An Overview of Software Quality Concepts and Management Issues

ABSTRACT

This chapter introduces the generally accepted knowledge on software quality that has been included in the (SWEBOK) Software Engineering Body of Knowledge (ISOTR19759-05). One chapter of the SWEBOK is dedicated to software quality (Apr05). It argues that ethics play an important role in applying the quality models and the notions of cost of quality for software engineers. It also describes the minimal content required in a software quality assurance plan. Finally an overview of what to expect in the upcoming international standards on software quality requirements, which transcend the life cycle processes of all IT processes, is presented.

Keywords: Capability Maturity Model®[1], cost of quality, software quality requirements, software product quality, software quality measurement, software quality improvement, software quality fundamentals, software quality.

INTRODUCTION

The business value of a software product results from its quality as perceived by both acquirers and end-users. Therefore, quality is increasingly seen as a critical attribute of software, since its absence results in financial loss as well as dissatisfied users, and may even endanger lives. For example, Therac-25, a computer-driven radiation system, seriously injured and killed patients by massive overdosing (Lev93). Improving recognition of the importance of setting software quality requirements and of assessing quality causes a shift in the 'center of gravity' of software engineering from creating technology-centered solutions to satisfying stakeholders. Software acquisition, development, maintenance and operations organizations confronted with such a shift are, in general, not adequately equipped to deal with it. Until recently, they did not have at their disposal the quality models or

[1] Capability Maturity Model and CMM are registered trademarks in the U.S. Patent and Trademark Office.

measurement instruments to allow (or ‘facilitate’) the engineering of quality throughout the entire software product life cycle. The objective of software product quality engineering is to achieve the required quality of the product through the definition of quality requirements and their implementation, measurement of appropriate quality attributes and evaluation of the resulting quality. The objective is, in fact, software product quality.

This chapter is structured in accordance with the SWEBOK classification of the software quality body of knowledge (www.swebok.org):

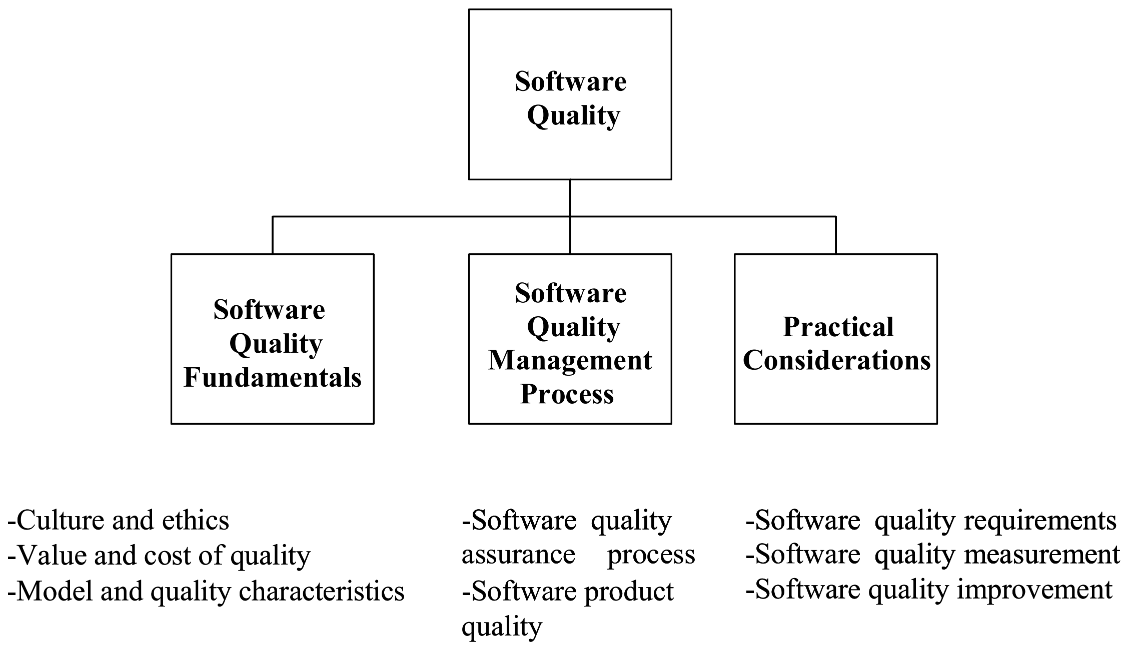

Figure 1. Adapted breakdown of software quality topics (ISOTR19759-05).

Software Quality Fundamentals

Agreement on quality requirements, as well as clear communication on what constitutes quality, require that the many aspects of quality be formally defined and discussed. Over the years, authors and organizations have defined the term “quality” differently. IBM used the phrase “market-driven quality”, which is based on achieving total customer satisfaction. This definition was influenced by the total quality

CMM Integration and CMMI are service marks of Carnegie Mellon University.

management approach of Phil Crosby (Cro79), who defined quality as "conformance to user requirements". Watts Humphrey (Hum89), looking at quality in the software industry, defined it as "achieving excellent levels of fitness for use". More recently, quality has been defined in (ISO9001-00) as "the degree to which a set of inherent *characteristics* fulfills the *requirements.*" The next section looks at how organizational culture and individual ethics play a role in quality in the organization.

Culture and Ethics

Edward B. Tylor (Tyl71) defines human culture as "... that complex whole which includes knowledge, belief, art, morals, law, custom, and any other capabilities and habits acquired by man as a member of society." Culture guides the behaviors, activities, priorities and decisions of an individual, as well as those of an organization. Karl Wiegers (Wie96), in his book *Creating a Software Engineering Culture*, illustrates (see Figure 1) the interaction between the software engineering culture of an organization, its software engineers and its projects. Why should we, as technical people, care about such a 'soft' issue? First, a quality culture cannot be bought. It needs to be developed, mostly at the beginning, by the founders of the organization. Then, as employees are selected and hired, the initial leader's culture will start to slowly adjust to the pressures of the environment, as shown in Figure 2. Quality culture cannot be 'bolted on' to an organization, it has to be designed-in and nurtured. The ultimate aim of upper management is to instill a culture that will allow the development of high-quality products, and offer them at competitive prices, in order to generate revenues and dividends in an organization where employees are committed and satisfied.

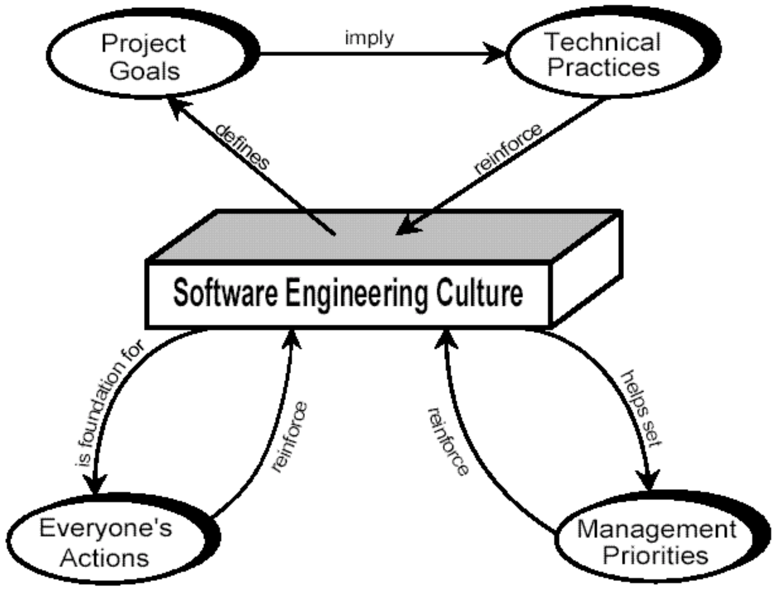

Figure 2. A software engineering culture (Wie96).

