for example, could 'possibly' easily meet the ISO 9000-3 requirements.

In a more detailed analysis, Bamford and Deibler (1993) concluded 'ISO 9001 and the SEI model are compatible where they overlap'. The agreed on number of critical success factors addressed by the two models were; the independent audits of development activities, corrective action, employee training and, detailed process and life cycle definition. Whereas the SEI questionnaire addressed the implementation of statistical methods, definition of standards, and implementation of technology in significantly more detail than ISO 9001; the analysis indicated a number of areas critical to ISO 9001 compliance missing. These areas included the following with ISO referencing (Dichter, 1993; p. 70):

- acceptance criteria for each phase of the design and development process (4.4.4);
- document control (4.5);
- records (4.16);
- assignment of responsibility (4.1.2) and competent personnel (4.18);
- packaging, handling, storage, delivery (4.15), installation (4.9.1), and servicing (4.19); and
- responsibility for portions of the product that are sub-contracted (4.6.2), purchased from another company (4.6.3, 4.6.4), or supplied by the original purchaser (4.7).

In conclusion of these analyses, although one organization of an advanced maturity level might be ISO 9000 compliant relatively easily, another organization might not. Although evidence, as previously explored, indicates movements to bring the models together, the DoD may need to reconsider its position as advocacy to replace the IEEE and DoD standards by ISO 9000-3 has been seen (Egan, 1993).

The most notable observations of the past year indicate a movement within the DoD of using both the CMM and ISO 9000 quality standards. Some of these observations are presented in the following paragraphs.

In February 1994, the Air Force Acquisition issued a Software metrics policy, Policy 93M-017, whereby the collection, analysis, and use of software metrics covering cost, schedule, and quality were mandated for software systems greater than 20 000 source lines of code. The 'core' attributes of size, effort, schedule, software quality, and rework were mandated, but program managers were given the flexibility to determine 'how' they were to be measured. In other words, the policy did not prescribe the specific metrics to be used, but provided a 'handbook' with guidelines describing the attributes of these metrics. The Guidelines included a bibliography and references to the Software Engineering Institute's framework documents, Army Software Test and Evaluation Panel Metrics, and Air Force Pamphlet 800-48 'Software Management Indicators', for descriptions of additional

Size:	Source lines of code Function points
Effort:	Number of staff hours expended monthly
Schedule:	WBS activity start/end dates DoD-STD-2167A milestone start/end dates Key interim product milestone start/end dates Progress to date Estimated slip
Software quality:	Total number of errors opened/closed Number of errors opened/closed since last report Type of error (testing, action item, document comment) Classification and priority Product in which error was found
Rework:	Number of open Software Change Orders (SCOs) Number of closed SCOs Total number of SCOs

Fig. 7. Elements within mandated attributes

useful metrics. The data elements identified within each of the mandated attributes are shown in Figure 7.[1]

These core measures, except for rework, were as described by SEI's 1992 frameworks. The policy stated that within these frameworks, 'measurement should be used as a problem identification and resolution tool targeted toward making the software processes and products better'. Recognizing that no single set of measures can ever encompass all that the Air Force needs to know about software products and processes, managers are strongly encouraged to use other measures besides the core that they believe will be of value to themselves and their programme. The other measures encouraged were described in a table of categories, metric names, and measurements as shown in Table 2.

With regards to the new metrics policy, the Deputy Assistant Secretary of the Air Force stated 'programs should also measure requirements traceability, requirements stability, design stability, complexity, breadth of testing, depth of testing, and reliability for a true picture of their process and product. On the other hand, the minimum, basic metrics now mandated will go a long way toward assuring that software-intensive acquisitions stay on track to successful and predictable fulfillment ... In summary, the Air Force is committed to developing software on predictable schedules, at predictable cost, with predictable performance and reliability' (Mosemann, 1994).

At the same time, on 14 February 1994, the US Undersecretary of Defense John M. Deutch signed the policy on the use of commercial quality system standards in the DoD and authorized the use of ANSI/ASQC Q90 and the ISO 9000 series standards in contracts for new programs. Later, on June 29, 1994, Secretary of Defense William Perry signed a memorandum approving a plan to reduce the reliance on military specifications, including the US DoD-STDs 2167, 2167A, and 2168 (Dawood and Egan, 1994).

[1] The rework attribute measures in SLOC the amount of software damaged and repaired for each type of SCO collected.

Table 2. Examples of additional measures

Category	*Name of metric*	*Measurement*
Management metrics	Computer Resource	Resource capacity utilized
Requirements metrics	Requirements Traceability	% Requirements traced to design code test
	Requirements Stability	Requirements changes and effects
Quality metrics	Design Stability	Design changes completed
	Complexity	Quality of code
	Breath of Testing	% Function requirements shown
	Depth of testing	Degree of code testing
	Reliability	Mission failure due to software

This suggests that the DoD sees the benefits of ISO 9000 standards, particularly those regarding software, ISO 9000-3. MIL-STD-2167 is the standard many DoD contractors have had to deliver in accordance with whereas MIL-STD-2168 is the DoD's quality assurance standard. Contractors complying with these standards should have little problem meeting ISO 9000-3 certification by adding quality planning and corrective actions to their procedures, provided they have already documented their processes.

