EPM/RT-95/03

A Software System Evaluation Framework

gratuit

Germinal Boloix
Pierre N. Robillard

Département de génie électrique
et génie informatique

École Polytechnique de Montréal
Janvier 1995

Dépôt légal, novembre 1993
Bibliothèque nationale du Québec
Bibliothèque nationale du Canada

Pour se procurer une copie de ce document, s'adresser:

Les Éditions de l'École Polytechnique
École Polytechnique de Montréal
Case postale 6079, succ. Centre-ville
Montréal, (Québec) H3C 3A7
Téléphone: (514) 340-4473
Télécopie: (514) 340-3734

A Software System Evaluation Framework

Germinal Boloix, Pierre N. Robillard
November 1994
École Polytechnique de Montréal
Département de Génie Electrique et de Génie Informatique
C.P. 6079 succ Centre Ville
Montréal, Québec H3C 3A7
Tel. (514) 340-4031, 340-4238 - Fax. (514)340-3240
boloix@rgl.polymtl.ca
pnr@rgl.polymtl.ca

Abstract

This paper proposes a software system evaluation framework which provides a classification scheme to identify sets of similar systems. The framework integrates previous studies on software evaluation, productivity models, software quality factors and total quality models; the appendix provides a summary of these approaches. The framework provides a taxonomy to classify information about software systems which integrates three perspectives: project, system and environment. The project view (i.e., developers, enhancers) considers the characteristics of the agents, the process and the tools used during software system production. The system view (i.e., operators, administrators or managers) depicts the characteristics of the software system, its technology and its performance. Finally, the environment view (i.e., users, stakeholders) identifies users' concerns, such as the degree of compliance of the software system with its requirements, the usability of the system and the contribution of the system to the organization. The framework can help software engineering non-experts, those which require software system evaluations for practical purposes.

Keywords: software engineering, software system evaluation, software quality, productivity factors, quality factors, total quality management, software metrics

1.0 Introduction

Information about software systems represents a useful resource for assessing the intrinsic characteristics of software, the software production process and the utility of systems in their environment. Information about software systems is important from various perspectives: those of producers, operators and users. The amount of available information about software systems and their interrelationships is impressive, and simplifying mechanisms are required to make them useful.

Management personnel, normally non-expert users of the information, require high-level information to understand the characteristics of software systems for a number of purposes, such as planning and estimation, the monitoring of project progress and the evaluation of project results. A software system evaluation framework for hierarchically organizing the various types of information to be gathered, which characterize a software system, is therefore needed. Eventually, detailed information may be required for purposes of confirming certain trends. If the information is well organized, according to a structured framework, specific aspects of software systems can be identified to improve decision-making activities.

Information related to software systems includes intrinsic software attributes, the process for producing the software and the contribution of the system to the organization. Productivity in the development process is a familiar concern which affects the cost of software systems. In fact, personnel considerations are of paramount importance to the cost of a system. The complexity of a system is another concern: the characteristics of the system determine how easily it is produced and maintained, which in turn impacts the total life-cycle costs. Individual and organizational productivity are affected by the use of computer systems, therefore the contribution of a system is fundamental to improving organizational performance.

In the information systems area, post-implementation evaluation approaches have been suggested to establish the worth of information systems.[1] Multiple-criteria evaluation approaches, which include subjective and objective evaluations, give equal consideration to both user and system constraints.[2] The problem of conflicting user points of view has been highlighted in these approaches. User, manager and developer evaluations are considered within a goal-centered view to compare pre-established objectives to actual results. The Goal-Question-Metric (GQM) paradigm[3] provides a mechanism for formalizing the characterization, planning, construction, analysis, learning and feedback tasks by establishing a systematic approach for setting project goals and defining them in an operational and tractable way. As objectives may vary for different projects, a common reference framework would be helpful for multi-project comparisons.

Other authors have identified productivity factors that affect software systems production. In general, these approaches emphasize product, personnel, project and computer attributes. Software quality has been another area contributing to the identification of software factors to be evaluated. Quality factors primarily evaluate the product and its interaction with users, neglecting the development process. Existing productivity and quality models disregard the importance of the contribution of a system to the organization. SEI's Capability Maturity Model (CMM), which assesses the software process maturity level of organizations, has been proposed to improve current software production practices. Another area that provides wider organizational insight, by considering the importance of customers, is Total Quality Management (TQM). The Malcolm Baldrige National Quality Award and the European Quality Award are examples of quality evaluation approaches that go beyond software systems. An overview of these approaches is given in the appendix.

These productivity and quality approaches demand huge amounts of information to characterize software system attributes. Evaluation of a software system following these approaches requires weeks or even months to produce an assessment of quality. Even after results are available, their interpretation is difficult because there are no mechanisms for synthesizing the information. By establishing a top-down approach to identify the important elements to be captured, high-level understanding is possible because the number of elements is limited and structured. A top-down approach also has the advantage of flexibility, permiting extensions by following a predefined pattern.

The importance of the various types of information to be gathered during software system projects is also recognized in conceptual modeling. Mylopoulus[4] suggested that the important types of knowledge be gathered about a system in what has been called 'worlds'. The *usage* world records information about the (organizational) environment; the *development* world describes the process that led to the development (or maintenance) of the information system; the *system* world describes the system at different layers of implementation detail; and the *subject* world consists of the subject matter for the system. Even though these knowledge worlds were not suggested specifically to evaluate software systems, their knowledge content can certainly overlap the software evaluation and measurement arena.