The second reason why we should be interested in the cultural aspects of quality is that any change an organization wants to make, for example moving up on the Capability Maturity Model integration[SM] (CMMi[SM]) (Sei02) maturity scale, cannot simply be ordered; the organization has to cope with the current culture when making a change in maturity, especially when such a change implies a profound change in that culture. An organization cannot just 'buy and deploy' off-the-shelf processes that contain quality. It has been demonstrated that one of the major inhibitors of change is the culture of the organization. The management of change is imperative (Lap98) in order to achieve the desired result.

The issue of quality is also central to the Code of Ethics, developed by and for software engineers, which was released in 1999 (Got99a, Got99b and IEEE-CS-99). The Code describes eight top-level technical and professional obligations against which peers, the public and legal bodies can measure a software engineer's ethical behavior. Each top-level obligation, called a principle, is described in one sentence and supported by a number of clauses which give examples and details to help in interpretation and implementation of that obligation. Software engineers adopting the Code commit to eight principles of quality and morality. The following are examples: Principle 3 (Product) states that software engineers shall ensure that their products and related modifications meet the highest professional standards possible. This principle is supported by 15 clauses, and clause 3.10 reads that software engineers shall ensure adequate testing, debugging and review of software and related documents on which they work. The Code has been translated into eight languages: Chinese, Croatian, English, French, Hebrew, Italian, Japanese and Spanish, and all versions are available on a Web site (http://seeri.etsu.edu/Codes/default.shtm). It has been publicly adopted by many organizations, as well as by a few universities as part of their curriculum in software engineering.

Value and Costs of Quality

To promote a quality approach to software, a number of arguments must be developed to support its value and benefits. Quality, as a whole, is not always perceived positively and is a hard sell for software project managers. One famous book (Cro79) addressing the costs of quality has been published on this topic. Since its publication, many organizations, mainly manufacturers, have successfully promoted the use of the cost of quality concepts and framework in their project management processes. A few papers have been published on the adoption of these concepts in the software industry (Sla98, Cam90, Man90, Dub99, Dia02 and Gal04a), but very few case studies have been published by the software industry itself (Dio93, Kno93, Hal96 and Hou02). A paper by Haley of Raytheon (Hal96) illustrates very well the links between the cost of rework and the investment needed to reduce waste. Haley also illustrates that a long-term perspective is needed, as well as a commitment from senior management, in order to capture the benefits. Another fact illustrated by Haley is the link between the investment/benefits and the Capability Maturity Model for Software (CMM) ® (Pau93) maturity level of an organization (see Figure 3). At the initial CMM® maturity level, most, if not all, organizations have no reliable data or measurement system. When an organization has reached levels 2 and 3, it has the foundation to measure and select beneficial process improvement areas. Many organizations have published the results obtained in climbing the maturity ladder. They show the relationships between maturity level, quality, productivity and project cost.

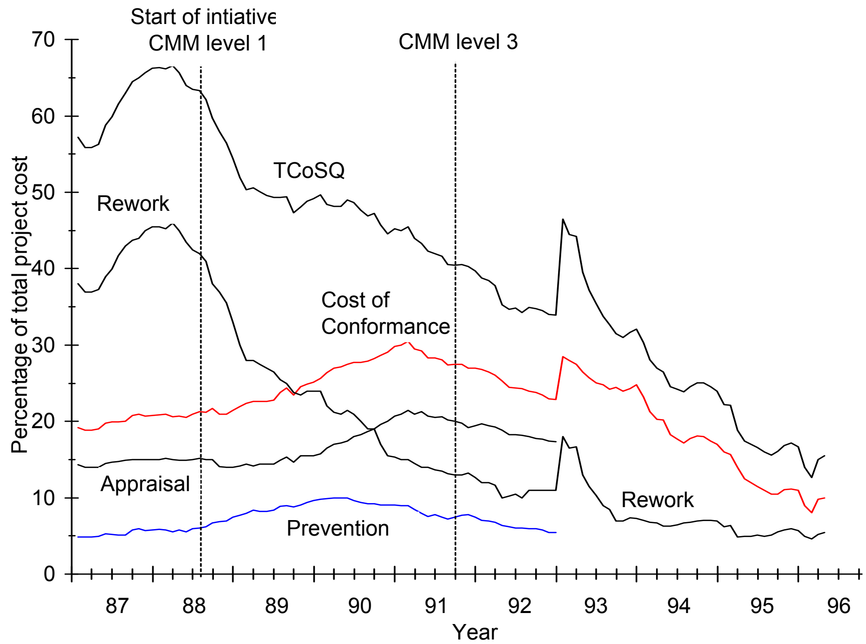

Figure 3. Improvement data (Dio93, Hal96).

Galin, in a recent software quality book (Gal04b), illustrates the cost and benefits of quality practices. As shown in Table 1, the addition of effective quality practices, such as inspection and review, even reduce the cost of a project by eliminating rework. The left-hand column lists typical quality assurance activities, such as reviews and inspections, of an ongoing project. The middle column shows the effectiveness of typical defect removal activities included in a software quality plan. It shows that such a process would deliver a product where 93.1% of the total defects have been detected, leaving 6.9% of them, for a QA cost of 1014.3 units. On the right-hand side, two activities have been added to the quality plan: a design inspection and a code inspection. Even though these activities are not free (18.5 and 158.6 units respectively), the net result is that 97.4% of the total defects have been detected, leaving only 2.6% of the defects in operation Note also that the total QA cost is 760.4 units, compared to 1014.3 units previously.

Quality assurance activity	Standard plan		Comprehensive plan	
	Percentage of defects removed	Cost of removing defects (cost units)	Percentage of defects removed	Cost of removing defects (cost units)
Requirements specification review	7.5%	7.5	9%	9
Design inspection	--	--	28.7%	71.8
Design review	21.3%	53.2	7.4%	18.5
Code Inspection	--	--	24.4%	158.6
Unit test – code	25.6	166.4	4.2%	27.3
Integration test	17.8%	284.8	9.8%	156.8
Documentation review	13.9%	222.4	9.9%	158.4
System test	7.0%	280	4%	160.
Total for internal QA activities	**93.1%**	**1014.3**	**97.4**	**760.4**
Defects detected during operation	6.9%	759	2.6%	286
Total	**100.0%**	**1773.3**	**100%**	**1046.4**

Table 1. Illustration of the cost and benefits of quality practices from (Gal04).

The following quote from Norman Augustine (Aug97) summarizes this section: “It costs a lot to build bad products.”

Model and quality characteristics

Quality models for software have been developed to assess both software processes and software products. Software process engineering has made many advances in the last decade, and has introduced numerous quality models and standards concerned with the definition, implementation, measurement, management, change and improvement of the software engineering process itself. Assessing the quality of software products requires that the quality characteristics of software be defined and measured. The quality of a software product is determined by its properties. Some of these are inherent to the software product, while others are assigned, by priority, to the software product during its requirements definition process. Process quality is concerned with the technical and managerial activities within the software engineering process that are performed during software acquisition, development, maintenance and operation.

In contrast, software product quality models describe the many quality attributes of software. These have been presented by many authors (Boe78, McC77) and have more history. Although this is an older topic, it has not quite reached the popularity of software process quality models. During the early years, the terminology for software product quality differed from one model of software quality to another. Each model had a different number of hierarchical levels and number of characteristics, and especially confusing were the different naming conventions used. Based on these efforts, the International Organization for Standardization (ISO) normalized three software product quality models (i.e. internal quality, external quality and quality in use) (ISO9126-01) and accompanied them with a set of guides like ISO/IEC14598 which explains how to evaluate a software product (see Figure 4).