As for the DoD's goals, as outlined in the SWAP, it would seem they are realistic if Motorola's exemplification is accurate. Motorola attended the 1994 Software Engineering Process Group national meeting which had a presentation track and one tutorial devoted to ISO 9000 and the CMM. Here, Motorola reported that process-improvement initiatives based on the SEI's CMM have reduced development-cycle time by 20%, increased productivity by a factor of two to four, and resulted in some products whose quality is in the range of two to three errors per million lines of code. However, it was noted that the performance of individual organizations may differ by a factor of up to 30 (Card, 1994).

9. Conclusions

Choosing the right process model depends on many factors. These factors include items such as the organization size, the finance available, the type of business, and the customers of the company. If a company contracts out much of its own work the SEI CMM model may be a good choice as it looks at the issue of sub-contract control in great detail.

For new companies just starting out, ISO 9000 would be the best choice. It helps pick and choose beginning metrics and sets up quality control standards. ISO 9000 also indicates what to look for when setting up a new company. Moreover, many companies have already successfully applied the ISO 9000 standards to their business and have many lessons learned to share. The SEI CMM would not be suited for fledgling companies. It is more of a snapshot of the company at a given time.

Any company which deals with US government contracts should consider the SEI CMM model. The US government has a large amount of money invested in developing this model. They want to make sure any company that works for them falls into a mid to higher level on the SEI CMM scale. Although the model may not fit the situation in other ways, the US government wants to ensure that a developer organization what they consider 'acceptable quality standards' when that organization is making their software. Furthermore, the US DOD has mandated level 3 compliance for all DOD contractors by 1998.

Companies short on finance may not be able to afford all of the up front cost of an SEI CMM

evaluation. If they try to stay with ISO 9000 standards, they will probably gain much benefit at lower costs. MIS organizations can probably benefit from either models. According to Mark Dawood, 'ISO 9000 is here now and it works. Whether an organization is a defense software developer or a DOD purchaser of software, its time to make ISO 9000 a central part of its software activity. ISO 9000 will create a win-win situation for all.'

Both of the models are based on the same basic principles of quality and measurement. The key is to find which fits an organization the best. The ISO 9000 series of documentation has also been well used and can adapt to a wide variety of organizations. The CMM is a descriptive process improvement model that characterizes an organization at a particular maturity level. The CMM was written exlusively for software developers and rarely involves the work of other process groups (Dawood, 1994). On the other hand, ISO 9000 is a prescriptive model that outlines how to develop and implement a software quality system. Both the CMM and ISO 9000 focus on quality products.

The SEI's CMM is a major effort at improving the overall excellence of current software engineering practices. This initiative includes enforced compliance as the means to assure its adoption. Arguments have been presented which point out perceived flaws in the CMM and its methods of implementation. However, these arguments, for the most part, do not imply that the CMM is more bad than good, nor do they imply that achieving higher levels of CMM compliance will lead to worse software. On the contrary, it appears generally accepted that most of the CMM practices are beneficial and their implementation will improve software engineering. Since no approach that enforces changes and improvements in software engineering will be universally acceptable in all aspects to all concerned, the CMM, on balance, can be considered to be a very successful model, particularly when combined with TQM principles.

What may be less certain is whether the costs of attaining and maintaining a CMM level will be recouped through reduced software production costs and more efficient software engineering practices. Published studies, some cited in this article, appear to imply that process improvement based on the CMM more than pays for itself. However, it can be safely assumed that studies reporting a positive outcome will be far more numerous than those reporting a negative outcome. Few companies will be willing to publish their failures at process improvement.

Though positive and negative is relative to the beholder, it does appear that certain outcomes are more generally acceptable than others. The cited studies of CMM costs were not about randomly selected companies that have recently improved their CMM level rating. These CMM experiences were purposefully selected for publishing in order to relay some interesting information. No attempt has been made to, for example, study fifty randomly selected companies or projects and to statistically report the results. In fact, it is probably too early in the evolution of CMM to perform this type of study. Hence, time is yet needed to fully analyse CMM and its impact. Nevertheless, with continued strong sponsorship by the DoD, there is likely to be an increasing number of companies that base their software process improvement efforts on the CMM.

Acknowledgement

Hossein Saiedian's work was partially supported by a UCR grant from the University Committee on Research, University of Nebraska at Omaha.

References

Bamford, R.C. and Deibler II, W.J. (1993) Comparing, contrasing ISO 9001 and the SEI capability model. *Computer* **26**(10), 68–70.

Baumert, J.H. and McWhinney, M.S. (1992) Software measures and the capability maturity model. Software Engineering Institute, CMU/SEI-92-TR-25, Carnegie Mellon University, Pittsburgh, PA 15213.