User satisfaction and economic returns are some important considerations for evaluating software systems effectiveness. Evaluations of the system can be performed prior to undertaking system development, while it is being developed (normally after each stage in the software life cycle), just before or after installation, and once the system has had a chance to settle down.[1] However, the number of complex, tangible and intangible factors involved requires a multiple-viewpoint approach for evaluation. [2] Organizing the multitude of factors according to different points of view, i.e., producers, operators and users, facilitates the selection of important metrics to be used during software system evaluation. However, it is important to recognize that finding concensus in the evaluation is a difficult task. There are many viewpoints, with multiple contributions.

Our objective is to develop a software system evaluation framework providing a basic set of attributes to characterize the important dimensions of software systems. The categories defined in the framework have to be clear in order to facilitate the evaluation process and reduce the conflicting points of view of evaluators. A top-down approach identifies the main dimensions and factors hierarchically and has the advantage of providing a selection mechanism that avoids overflow on the amount of information to be gathered. The framework provides information about software systems from three basic perspectives: the production process, the intrinsic attributes of software system and the organizational environment. The classification schema would be appropriate for identifying sets of similar projects and identifying the range of projects undertaken by an organization.

Even though the framework has been proposed to evaluate software systems, software is embedded in an environment (i.e., the organizational environment or a larger automated system). In many cases, it is difficult to establish the frontier between the software systems and the surrounding system, and to evaluate the software system exclusively. The project and system dimensions, as depicted in the framework, are directly related to evaluating software systems, whereas the environment dimension may involve evaluating the software system and its components which interact with the surrounding system to determine the contribution to the organization or larger automated system.

Metrics data can be classified consistently for comparison purposes using the perspectives of the framework. Homogeneous project data, derived from the framework, can be used for the analysis and evaluation of project results or for estimation purposes. Each project (i.e., development, correction, modification, enhancement) should register metrics information using the framework. Specific characteristics of each project are recorded to account for different circumstances during the life cycle of a system; a system which followed a rigorous development methodology may face more relaxed enhancement activities, or vice versa. A software system's internal characteristics help to explain its complexity. Finally, organizational environment considerations provide the system's contribution from an organizational perspective.

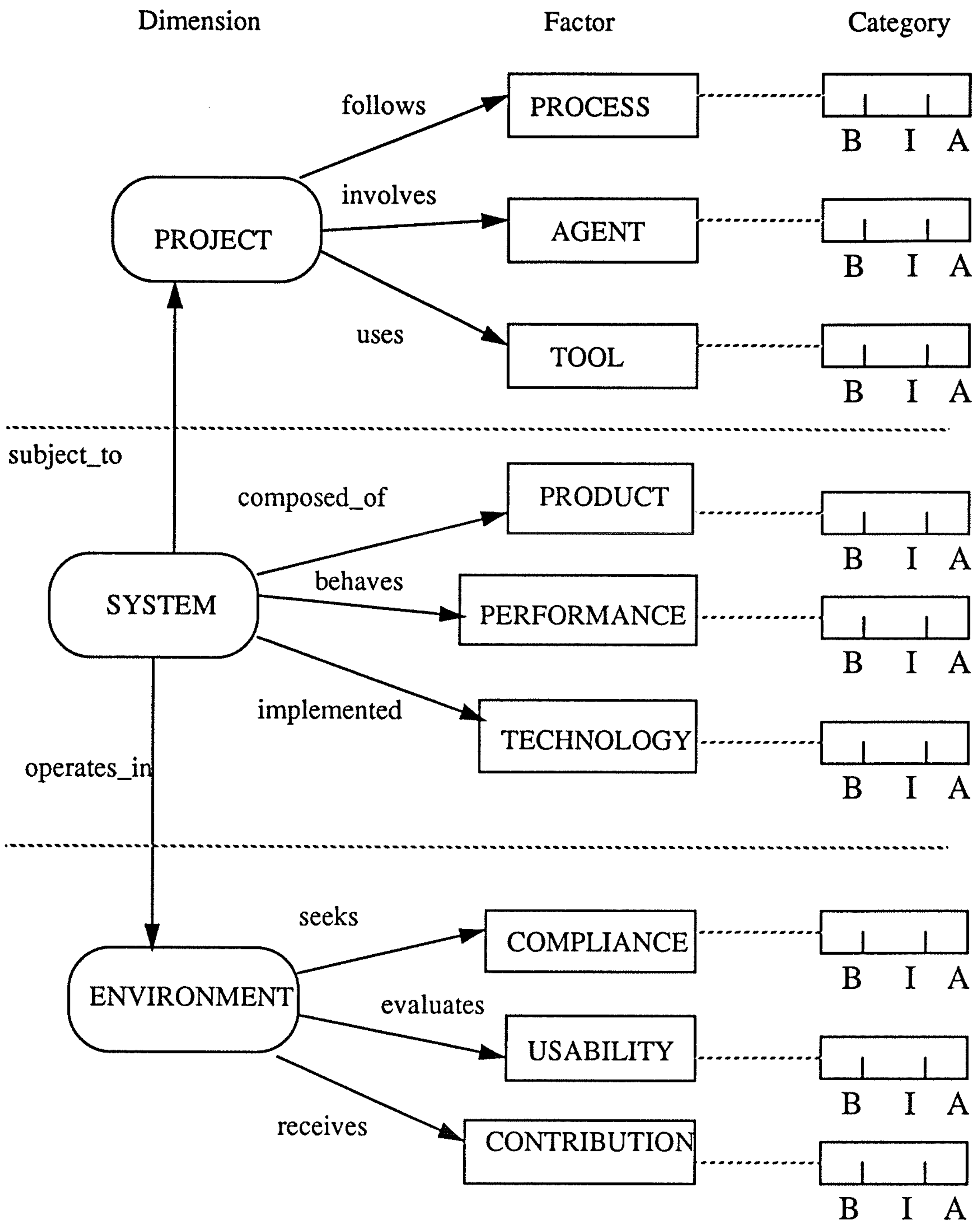
The article is organized as follows. Section 2 describes the software system evaluation framework and its levels of categorization. Section 3 analyzes examples of project classifications. Section 4 gives some conclusions and ideas for further research. Finally, the appendix presents current approaches related to our framework.