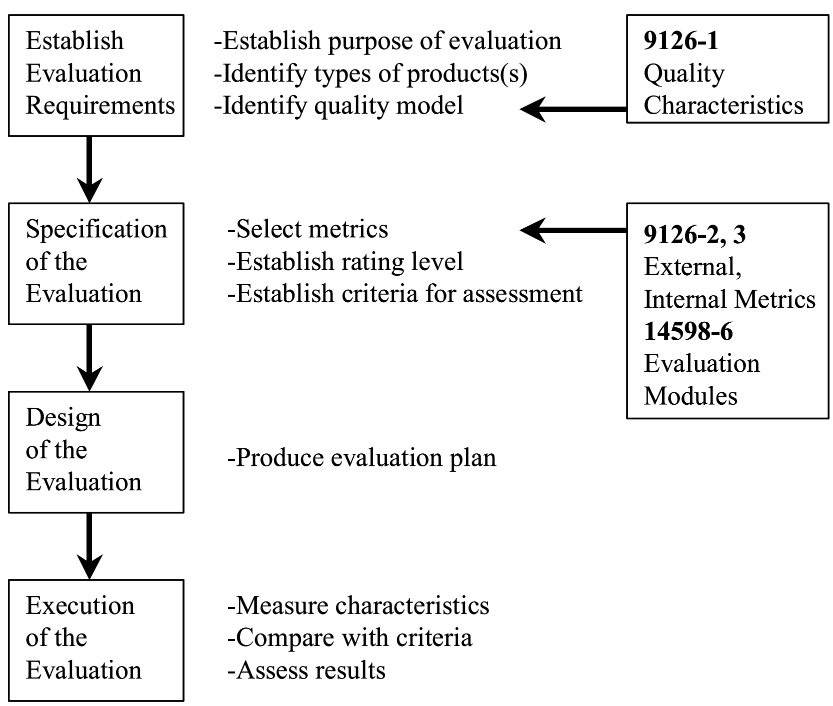

Figure 4. ISO/IEC 14598 – Evaluation process (Sur03)

Software Quality Management Process

Software quality management (SQM) defines processes, process owners, requirements for those processes, measurements of the process and its outputs and, finally, feedback channels (Art92). SQM processes are useful for evaluating the process outcome as well as the final product.

The Juran Trilogy Diagram, illustrated in Figure 5, depicts quality management as comprising three basic processes: quality planning, quality control and quality improvement. The diagram shows how waste caused by mismanaging quality, or not managing it, can be reduced by introducing quality improvement and feeding back the lessons learned from this into the quality planning process. Many, if not most, software engineering managers are not aware of the magnitude of the waste, i.e. between 30 and 40%, in their processes.

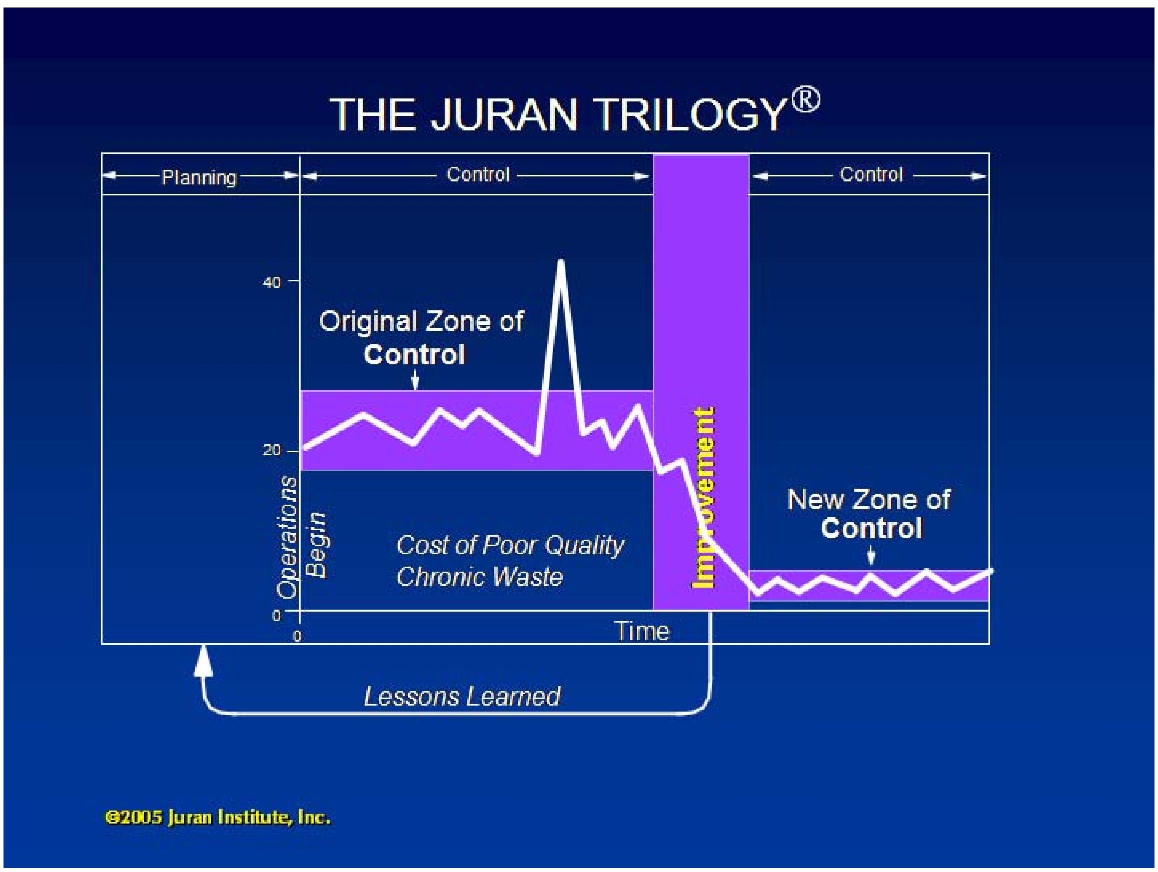

Figure 5. Juran Trilogy Diagram

Software quality management processes consist of many activities. Some may find defects directly, while others indicate where further examination may be valuable. Planning for software quality involves:

(1) defining the required product in terms of its quality characteristics;
(2) planning the processes to achieve the required product quality.

Some of the specific SQM processes are defined in standard (IEEE12207.0-96):

- Quality assurance process
- Verification process
- Validation process
- Review process
- Audit process

These processes encourage quality and also find possible problems, but they differ somewhat in their emphasis. SQM processes help ensure better software quality in a given project. They also, as a by-product, provide management with general information, including an indication of the quality of the entire software engineering process.

SQM processes consist of tasks and techniques to indicate how software plans (e.g. management, development and configuration management) are being implemented and how well the intermediate and final products are meeting their specified requirements. Results from these tasks are assembled into reports for management before corrective action is taken. The management of an SQM process is tasked with ensuring that the results of these reports are accurate.

As described in this Knowledge Area (KA), SQM processes are closely related; they can overlap and are sometimes even combined. They seem largely reactive in nature because they address the processes as practiced and the products as produced, but they have a major role at the planning stage in being proactive in terms of the processes and procedures needed to attain the quality characteristics and degree of quality needed by the stakeholders in the software.