Bollinger, T.B. and McGowan, C. (1991) A critical look at software capability evaluations. *IEEE Software* **8**(4), 25–41.

Caldiera, G. (1995) personal communication.

Card, D. (1993) BOOTSTRAP: Europe's assessment method. *IEEE Software* **10**(5), 93–95.

Card, D. (1994) Making the business case for process improvement. *IEEE Software* **11**(7), 115–116.

Carleton, A.D., Park, R.E. and Goethert, W.B. (1992) Software measurement for DoD systems: recommendations for initial core measures. Software Engineering Institute, CMU/SEI-92-TR-19, Carnegie Mellon University, Pittsburgh, PA 15213, September.

Dawood, M. and Egan, L.G. (1994) ISO 9000 in the Department of Defense. *CrossTalk*, November, pp. 28–30.

Dawood, M. (1994) It's time for ISO 9000 *CrossTalk*, March, pp. 26–28.

Dichter, C. (1993) How good really? Software audits. *Unix Review* **11**(10), 43–49.

Egan Jr., L.G. (1993) ISO 9000-3: Key to quality software and global success. *I&CS*, January, 63–65.

Elliot, S. (1993) Management of quality in computing systems education: ISO 9000 series quality standards applied. *Journal of Systems Management* **44**(9), 6–13.

Florac, W.A. (1992) Software quality measurement: a framework for counting problems and defects. Software Engineering Institute, CMU/SEI-92-TR-22, Carnegie Mellon University, Pittsburgh, PA 15213, September.

Goethert, W.B., Bailey, E.K. and Busby, M.B. (1992) Software effort and schedule measurement: a framework for counting staff-hours and reporting schedule information. Software Engineering Institute, CMU/SEI-92-TR-21, Carnegie Mellon University, Pittsburgh, PA 15213, September.

Herbsleb, J., Carleton, A., Rozum, J., Siegal, J. and Zubrow, D. (1994) Benefits of CMM-based software process improvement: initial results. Software Engineering Institute, CMU/SEI-94-TR-13, Carnegie Mellon University, Pittsburgh, PA 15213, August.

Humphrey, W.S. and Sweet, W.L. (1987) A method for assessing the software engineering capability of contractors. Software Engineering Institute, CMU/SEI-87-TR-23, (Preliminary Version), Carnegie Mellon University, Pittsburgh, PA 15213.

Humphrey, W.S., Synder, T.R. and Willis, R.R. (1991) Software process improvement at Hughes Aircraft. *IEEE Software* **8**(4), 11–23.

Humphrey, W.S. (1992) Introduction to software process improvement. Software Engineering Institute, CMU/SEI-92-TR-7, Carnegie Mellon University, Pittsburgh, PA 15213.

Inwood, C. (1993) Developers still lagging in ISO preparation. *Computing Canada* **19**(17), 19.

Inwood, C. (1994) Standards may solve user frustration. *Computing Canada* **20**(2), 19.

Johnson, J. (1995) Chaos: the dollar drain of IT project failures. *Application Development Trends*, **2**(1), 41–47.

Jones, C. (1995) Gaps in SEI programs. *Software Development* **3**(3), 41–48.

Kan, S.H., Basili, V.R. and Shapiro, L.N. (1994) Software quality: an overview from the perspective of total quality management. *IBM Systems Journal* **33**(1), 4–19.

Mosemann II, L.K. (1994) Why the new metrics policy?. *CrossTalk*, April, p. 3.

Park, R.E. (1992) Software size measurement: a framework for counting source statements. Software Engineering Institute, CMU/SEI-92-TR-20, Carnegie Mellon University, Pittsburgh, PA 15213, September.

Paulish, D.J. and Carleton, A.D. (1994) Case studies of software process improvement measurement. *Computer* **27**(9), 50–57.

Paulk, M.C., Curtis, B., Chrissis, M.B. and Weber, C.V. (1993a) Capability maturity model, version 1.1. *IEEE Software*, **10**(7), 19–27.

Paulk, M.C., Weber C.V., Garcia, S.M., Chrissis, M.B. and Bush, M. (1993b) Key practices of the capability

maturity model, version 1.1. Software Engineering Institute, CMU/SEI-93-TR-025, Carnegie Mellon University, Pittsburgh, PA 15213.

Paulk, M.C. (1995) How ISO 9001 compares with the CMM. *IEEE Software* **12**(1), 74–82.

Rubin, H.A. (1995) Measurement: despite its promise, successful programs are rare. *Application Development Trends* **2**(1), 21–24.

Saiedian, H. and Kuzara, D. (1995) SEI capability maturity model's impact on contractors. *IEEE Computer* **28**(1), 16–26.

Schmauch, C. (1992) *ISO 9000 for Software Developers*. ASQC Quality Press.

Software Acquisition Metrics Working Group (1992) Software measurement concepts for acquisition program managers. Software Engineering Institute, CMU/SEI-92-TR-11, Carnegie Mellon University, Pittsburgh, PA 15213, June.