2.0 Evaluation Framework

This section presents, in a top-down manner, the dimensions of the framework, its decomposition into factors and the categorization of these factors for classification purposes. The terms 'software' and 'system' are used interchangeably throughout the text, even though when 'software' is used, it corresponds specifically to the target software system; when the term 'system' is used alone, it is because a wider perspective is required through human or automated interfacing and the contribution of the software system to the organization.

Evaluation is the process of judging the merits of a phenomenon (e.g., a system, a person, a thing, an idea) against some explicit or implicit yardstick. A summational evaluation is performed after the system has been completed, and provides information about the effectiveness of the system. Formative evaluation produces information that is fed back during the development of a system to help improve it; it serves the needs of producers. The framework identifies the main dimensions of software and supports both types of evaluation. Software evaluation stages follow the software life cycle: before the software is operational (i.e., development or enhancement), during the operation of the system (i.e.,

FIGURE 1. Categories in the framework



installation and operation), and after the system has reached steady operation (i.e., post-evaluation stage).

2.1 Model dimensions

The framework is organized along three dimensions corresponding to the software's producers, operators and users. The information about software systems captured in the framework is that of the software product, its production process and its end-impact on the organization. The viewpoints represented in the framework depict the project, the system and the environment.

The three dimensions in the framework are interconnected. A software *system* may be *subject_to* several *projects* during its lifetime: initial development and enhanced versions. A system *operates_in* an organizational *environment*: users interface with the system and services are provided by the system.

Figure 1 presents an entity-relationship diagram which further decomposes the framework into factors. A project *follows* a process, it *involves* some agents and *uses* some tools. The system is *composed_of* products, it *behaves* at some performance level and it is *implemented* in a particular technology. The environment *seeks* compliance with system requirements, it *evaluates* the usability of the system from the user's perspective and it *receives* a contribution or benefit from the operation of the system.

Each factor is categorized in the framework. Categories are useful for classifying information about software from a maturity perspective. To keep categories simple, only three ratings have been identified: basic (B), intermediate (I) and advanced (A). A basic category indicates the lowest maturity rating for a particular factor, an intermediate category may indicate a nominal rating or a standard in the industry, whereas an advanced category identifies a higher maturity rating which demonstrates excellence.

Validating framework completeness

To validate the completeness of the framework, we have chosen current productivity and quality models found in the literature. A complete summary of these approaches is presented in the appendix. The objective is to verify that the framework consolidates the different attributes found in the literature. The following summary presents current approaches versus the framework. The figures in Table 1 represent the number of attributes suggested by current approaches. For example, let us take the first column, WF-

Table 1: Current approaches versus Framework

	WF 77	BZ 78	B 81	M 81	SEI 88	CDS 86	RB 89	KMC 91	BBL 76	Mc 79	ISO 9126
Process	9	6	2	6	11	7	3	6			
Agent	5	2	5	4	2	1		24			
Tool	2	5	2	2		4		2			
Product	9	4	2	19	3	6	1		12	19	10
Performance	2	3	3	2		4			6	8	5
Technology		1	1	6	1	1					
Compliance	1				1	1			1	2	3
Usability	1			1					4	4	3
Contribution											
TOTAL	29	21	15	40	18	24	4	32	23	33	21

77. This is the Walston and Felix approach (see Appendix), which suggests 9 attributes related to the process factor, 5 related to the agent factor, and so on; the total number of attributes suggested by these authors was 29. The abbreviated names of columns correspond to the approaches analyzed and presented in the appendix.

The completeness of the framework has been validated against current approaches. Productivity models center their analysis on the project and system dimensions, whereas quality models emphasize the system and some usability factors from the environment. There is a lack of attributes for evaluating the contribution of the system to the organization and the lack of system post-evaluation appraisals. Contribution has indirectly been addressed in information system evaluation approaches and Total Quality Management aproaches which emphasize the contribution of systems to customers.