Risk management can also play an important role in delivering quality software. Incorporating disciplined risk analysis and management techniques into the software life cycle processes can increase the potential for producing a quality product (Cha89). Refer to the Software Engineering Management KA of the SWEBOK for material related to risk management.

Software Quality Assurance Process

Software quality assurance processes have been found to have a direct effect on the quality of a software product. During a software development project, it is difficult to completely distinguish the quality of the process used to develop the software from the quality of the product itself. Process quality assurance influences process quality, which in turn influences the quality characteristics of the software product.

Process quality is addressed by two important process models, ISO9001 and CMMi℠ (Sei02). First, ISO9001:2000 (Iso9001-00) is an international standard, while CMMi℠ is a process model. Compliance with the key quality concepts of the ISO9001:2000 standard requires an interpretation for each industry. Software, being a service industry, has initially had more difficulty interpreting and implementing the ISO9001 standards (Bou99, Nie00 and May02). Service organizations also experience greater difficulty dissociating the final product from the processes used in its production (May02). To help with this problem, additional guidance has been developed by the ISO for the implementation of ISO9001:2000 in the software industry (Pea94, Hai98, May02). This guide, numbered ISO90003, helps link all the software-related standards to the quality system required by ISO9001:2000.

In contrast, CMMi[SM] (Sei02) is a process model which proposes proven practices at different maturity levels. It is divided into process areas, and key to quality management are the following three: a) process and product quality assurance; b) process verification; and c) process validation.

There was initially some debate over whether ISO9001:2000 or CMMi[SM] should be used by software engineers to ensure quality. It has been observed that many organizations use their ISO9001 certification assets as a stepping stone towards meeting the CMMi[SM] requirements. It is believed that level 3 of CMMi[SM] is reachable for an organization that already has ISO9001:2000 certification. In competitive situations, it is not unusual to find that the two are used concurrently to meet shared quality objectives.

In general, process quality models are concerned with process outcomes. However, in order to achieve the process outcomes desired (e.g. better quality, better maintainability, greater customer satisfaction), we have to implement and measure the process in action.

Current quality models are not complete at this time, as there are many other factors to consider which have an impact on outcomes. Other factors, such as the context, industry domain, capability of the staff and the tools used also play an important role (see Figure 6). "Furthermore, the extent to which the process is institutionalized or implemented (i.e. process fidelity) is important, as it may explain why "good" processes do not give the desired outcomes." (ISOTR19759-05)

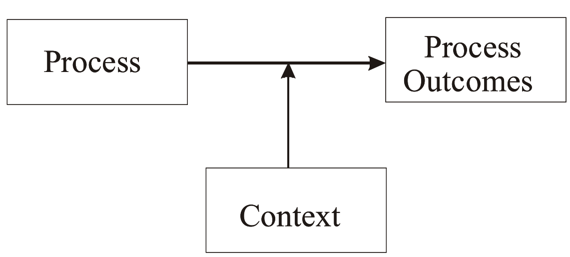

Figure 6. Relationships between process and outcomes (ISOTR19759-05).

A word of caution is needed with regard to the term "software quality assurance" (SQA). It is actually a misnomer, because SQA cannot *assure* quality, but only *provide adequate assurance*. ISO 12207 (ISO/IEC12207.0-96) defines an SQA process as a process for providing adequate assurance that the software products and processes in the project life cycle conform to their specified requirements and adhere to their established plans.

The persons or groups providing SQA services need to be independent of the developers in order to provide an unbiased opinion on the conformance of products and processes to the standards imposed by a customer, an organization or a country. SQA services can be provided by an entity that may be internal or external to the developing organization.

An SQA plan, often mandated by an SQA standard, derives its requirements from the contract, the project plan and the organization's quality system. It defines the means that will be used to ensure that software developed for a specific product satisfies the user's requirements and is of the highest quality possible within project constraints. The content and format of an SQA plan are available in government documents, such as ESA and NASA, or publicly, in standards such as IEEE standard 730 (IEEE730-02).

ISO/IEC 12207 proposes a minimal content for the SQA plan:

- Quality standards, methodologies, procedures and tools for performing quality assurance activities (or their references in the organization's official documentation);
- Procedures for contract review, and their coordination;
- Procedures for identification, collection, filing, maintenance and disposition of quality records; resources, schedule and responsibilities for conducting the quality assurance activities;
- Selected activities and tasks from processes, such as verification, validation, joint review, audit and problem resolution.

To illustrate the importance of SQA practices, Table 2 (Cus03) lists a few practices included in a sample of Japanese and American organizational processes. The data show the percentage of sampled projects using a particular practice.

Practices Used	Japan	US
Architectural specifications	70.4%	54.8%
Functional specifications	92.6%	74.2%
Detailed designs	85.2%	32.3%
Design reviews	100%	77.4%
Code reviews	74.1%	71%
Regression testing	96.3%	71%

Table 2. Organizational quality practices (Cus03).

Table 3 from the same paper shows the performance data resulting from implementation of the practices. As can be observed, the increased performance of the practices in Japan resulted in increased productivity and quality performance in that country.

Performance Crieria	Japan	US
Median output (see note 1)	469	270
Median defect rate (see note 2)	.020	.400
Note 1: Output per programmer-month of effort in new Lines of Code (LOC). *Note 2: Number of defects reported per 1000 LOC (in the 12 months after delivery to customers).*		

Table 3. Performance comparison (Cus03).

Table 4 illustrates the important improvements, especially in quality, measured in Japan and the US over the ten-year period.

	Japan – 1990	Japan - 2003	US – 1990	US -2003
Productivity	389	469	245	270
Quality	.20	.020	.83	.400

Table 4. Productivity and Quality improvements over a decade (Cus03).

Software product quality

The quality of software is not only influenced by the quality of the process used to develop it, but by the quality of the product as well. Software product quality has also been the subject of much research, and of many international standards and publications. Software engineers determine, with the stakeholders, the real purpose of future software. In this regard, it is of prime importance to keep in mind that the customer's requirements come first, and that they should include non-functional requirements like quality, not just functionality. Thus, the software engineer has a responsibility to elicit quality requirements from future software which may not be made explicit at the outset. It is key that he assess their relative importance, as well as the level of difficulty in achieving each of them. The section on value and cost of quality presented the additional processes associated with defining and assessing software quality that need to be added to software projects, and each carries additional costs.

Software product quality standards define the related quality characteristics and sub-characteristics, as well as measures which are useful for assessing software product quality (Sur03). In the standards, the meaning of the term "product" is extended to include any artifact which is a software process outcome used to build the final software product. Examples of a product include, but are not limited to, an entire system requirements specification, a software requirements specification for a software component of a system, a design module, code, test documentation or reports produced as a result of quality analysis tasks.

The software product quality model provided in ISO/IEC 9126-1 (Iso9126-01) defines six quality characteristics: functionality, reliability, usability, maintainability, portability and efficiency, as well as quality in use, which is defined as effectiveness, productivity, safety and satisfaction (see Figure 7). The six quality characteristics have defined sub-characteristics, and the standard also allows for user-defined components. The intention is that the defined quality characteristics cover all quality aspects of interest for most software products, and as such can be used as a checklist for ensuring complete coverage of quality early in the specifications phase.

The quality model suggests three views: internal quality, external quality and quality in use. “Internal quality provides a ‘white box’ view of the software and addresses properties of the software product that typically are available during development. External quality provides a ‘black box’ view of the software and addresses properties related to its execution. “The quality-in-use view is related to application of the software in its operational environment, for carrying out specified tasks by specified users. Internal quality has an impact on external quality, which again has an impact on quality in use” (Sur03). External and internal quality models presented by ISO/IEC9126 are less trivial and have to be explained further. Let us take the maintainability of software as a quality attribute, and see both the external and internal perspectives in action.

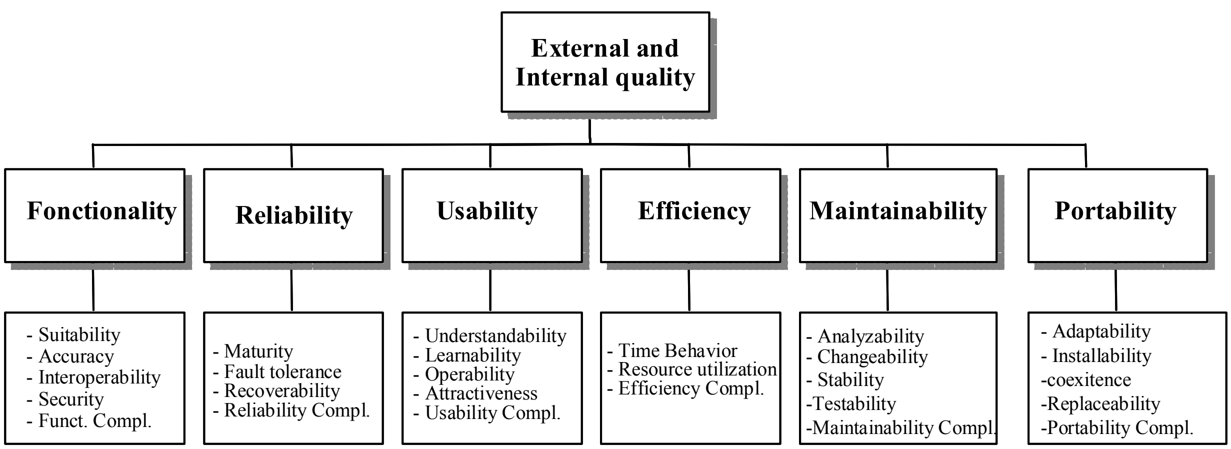

Figure 7. External/Internal quality of ISO/IEC 9126 (Iso9126-01).

From an external point of view, maintainability attempts to measure the effort required to diagnose, analyze and apply a change to a software product. Thus, an external measure of maintainability would constitute the effort to make a specific change.

From an internal product point of view, the quality model proposes measurement of the attributes of the software that influence the effort required to modify it. A good example is source code complexity. Structural complexity measures are generally extracted from the source code using observations on the program/classes/module/functions and graph representation of software source code. The more complex the source code, the more difficult it is to grasp its meaning, and to understand and document it. These studies are based on work carried out in the ’70s by McCabe, Halstead and Curtis (McC76, Hal78 and Cur79). These are all internal measures, which in turn affect the external point of view of maintainability.

While most product quality publications have described product quality in terms of the final software and system performance, sound software engineering practice requires that key artifacts impacting quality be evaluated throughout the software development and maintenance processes.

Practical Considerations

Software Quality Requirements

In order to implement software process and product quality, the software engineer must choose and document quality characteristics at the outset of the projects during the requirements definition process. A generic life cycle model like ISO/IEC 15288 – *System life cycle processes* demonstrates generic stages of a typical development process (Figure 8) and the mapping to show *how* standards can be used across all stages of the life cycle of a software product.

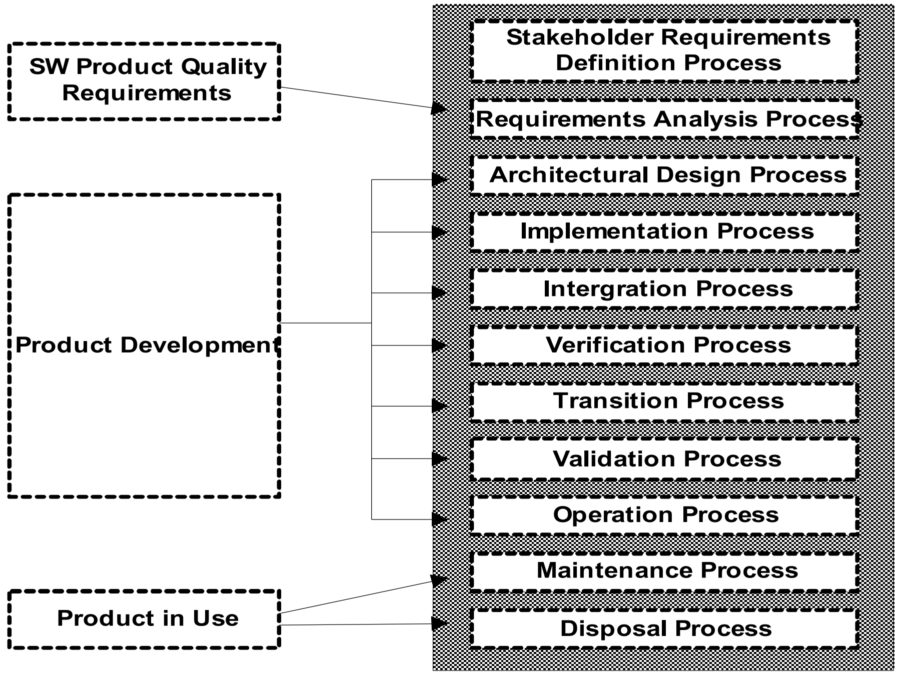

Figure 8. Product life cycle mapping to the technical process life cycle (Sur03).

We have already stated that both functional and non-functional requirements are captured using the requirements definition process (Pfl01). Functional requirements specify what a product should do. Non-functional requirements, also called quality requirements, place constraints on the functional requirements. The quality requirements need to refer to a quality model like the one described in ISO/IEC 9126-1 to be well understood. Quality requirements address important issues of quality for software products. Software product quality requirements are needed for:

- Specification (including contractual agreement and call for tender)
- Planning (e.g. feasibility analysis and translation of external quality requirements into internal quality requirements)
- Development (early identification of quality problems during development)
- Evaluation (objective assessment and certification of software product quality)

The requirements definition process typically takes all stakeholders' needs, requirements and expectations into consideration and ranks them by priority. All relevant stakeholders must be identified before the exercise begins. A first analysis activity during the requirements definition process is aimed at identifying user requirements, sometimes also called business requirements. This view of software requirements should be documented in wording that is easily understood by both users and customers.

Business requirements describe the functionality that the software executes as needed by the stakeholders. Good functional requirements are expressed clearly and are consistent. All requirements need to be correct, non-redundant and not be in conflict with other requirements. Each requirement has to be uniquely identified and traceable to identified stakeholder(s). They should also be documented in the business requirements specification (BRS). If a stakeholder requirement is not taken into consideration, the software engineer needs to state the reason why in his documentation.

The second analysis activity is aimed at transforming stakeholder functional requirements into system requirements that can be used by designers to build software. This more technical view of requirements is called a software requirements specification (SRS). During this second analysis activity, the quality requirements are added to the functional requirements. They need to be stated clearly and unambiguously. It is the software engineer's responsibility to go through each quality characteristic and assess the need to include it in the project. System requirements should also be verifiable, and state both the business requirements and the non-functional requirements that the software must possess in order to satisfy all stakeholder requirements. If they do not, they may be viewed, interpreted, implemented and evaluated differently by different stakeholders. If the software engineer does not become involved in promoting the non-functional requirements, they may be ignored altogether. This may result in software which is inconsistent with user expectations and therefore of poor quality.