2.2 Framework description

The objective of the software system evaluation framework is to provide an assessment of the level of quality and sophistication of software from different points of view, those of producers, operators and users. Each major point of view may involve several roles (e.g., for producers: project managers, developers, maintainers and user representatives participating during the development of new versions of a system; for operators: technicians, administrators and managers; for users: those interacting directly with the system and stakeholders interested in the benefits provided by the system). We utilize the GQM paradigm[3] to define evaluation objectives for each element of the framework and establish questions for each factor category. The following paragraphs describe the characteristics of the factors for each dimension and the corresponding factor categories.

PROJECT

Projects follow specific organizational paradigms which define how working groups set priorities and deal with human issues. These include the aspect of how to manage extreme positions such as continuity and change, tradition and innovation, individual and group, and unity and diversity.[5] Well-defined project organizations with rigid lines of command and pre-established decision-making authority function differently from less formal organizations.

A software project may be oriented towards different types of activities (i.e., development or maintenance). Maintenance includes major changes implemented after the initial development, sometimes referred to as enhancements. Each change may be performed through different projects having different characteristics. Project information is relevant during the initial development of the system, and also later in its life cycle.

> The project dimension seeks to characterize project efficiency considerations (i.e., ability to develop a system without waste of time, energy, etc.) from the point of view of producers.

Process

The process which led to the development of the system, together with the design decisions and their justification, constitutes the backbone for improved quality and productivity. The life cycle model: (e.g., waterfall, prototyping, incremental) and its level of detail are aspects that determine project management style. Planning and control activities are better performed when precise process models are established. The more constrained the processes, the stricter the control has to be. SEI's assessment procedure indicates the level of maturity of the software organization and identifies the possible quality of the software being produced.

Techniques adopted to build the software determine the characteristics of the tangibles that will be produced. Well-documented standards determine the type of deliverables expected. Specification and design models specify their level of documentation detail; working products and deliverables have to be assessed on the required level of description overhead. System uniqueness and degree of application difficulty determine the level of test, debug and rework activities that must be allowed for in the chosen method. Additional considerations involve an assessment of whether or not the chosen technique fits the application (e.g., if data manipulation routines must support user interfaces, a technique suggesting the development of those routines prior to establishing user interfaces would not be appropriate).

Reuse of existing artifacts, such as specifications or code, is another factor to consider when characterizing the software production process. Those applications which are not unique or common are candidates for reusable code. The process has to be assessed whether it supports code reuse within the developed software and whether the process environment supports code reuse across projects.

The process factor seeks to determine the degree of efficiency and continuity of the process from the point of view of producers, basically process management.

Process categories

Basic: The project is characterized as one without a stable environment for producing software. Methodologies are adapted for each project and there is no follow-up of organizational learning from experience. Performance can only be predicted by individual, rather than project, capability.

Intermediate: The project follows a standard process for producing software. The software engineering and software management groups facilitate software process definition and improvement efforts. Projects use the organization-wide, standard software process to create their own defined software process which encompasses the unique characteristics of the project. Each project uses a peer review approach to enhance software product quality.

Advanced: The project sets quantitative quality goals for software products. Productivity and quality are measured for important software process activities. Software processes have been instrumented with well-defined and consistent measures which establish the quantitative foundation for evaluating project processes and software products.

Agent

The team of managers, engineers, systems analysts, programmers and users participating during development or enhancement is a major factor in system success; it is well known that the quality of resources has a major impact on project quality and productivity. Major aspects affecting agent assessment are experience, organization and motivation.

The experience of personnel involved during software development or enhancement may impact attributes of the software (e.g., quality, complexity). The important experience to retain is that of the application domain, technology and tools, software concepts and methods, and management issues evaluated from a technical and managerial perspective.

For categorization, software development organizations following well-defined lines of command and pre-established decision-making authority have to be differentiated from less formal organizations. Management involves leverage of extreme positions such as continuity and change, tradition and innovation, individual and group, and unity and diversity.

Motivation is also an important characteristic to retain, even if difficult to measure objectively. Lack of expert feedback on individual accomplishments and lack of understanding by the organization of needed individual career growth differentiates unsuccessful organizations from those which provide continual career growth opportunities. People are assigned to positions where they can use their skills and be highly productive, and frequent feedback is provided which is constructive and relevant to their individual skill levels.

The agent factor seeks to assess the team capability of people participating in the project from the point of view of producers, considering both managerial and technical aspects.

Agent categories

Basic: The team has limited experience in general, and no experience in the current type of project. People gain experience on their own and do not have a mentor structure to turn to. Training to gain or maintain skill levels is not available. Organizational structure does not support sharing of expertise across projects, if the expertise exists at all.

Intermediate: The team has experience in general, and limited experience in the current type of project. Few experts are available to provide advice when required. Training is available only on an ad-hoc basis, once urgent needs are identified.

Advanced: The team has the required experience for the project, and has already successfully demonstrated its capacity to undertake similar projects. A mentor system is in place to provide direction either within the project or across the organization. The organizational structure supports having personnel with specific experience available across projects as needed. Training is available to support skill growth at all experience levels.

Tool

Tools to aid in the production of software include available automated facilities that may impact productivity. Programming languages and database management systems may be supported by integrated software development tools. The importance of using tools during the early stages of software projects is widely recognized in the software community, though definite conclusions as to their impact are lacking. A tool is useless if no one can use it effectively. There must be training available, either formal in-house vendor courses or informal tutorials. Experienced personnel within the organization or from the vendor organization should be available on demand.