We have often talked about stakeholders, but who are they? Stakeholders include all individuals and organizations with a legitimate interest in the software. Different stakeholders will express different needs and expectations that represent their own perspective. For example, operators, testers and quality engineers are stakeholders. These needs and expectations may change throughout the system's life cycle and need to be controlled when changed. The stakeholders rarely express non-functional requirements, as they may have an unclear understanding of what quality really means in a software process. In reality, they typically perceive it initially as more of an overhead than anything else. Quality requirements will often appear as part of a contractual agreement between acquirer and developer. They may also be required for a specific product quality evaluation when some level of software criticality is reached and human lives are at risk.

We have already stated that a software quality requirement should refer to a specific quality model. To clearly specify quality requirements, ISO/IEC working group six (WG6) of the software engineering subcommittee (SC7) is developing a standard as part of the new ISO/IEC 25000 – *SQuaRE* (*S*oftware Product *Qua*lity *R*equirements and *E*valuation) initiative. The Quality Requirements standard will include a new section for the specification of quality requirements.

A number of obligations are likely to be imposed on software engineers that demand conformance to this new section of ISO/IEC 25000. First, each quality requirement will need to be categorized and prioritized according to an explicit quality model, like:

- Quality-in-use requirement;
- External quality requirement; and
- Internal quality requirement.

A software quality requirement will also need to be specified with reference to a set of functional properties. It is also possible that critical functional properties will create an obligation to impose some quality characteristics on a quality requirement.

Since each quality requirement needs to be assessed throughout all the life cycle stages, it needs to be specified in terms of a measurement function and be assigned target values. This new standard will provide examples of measures which may be adapted to specific organizational needs for specifying quality requirements. This new standard is also likely to recommend that software engineers develop and maintain a list of quality measures that are used consistently across their organization. Applying the same set of measures makes it possible to create a homogeneous repository of historical data, which in turn helps in the creation of predictive models such as productivity and estimation models.

The new standard could also focus on the notion of the degree of uncertainty associated with a quality requirement. Measures tending to yield large deviations need to be explained because of their erratic behavior. Explanations include the measurement process of that measure and the errors associated with each of its components.

Many quality standards already identify the review and approval stages, so that the set of quality requirements is identified and reviewed, and who approved them is clearly stated. The last item that is likely to appear in such a standard refers to the need for quality requirements, just like functional requirements, to be managed according to a configuration and change management system throughout the entire project.

Software Quality Measurement

The main reason why product quality measurement is only slowly gaining acceptance is the inherent difficulty of measuring in a practical way its quality-impacting processes and artifacts throughout the software engineering process. Some product quality perspectives (e.g. size and source code structure) have been easier to measure than others (e.g. suitability, replaceability and attractiveness). “Furthermore, some of the dimensions of quality (e.g. usability, maintainability and value to the client) are likely to require measurement in qualitative rather than quantitative form” (ISOTR19759-05).

For practical purposes, the software engineer will need to define the quality requirements and the means to assess them early on in the life cycle of the project. During the requirements definition stage, statements about the quality of each quality characteristic are defined as expressing the capability of the software to perform at, and maintain, a specified level of service. The measurement approach requires that individuals who carry out software engineering activities capture the measures and make a judgment about the degree to which the many software properties fulfil their specified requirements as agreed by the stakeholders. Both qualitative and quantitative measures can be collected during all stages of the software life cycle.

The ISO9126 quality model offers definitions of software properties which can be distinguished as quantitative or qualitative. Before quality attributes can be captured and validated, someone has to design them. Then, the most practical means possible must be implemented to measure them.

We measure by applying measurement method rules (see step 2 in Figure 9). A measurement method is a logical sequence of operations that can be used by software engineering personnel to quantify an attribute on a specified scale. The result of applying a measurement method is called a base measure. The ISO9126 quality characteristics that were defined and agreed on can now be measured.

A list of "Quality Measures" is available in ISO/IEC 9126 and ISO/IEC 14598, and the standards present examples of mathematical definitions and guidance for practical measurement of internal quality, external quality and quality in use.

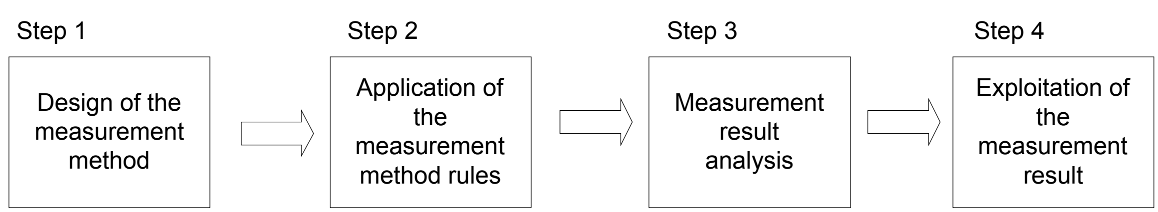

Figure 9. Software Measurement Method (Abr97)

The proposed ISO/IEC 25000 – *SQuaRE* will help by providing guidance, as follows:

- *'Measurement reference model and guide* – to present introductory explanations, the reference model and the definitions that are common to measurement primitives, internal measures, external measures and quality-in-use measures. The document will also provide guidance to users for selecting (or developing) and applying appropriate measures;

Example: the construction phase creates intermediate, non-executable software products that can only be evaluated from a static point of view. In this case, the user will be directed to the internal quality measures standard, where he can choose the measures that best serve his information needs;

- *Measurement primitives* – to define a set of base and derived measures, being the measurement constructs for the internal quality, external quality and quality-in-use measurements;
- *Measures for internal quality* – to define a set of internal measures for quantitatively measuring internal software quality in terms of quality characteristics and subcharacteristics;
- *Measures for external quality* – to define a set of external measures for quantitatively measuring external software quality in terms of quality characteristics and subcharacteristics;
- *Measures for quality in use* – to define a set of measures for measuring quality in use. The document will provide guidance on the use of the quality-in-use measures.

Software Quality Improvement

As defined by the ISO/IEC 12207 standard (ISO/IEC 12207:95), an Improvement Process is a process for establishing, assessing, measuring, controlling and improving a software life cycle process. This definition assumes that an organization already has a written development process in place. A first step in any improvement strategy is to assess the current situation (i.e. a baseline), and then to identify improvement goals and implement the actions required to achieve the goals. Similarly, quality improvement should follow the same steps. Important assumptions are that the organization has stable processes such that data collected are reliable and that they are quality data. If no data are available, some baseline activities have to be performed. The measurement and analysis process area of the CMMi (Sei02) describes the basic components of such a process.

The organization then has to establish quality goals. One approach is to use the well-known GQM method (Bas88). The organization determines its business goals, and then the technical personnel establish quality goals in order to meet the organization's goals. The quality goals could be established using the Cost of Quality approach or a standard business case approach, i.e. by evaluating the cost of prevention and evaluation, and the cost of internal and external failure.

Quality could be improved, first, by removing defects as early as possible in the development life cycle using techniques such as reviews, walkthroughs and inspections (IEEE-1028), and second, by putting in place defect prevention activities. Software inspection (IEEE-1028, Rad02, Gil93) is an example of a widely recognized quality improvement practice. A Web site displays this technique, as well as providing a wealth of supporting information (Http://www.goldpractices.com/practices/fi/index.php). The inspection process is typically composed of six stages: plan, kick-off meeting, document checking, logging meeting, editing (i.e. rework) and follow-up (Figure 10).

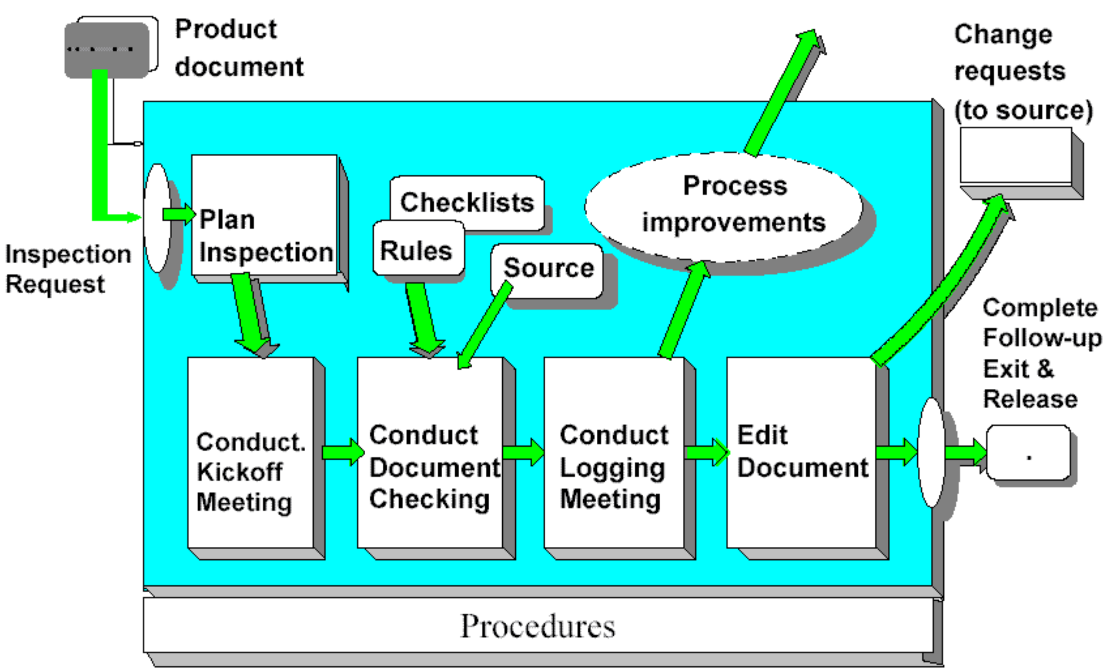

Figure 10. The Inspection Process (adapted from (Hol98)).

As shown in Table 5, the defect detection effectiveness of inspections can reach 84%. Radice (Rad02) has collected data from organizations of different CMM maturity levels which show an effectiveness varying from as low as 50% to as much as 100% (Table 6).

Defect Detection Technique	Minimum Value	Most Likely Value	Maximum Value
Design Inspections	25%	57%	84%
Code Inspections	19%	57%	70%

Table 5. Distribution of defect detection effectiveness of inspections (Bri98)

CMM Maturity Level	Defect Removal Effectiveness
Level 5	90% - 100%
Level 4	75% - 90%
Level 3	65% - 75%
Level 2	50% - 65%
Level 1	Less than 50%

Table 6. Defect detection effectiveness (Rad02).

It is interesting to note that using inspection to detect defects is more effective than test. The table below (Bri98) shows the average effort required to detect defects in hours per defect. Unfortunately, even though it has been around for close to 30 years (Fag76), most organizations ignore this practice. Many organizations, if not most, rely only on tests performed toward the end of the development process to identify defects, even though tests are much less effective than inspections at defect removal. One of the reasons is that inspections find defects where they occur, while testing finds only the symptoms and only very late in the development cycle.

Defect Detection Technique	Minimum Value	Most Likely Value	Maximum Value
Design Inspections	0.58	1.58	2.9
Code Inspections	0.67	1.46	2.7
Testing	4.5	6	17

Table 7. Distribution of Average Effort to Detect Defects (Bri98)

Future Trends and Conclusions

To illustrate the current state of the quality of many shrink wrap software vendors, here is a typical warranty statement as printed on a software product: *By opening this sealed software media package, you accept and agree to the terms and conditions printed below. If you do not agree,* ***do not open the package****. Simply return the sealed package. The software media is distributed on an 'AS IS' basis, without warranty. Neither the authors, the software developers nor Acme Inc make any representation, or warranty, either express or implied, with respect to the software programs, their quality, accuracy, or fitness for a specific purpose. Therefore, neither the authors, the software developers nor Acme Inc shall have any liability to you or any other person or entity with respect to any liability, loss, or damage caused or alleged to have been caused directly or indirectly by the programs contained on the media. This includes, but is not limited to, interruption of service, loss of data, loss of consulting or anticipatory profits, or consequential damages from the use of these programs. If the media is defective, you may return it for a replacement.*

As more customers demand quality from their vendors, the competitive marketplace will eventually respond to these demands and software products will be warranted in the same way as any other products. Vendors will be held responsible for making the appropriate corrections at their expense like other industrial sectors. Already a few organizations have demonstrated that quality, productivity, cycle time and cost could all be improved provided that senior management commit adequate resources and time to improvement. Diaz (Dia97, Dia02) has published results of major improvement programs at Motorola and General Dynamics. For example, at General Dynamics, rework dropped from 23.2% to 6.8%, phase containment effectiveness increased from 25.5% to 87.3%, defects reported by customers dropped from 3.2 per thousand source lines of code (KSLOC) to 0.19, and productivity increased by a factor of 2.9. Such organizations have the data to manage the quality of their processes and their products. As stated by Boehm and Sullivan (Boe00), since design is an investment activity, knowing the economics of software engineering, these organizations can offer their customers an optimized value proposition by using the proper processes, tools and metrics.

There is no instant gain, however, as illustrated in Figure 3. It takes time to define and stabilize a process, measure its performances and make improvements. Those organizations that have succeeded know that improvements require not only resources, but also a major culture change. By definition, a cultural change takes time and can rarely be imposed, and it has to be managed from the top, with senior executives no longer requesting unrealistic schedules with unrealistic budgets to deliver quality products. Moreover, if software developers in the West do not shape up rapidly, what has happened and is still happening to the automobile industry may happen to them, i.e. a take-over by foreign, high-quality, lower-cost manufacturers.

With the arrival of India (Moi01) and China (Ju01) into the software market with their focus on quality and process maturity, the software industry will be pushed to move faster, otherwise Darwin will favor those who survive this fight.

REFERENCES

(Abr97) Abran, A., Jacquet, J-P.(1997). From Software Metrics to Software Measurement Methods: A Process Model, *Third International Symposium and Forum on Software Engineering Standards (ISESS97)*, Walnut Creek CA, June 2-6.

(Apr05) April, A., Reeker, L., Wallace, D. (2005). 'Software Quality', in "Ironman version of the Guide to the Software Engineering Body of Knowledge", IEEE-Computer Society Press, 157–170.

(Art92) Arthur, L.J.(1992). *Improving Software Quality: an insider's guide to TQM*. John Wiley & Sons.

(Aug97) Augustine, N.R.(1997) *Augustine's Laws* (6th ed.), American Institute of Aeronautics and Astronautics: Reston VA.

(Bas88) Basili, V,R,, Rombach, H.D.(1988). The TAME project: Towards improvement-oriented software environments, *IEEE Transactions on Software Engineering*, 14(6).

(Boe78) Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., McLeod, G., Merritt, M.(1978). Characteristics of software quality. *TRW series on Software Technologies,* North-Holland : Amsterdam Netherlands.

(Boe00) Boehm, B.W., Sullivan, K.J.(2000). Software Economics: a Roadmap, Proceedings of the Future of Software Engineering, International Conference on Software Engineering, Limerick, Ireland.