A framework which classifies these tools into *toolkits* that support only specific tasks in the software process (a basic programming toolkit is available for software production, i.e., compilers, debuggers and testers), *workbenches* that support only one or a few activities (CASE tools for software production) and *environments* that support (a large part of) the software process[6] (software development environments) is a convenient way to establish the level of sophistication of the tools.

The tool evaluation factor requires the establishment of the level of proficiency with the tools from the point of view of producers, regarding availability, training and support of technical activities.

Tool categories

Basic: Available tools are not supported by experienced personnel or tool vendors. No formal training is available, people learn on their own.

Intermediate: Tool support is limited because there is limited organizational experience or vendor support. Training is offered on a limited basis according to the urgency of the need.

Advanced: Tools are supported by experienced personnel within the organization or by tool vendors. Training is widely available to newcomers and frequent refresher sessions are offered to experienced personnel.

SYSTEM

There are several types of software systems widely known in the software community. Some examples are batch or interactive computing, real-time computing, fault-tolerant computing, high-safety and high-security computing, high-performance networks and multimedia technologies.

The system dimension is oriented towards evaluating intrinsic software attributes and the type of technology implementing the software, from the point of view of operators, administrators and managers.

Product

During the software life cycle, different types of documents are generated and referred to. The characteristics of these documents, including work products, deliverables and the final code, may be assessed on their quality. Documentation about the system in the form of user and system manuals, and at several layers of detail, from specifications and conceptual design to implementation, has to be evaluated. The quality of the documents, regarding conformance to established standards, has to be determined.

Size and complexity are useful attributes in comparing software systems. These attributes are normally derived from the source code, but other approaches exist to compute them from the specifications or the design. Size and complexity can be measured using function-oriented software metrics (e.g., function points) or implementation-oriented software metrics (e.g., lines of code and software science measures). The degree of reuse within the delivered software, as well as in other types of documentation (e.g., specifications, design) has to be determined.

The product factor requires an overall assessment of intrinsic software system attributes regarding software product understandability from the point of view of operators and system administrators.

Product categories

Basic: Understanding the software product is a major drawback of the system. Non-experts have to expend an excessive amount of effort to understand product functionality. There are no trustful documents available which describe the software system and its operation. The only source of documentation is the code itself.

Intermediate: Understanding software product functionality requires a moderate amount of effort for non-experts. Documents and code can be browsed to identify product functionality, but consistency among documents and code is not guaranteed. There are some documents available describing the software system and its operation, including development and enhancement documents, but they are not kept up to date.

Advanced: Understanding software product functionality requires minimum effort for non-experts. Supporting documentation is very well organized. Documentation describing the software system and its operation, including rationale during the stages of development and enhancement, are available for consultation. Careful updating is performed on the documents to keep them consistent.

Performance

Performance analysis is the measuring and modeling of the time and space attributes of the software system. Efficiency concerns the optimal use of computer resources to provide improved response time, storage and throughput. Reliability is the probability of operating in a satisfactory manner for a specified period of time, under specified operating conditions. Software failures (i.e., behavior of the executing program that does not meet the customer's operational requirements) and faults (i.e., defects in the code that may cause a failure) are important in establishing reliability. Performance in the framework consists of assessing computer resources usage from the perspective of operators, although recognizing that in the end performance has an impact on the user.

> The performance factor concerns the assessment of dynamic software characteristics, such as reliability and efficiency, from the point of view of operators and system administrators.

Performance categories

Basic: The use of computer resources is considered inadequate under standard operating conditions. Constant operator intervention is required to keep system operation. The effect of a system failure can demand large amounts of resources and it can involve a major inconvenience.

Intermediate: The use of computer resources is considered acceptable under standard operating conditions. Some operator intervention is required to keep system operation. The effect of a software failure is a situation from which operators can recover without severe penalty.

Advanced: The use of computer resources is considered excellent under standard operating conditions. Minimum operator intervention is required to keep system operation. The effect of a software failure is an easily recoverable loss which does not require excessive resources.

Technology

Operating system characteristics, as well as hardware characteristics, are related to technology. Single-user or multi-user computers indicate the potential use distribution. Open-system technology offers distributed computing through networking. Languages and database management systems also indicate the stage of evolution of the technology. The level of sophistication of the technology has a major impact on those operating the system.

The technology factor requires the establishment of the level of mastery of the technology implementing the system, from the point of view of operators and system administrators.

Technology categories

Basic: The implementation technology has not been mastered well by operators and administrators of the software system. There is no local experience with the technology and there is no good vendor support.

Intermediate: The implementation technology has only partially been mastered by operators and administrators of the software system. There is limited local experience or vendor support.

Advanced: The implementation technology has been thoroughly mastered by operators and administrators of the software system. Local experience and vendor support are continuously available.

It is convenient to precisely identify the language and DBMS implementing the system and the operating system supporting it. For a specific technology, a range of languages may be accessible to implement the system. Operating systems and database management systems are usually linked to a limited range of hardware technologies.

ENVIRONMENT

The system must provide benefits to the users if it is going to be used at all. Customer satisfaction should be the goal of any software production process. The characteristics of the interfaces with the environment are means for providing information about software system functionality and interaction with the organization. Applications can be oriented towards transaction processing, operation of process control systems, scientific computing, maintainance of the corporate database and decision-support systems. There are many different organizational environments associated with software applications, some are user-oriented (e.g., information systems), while others are machine-oriented (e.g., real-time systems). In any case, stakeholders have to assess their level of satisfaction with the software system.

The environment domain evaluates the level of satisfaction with the software system, as well as the perceived contribution of the system to the organization from the point of view of users and stakeholders.

Compliance

Compliance involves the overall functional evaluation of the system from the user's perspective. Important activities are to verify that user requirements are met by the system, to verify that the user's security and safety requirements are respected, and to verify that the system provides disaster protection.

The compliance factor assesses the conformance of the software system with its requirements from the user's point of view. It is the user's view of conformance with requirements rather than the developer's view of conformance with specifications.

Compliance categories

Basic: User requirements are not fulfilled by the system. Improvements to the system are required in the short term.

Intermediate: User requirements are partially fulfilled by the system. Some areas of the system have to be improved, even if it is not considered urgent by the users.

Advanced: User requirements are completely fulfilled by the system. The system is considered to be stable for the foreseeable future.

Usability

Characteristics of the usability of systems have traditionally been subjective. The user's appraisal of a system is based on different types of evaluations, including the effort required to learn and use it, and the assessment of user satisfaction. Suggestions in this area include the importance of objective quantification of usability.[7] Usability evaluates the effort to recognize logical concepts and their applicability, the effort required to learn the system, prepare input, interpret output, control the system, and the level of redundancy requested by the system. For some types of computing which do not emphasize user interfaces, such as real-time computing, an operator would be playing the role of user. The term 'user', in this context, depends on the nature of the system.

The usability factor is directed towards assessing software system learnability from the user's point of view.

Usability categories

Basic: The system is not providing interfaces in user-oriented terms. Redundant information is required from the users, making its use inefficient. Learning the system may require a great deal of time.

Intermediate: The system provides interfaces using terms familiar to the user, but there are some concerns (e.g., error messages are not clear and there are no shortcuts for experienced users). The system requires some time to learn, but this is considered acceptable by the users.

Advanced: The system is user-friendly, efficient to use, prevents errors by the user and clearly signals the seriousness of any errors. Infrequent users have the facility to return to using the system without having to relearn it, and frequent users find shortcuts that improve their efficiency. The system requires a short time to learn.

Contribution

The contribution of a system to the organization is a major indicator in evaluating a system. There are, however, serious difficulties as to how to quantify it because, besides economic benefits, there are intangible benefits which are difficult to measure.[8] The role of information technology in organizational activities and their impact on the cost structures of firms and markets have to be evaluated. However, systems have to be assessed in terms of a specific managerial context: business functions, market conditions, industry characteristics and organizational cultures all have an impact on the assessment. The impact on users and their jobs and the degree to which the information is adequate, appropriate, timely and current have to be determined, to ensure user satisfaction and monitor attitudes towards the system.

The evaluation of the contribution of the system to the organization assesses the benefits provided by the system to the organization, from the point of view of users and stakeholders.

Contribution categories

Basic: There is no major impact on the users and their jobs, and only marginal benefits on the user's productivity. The service provided by the system can be inappropriate and the information inaccurate or untimely. There is no major contribution to the organization because the business process has not been reengineered.

Intermediate: There is an impact on the users and their jobs. The service provided by the system is appropriate, but requires improvements in accuracy or timeliness. There are some intangible benefits to the organization regarding better service to its end-customers. Information is processed in a cost-effective way, improving the quality and speed of decision-making processes.

Advanced: The service provided by the system is excellent, the information adequate, current, and timely. There is a major positive impact on the users and their jobs. The system provides tangible benefits to the organization. The organization is made more cost-effective by using the system. Overall organizational costs have been reduced.

3.0 Applying the framework

The framework has been applied to evaluating several systems, and two of these evaluations are presented here. One of the experiments was to evaluate CASE tools, developed in the Software Engineering Lab at the Ecole Polytechnique de Montréal. Other experiments to validate the framework involve evaluations of information systems developed by a major consulting firm.

3.1 CASE tool evaluations

Examples of categorizations are presented in the following paragraphs which correspond to CASE tools developed over the last three years by the same team of producers. The evaluation was carried out by the same project manager who participated during the development of both systems and who also gets frequent feedback from users of the CASE tools. A summary of the results is presented in the following tables.

Structure Editor

A Structure Editor which assists programmers during algorithm design and code generation was evaluated following the framework. The tool is the result of many years of research in software engineering. We present the results of the evaluation for the latest version of this tool in Table 1, and an analysis of the results in the following paragraphs.

At the project dimension level, it is possible to identify, regarding the process factor (I), the fact that the process model is standard in the industry, follows structured techniques, and involves the reuse of software products; for the agent factor (I), that a small team of people follows a semi-formal project organization; for the tool factor (I), that commercial tools are available for development, including workbenches, and that the experience of the group is limited.

The system dimension portrays the software product (I) as a relatively small software system with a low degree of complexity, which is documented at several levels, including user and reference manuals; performance considerations (I) indicate a system with intermediate requirements in terms of reliability and efficiency, data can be recovered without severe consequences in the case of a software failure; technology (I) makes available an intermediate target language, in this case C, available in an open-system environment.