(Bou99) Bouman, J., Trienekens, J., Van der Zwan, M.(1999). Specification of service level agreements, clarifying Concepts on the basis of practical research, *Proceedings of Software Technology and Engineering Practice*. IEEE Computer Society: Los Alamitos CA, 11–19.

(Bri98) Briand, L., El Emam, K., Laitenberger, O., Fussbroich, T.(1998). Using simulation to build inspection efficiency benchmarks for development projects, *Proceedings IEEE 20th International Conference on Software Engineering (ICSE 1998),* IEEE Computer Society, Los Alamitos CA, 340–349.

(Cam90) Campanella, J.(1990). *Principles of Quality Costs*. third ed. American Society for Quality Control: Milwaukee WI.

(Cha89) Charette, R.N.(1989). *Software Engineering Risk Analysis and Management*. McGraw-Hill: New York NY.

(Cro79) Crosby, P.B.(1979). *Quality is free*. McGraw-Hill: New York NY.

(Cur79) Curtis, B.(1979). In search of software complexity, *Proceedings IEEE PINY Workshop on Quantitative Software Models*, Catalog No. TH0067-9. IEEE Computer Society, Los Alamitos CA, 95–106.

(Cus03) Cusumano, M., MacCormack, A., Kemerer, C., Grandall, B.(2003). Software development worldwide: the state of the practice, *IEEE Software*, 20(6): 28–34. *Figures abstracted from IEEE Software*.

(Dia97) Diaz, M.(1997). How software process improvement helped Motorola, *IEEE Software*, 14(5): 75–81.

(Dia02) Diaz M. How CMM impacts quality, productivity, rework, and the bottom line. *Crosstalk Journal*, U.S. Department of Defense, Hill Air Force Base UT, 2002; 15(3):9–14.

(Dio93) Dion, R.(1993). Process improvement and the corporate balance sheet. *IEEE Software*, 10(4): 28–35. *Figure abstracted from IEEE Software.*

(Dob99) Dobbins, H.D.(1999). The cost of software quality. *Handbook of Software Quality Assurance*, (3rd ed.). Prentice Hall PTR: Upper Saddle River NJ, 195–216.

(Fag76) Fagan, M.E.(1976). Design and code inspections to reduce errors in program development, *IBM System Journal*, 15(3):182–211.

(Gal04a) Galin, D.(2004). *Toward an Inclusive Model for the Costs of Software Quality*, SQP; 6(4): 25–31.

(Gal04b) Galin, D.(2004). *Software Quality Assurance*, Addison-Wesley: Harlow England.

(Gil93) Gilb, T., Graham, D.(1993). *Software Inspection*, Addison Wesley.

(Got99a) Gotterbarn, F.(1999) How the new software engineering code of ethics affects you, *IEEE Software*, 16(6):58–64.

(Got99b) Gotterbarn, D., Miller, K., Rogerson, S.(1999). Computer Society and ACM Approve Software Engineering Code of Ethics, *IEEE Computer*, 32(10):84–88.

(Hai98) Hailey, V.A.(1998). A comparison of ISO9001 and the SPICE framework, SPICE: an empiricist's perspective, *Proceedings of the Second IEEE International Software Engineering Standards Symposium (ISESS 1998)*. IEEE Computer Society: Los Alamitos CA, 233–268.

(Hal96) Haley, T.J.(1996). Software process improvement at Raytheon, *IEEE Software*, 13(6):33–41. *Figure abstracted from IEEE Software.*

(Hal78) Halstead, M.H.(1978). Software science: a progress report, *Proceedings IEEE Second Software Life Cycle Management Conference*, Atlanta GA, 174–179.

(Hol98) Holland, D.(1998). Document Inspection as an Agent of Change. in "Dare to be Excellent", Eds. Alka Jarvis and Linda Hayes. Upper Saddle River, NJ: Prentice Hall.

(IEEE 730-02) IEEE Std 730-2002. (2002). *IEEE Standard for Software Quality Assurance Plans*: IEEE.

(IEEE 1028-97) IEEE Std 1028-1997. (1997), *IEEE Standard for Software Reviews*: IEEE.

(IEEE 12207.0-96) IEEE/EIA 12207.0-1996. (1996). *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, vol. IEEE.

(IEEE-CS-99) IEEE-CS-1999.(1999) Software Engineering Code of Ethics and Professional Practice, IEEE-CS/ACM, 1999, Http://www.computer.org/certification/ethics.htm [02 February 2005]

(ISO90003-04) International Standards Organization. Software Engineering: *Guidelines for the application of ISO9001:2000 to computer software*, ISO/IEC Standard 90003:2004.(2004). International Organization for Standardization/International Electrotechnical Commission: Geneva Switzerland.

(ISOTR19759-05) International Standards Organization. *Software Engineering Body of Knowledge,* Technical Report ISO/IEC PRF TR 19759.(2005).

(Iso9126-01) International Standards Organization. *Software Engineering-Product Quality-Part 1: Quality Model*, ISO/IEC Standard 9126-1.(2001) International Organization for Standardization/International Electrotechnical Commission: Geneva Switzerland.

(Ju01) Ju, D.(2001) China's budding software industry, *IEEE Software*, 18(3):92–95.

(Kno93) Knox, S.T.(1993). Modeling the cost of software quality. *Digital Technical Journal*, 5(4): 9–16.

(Lap98) Laporte, C.Y., Trudel, S.(1998). Addressing the people issues of process improvement activities at Oerlikon Aerospace. *Software Process-Improvement and Practice*, 4(1):187–198.

(Lev93) Levenson, N., Turner, C.(1993). An Investigation of the Therac-25 Accidents, *IEEE Computer,* 26(7): 18–41.

(Man90) Mandeville, W,A.(1990). Software cost of quality, *Proceedings IEEE Journal on Selected Areas on Communications*, 8(2): 315–318.

(May02) May, W.(2002). A global applying ISO9001:2000 to software products, Quality Systems Update, 12(8): 7–13.

(Moi01) Moitra, D.(2001). India's software industry*, IEEE Software*, 18(1):77–80.

(McC76) McCabe, T,J.(1976). A complexity measure, *IEEE Transactions on Software Engineering*, 2(4): 308–320.

(McC77) McCall, J., Richards, P., Walters, G.(1977). "*Factors in software quality*", Vol. I-III, Rome Air Defense Centre, New York, USA.

(Nie00) Niessink, F.(2000). Perspectives on improving software maintenance. *Doctoral dissertation*. Dutch Graduate School for Information and Knowledge Systems: Utrecht Netherlands.

(Pau93) Paulk, M., Curtis, B., Chrissis, M.B., Weber, C.V.(1993). *Capability Maturity Model for Software version 1.1,* Software Engineering Institute, CMU-SEI-93-TR-24.

(Pfl01) Pfleeger, S.L.(2001). *Software Engineering: theory and practice.* Prentice Hall: Englewood Cliffs NJ, 2nd edition.

(Rad02) Radice, R.(2002). High quality low cost software inspections. Paradoxicon: Andover MA.

(SEI02) Software Engineering Institute. *Capability maturity model integration for software engineering (CMMi)*.(2002). Version 1.1, CMU/SEI-2002-TR-028. Carnegie Mellon University: Pittsburgh PA, 94–528.

(Sla98) Slaughter, S.A., Harter, D.E., Krishnan, M.A.(1998). Evaluating the cost of software quality, *Communications of the ACM* , 41(8): 10–17.

(Tyl71) Tylor, Sir E.B.(1871). Primitive Culture: Researches into the development of mythology, philosophy, religion, art and custom, London UK.

(Wie96) Wiegers, W.(1996). *Creating a Software Engineering Culture*, Dorset House: New York NY.