The environment dimension indicates satisfaction from the user's perspective. Compliance with requirements is high (A), as this version of the software is the result of several years of evolution. Usability is advanced (A), as the software targets professional programmers rather than novices. The contribution to the users is intermediate (I), as this software saves user time during algorithm development and maintenance.

Static Analyzer

A software quality assessment tool which analyzes source code, translates it into a formal representation of the source program, which is programming-language-independent, and generates software metrics was also evaluated. A precise computation of many traditional and innovative metrics is performed using this tool. See Table 2 for the evaluation results using the framework.

Table 2: CASE tool evaluation

		Structure Editor	Static Analyzer
	Process	I	I
PROJECT	Agent	I	I
	Tool	I	I
	Product	I	I
SYSTEM	Performance	I	A
	Technology	I	I
	Compliance	A	I
ENVIRONMENT	Usability	A	I
	Contribution	I	I

(B: basic, I: intermediate, A: advanced)

The project dimension in the Static Analyzer is similar to that in the Structure Editor (I, I, I). The process model, methods and tools were basically the same, although minor changes tended to improve the management of the process for the Static Analyzer. A record of changes in the process introduced during the development of this tool was kept. The individuals working on the two projects were different, their tasks overlapping only at the supervisory or managerial level, but for the purposes of the framework they were equivalent.

In terms of the system dimension, the Static Analyzer is equivalent to the Structure Editor as far as the product factor is concerned (I). Being a source code analyzer, it was developed with more stringent requirements in terms of efficiency (A). Regarding technology (I), this tool was partially built using C++ and open-system technology; people in the group are improving their mastery of this technology.

As for the environment dimension, there are slight variations in these CASE tools. For usability, the Static Analyzer is a research tool, thus a user interface oriented towards researchers (I); the Structure Editor is a commercial tool, thus oriented towards users (A).

For compliance, the Structure Editor has high conformance to requirements (A), whereas the Static Analyzer is an evolving tool (I). For contribution (I), the two CASE tools are considered equivalent.

Analyzing these two system categorizations, it is possible to conclude that the framework is sensitive to changes in their dimension factors; the same development organization was responsible for producing both systems, and the profiles are different.

3.2 Rule-based aggregation

Factors in the framework can be represented by an evaluation profile (e.g., a histogram) for purposes of human interpretation. However, it would be convenient to aggregate the data to extract details out for purposes of high-level management interpretation. Following a rule-based approach, it is possible to aggregate factor categories and assign a category at the dimension level. Table 3 shows the assignment of categories at the project dimension level. Rules have the following appearance:

IF *(Process* = 'intermediate') AND *(Agent* = 'basic') AND *(Tool* = 'intermediate'),

THEN *Project* = 'low'

This rule represents the assignment of row 5 in Table 3 (I - B - I) to the column 'low' in the project category. An equivalent table can be built for each of the other dimensions (i.e., system and environment). The assignment from factors to dimensions is carried out empirically, the importance of the factors is the same; process is as important as agent, which is as important as tools. The first column is determined with the following rationale: if at least two categories are Bs, then 'low'; if there are two Is and one B, then 'low'. The third column is determined as follows: if there are at least two As, then 'high'; if there are two Is and one A, then 'high'. The middle column contains the remaining possibilities.

Table 3: Assignment of categories at the project dimension level

Low	Medium	High
B - B - B	I - I - I	A - A - A
B - I - B	B - A - I	A - I - A
B - B - I	I - B - A	I - A - A
I - B - B	A - B - I	A - A - I
I - B - I	I - A - B	A - I - I
B - I - I	B - I - A	I - A - I
I - I - B	A - I - B	I - I - A
B - B - A	A - A - B	
B - A - B	B - A - A	
A - B - B	A - B - A	

(Process - Agent - Tool)

Table 4 shows the aggregated classification of the tools according to each dimension. It is clear from these results that the organization has developed the tools through equivalent projects. Differences in the evaluation appear at the system and environment dimension levels. At the system dimension level, the Static Analyzer has a higher evaluation because of the performance factor, which is advanced. At the environment dimension level, one tool provides more support from the user's perspective.

Table 4: Comparison of CASE tools

	Structure Editor	Static Analyzer
PROJECT	Medium	Medium
SYSTEM	Medium	High
ENVIRONMENT	High	Medium

Our experience in evaluating systems according to the framework demonstrates that an evaluation requires a short time to complete; an average of one and a half hours per software system is expected. Categories are presented in a way that minimizes the effect of subjectivity in the selection process, participants interacting with interviewers to choose one option.

4.0 Conclusions and directions for further research

We have proposed a software system evaluation framework which integrates in a hierarchy the important dimensions of a software system, its development environment and the organizational assessment of the system. Several factors which impact project productivity, software product quality and user satisfaction can be organized around these three perspectives of the framework, and which represent the producers' (i.e., development or enhancement), the operators' (i.e., the software product itself) and the users' (i.e., post-evaluation stage) view of the system. The framework is an original contribution to software engineering and the evaluation of software systems.

The framework is a mechanism to gain insight into, and an understanding of, software system quality and sophistication from a high-level perspective. The approach facilitates assessments by non-experts on software system evaluation, representing a compromise between huge amounts of software metrics detail and practical evaluation considerations. Detailed approaches, described in the appendix, identify large amounts of information which is difficult to grasp. To evaluate a software system, weeks or months are required. The framework can be used to evaluate a software system in a few hours and identify areas that require further investigation. The framework can be used for summational evaluations once the software system has been completed, as well as for formative evaluations during the construction of the system. Even before the system is built, goals for each factor in the framework can be established, such that actual results can be compared to goals.

Because this framework represents a model of reality, it has its limitations, however, and avenues for extendibility can be identified by following the approach appropriately. The evaluation categories suggested in the framework are oriented towards assessing each factor globally, using an evolving maturity scale. It is clear that additional subfactors can be established for each factor, which can be categorized by objective or subjective evaluations. Each organization can identify these additional attributes using the framework as a baseline. In fact, experience in applying the framework to identifying a set of metrics for estimation purposes in industry indicates the usefulness of the framework: elimination of redundant metrics, assignment of metrics to the appropriate factors, analyzing composite metrics and understanding the nature of each measurement.

In some organizations there may be a need for more formal approaches to aggregation. A possible alternative would be to define a quantitative model to aggregate data, taken from the decision sciences, using a different scale (e.g., a ratio scale) and considering the relative importance among elements in the framework. A computational method is being analyzed for such purposes, based on a decision model.[9] The approach quantifies the relative importance of elements in the framework at each level in the hierarchy. Normalized indices for each dimension are obtained which indicate the level of quality and sophistication of the software system.

The software system evaluation framework acknowledges current contributions of productivity and quality models, and the assessment of software systems suggested in evaluation approaches. It has been determined that current approaches to productivity and quality do not consider the contribution of systems to the organization, however this aspect has been considered in information system evaluations and total quality management approaches. While customer satisfaction issues and the social impact of computing have been the subject of research lately[10], metrics which consider the overall benefits of a system to an organization are currently needed. Cost/benefit analyses of operational systems, as well as software quality evaluations involving user satisfaction, should be included in organizational evaluation programs.

Further research to evaluate systems from historical metrics data automatically is deemed important. The availability of subjective and objective metrics for each factor in the framework requires cohesive approaches for their integration. As the amount of information being gathered for a system increases, mechanisms have to be devised to determine commonality among metric data. The goal would be to attach an evaluation to the categories of the framework automatically, thereby minimizing evaluator judgments.

Defining a comprehensive software system evaluation framework to be shared among organizations is a step towards improvement-oriented software engineering. According to guidelines for implementing metrics programs in organizations, the initial stage requires the establishment of a subset of metrics to demonstrate its feasibility. Our research is a top-down categorization of important factors to be considered for an organizational metrics program. This model can be allowed to evolve in order to determine a basic set of metrics that could be used to exchange data among organizations. Useful analysis and comparisons can be performed when it is possible to share experiences based on this framework.

There are several possible applications of the framework. For evaluation purposes, it allows software system evaluation data to be identified uniformly for classification and comparison purposes. It can be used as a baseline to establish metrics programs in organizations. It can be used to validate current metrics in an organization. It can be useful in the selection of metrics for specific purposes, such as estimation or requirements definition. A taxonomy of software system characteristics can be deduced from the framework to determine homogeneous clusters of metrics data for comparison purposes. Several activities in software engineering, such as planning and estimation, require expert opinion to suggest the characteristics of software systems. Current research involves identifying and proposing a set of estimation metrics for an organization using the framework.

Acknowledgements

The authors would like to thank the anonymous referees for their valuable comments. The contribution of senior analysts from Groupe DMR who collaborated to test the framework with real cases is greatly appreciated. The contribution of Andre Beaucage, Project Manager at the Software Engineering Lab in evaluating CASE tools is also appreciated.

References

1 Kumar, K. 'Post Implementation Evaluation of Computer-Based Information System: Current Practices', Communications of the ACM, Vol. 33, No. 2, February 1990, pp. 203-212.

2 Chandler, J.S. 'A Multiple Criteria Approach for Evaluating Information Systems', MIS Quarterly, March 1982, pp. 61-74.

3 Basili, V.R.; Rombach, H.D. 'The TAME Project: Towards Improvement-Oriented Software Environments', IEEE Transactions on Software Engineering, Vol. 14, No. 6, June 1988, pp. 758-773.

4 Mylopoulus, J. 'Conceptual Modeling and Telos', in *Conceptual Modeling, Databases and CASE*, Loucopoulos, P.; Zicari, R. (editors), Wiley, 1992.

5 Constantine, L.L. 'Work Organization: Paradigms for Project Management and Organization', CACM, Vol. 36, No. 10, October 1993, pp. 34-43.

6 Fuggetta, A. 'A Classification of CASE Technology', IEEE Computer, Vol. 26, No. 12, December 1993, pp. 25-38.

7 Nielsen, J. *Usability Engineering*, Academic Press, Inc. 1993.

8 Brynjolfsson, E. 'The Productivity Paradox of Information Technology', Communications of ACM, Vol. 36, No. 12, December 1993, pp. 66-77.

9 Saaty, T.L. *The Analytic Hierarchy Process*, McGraw-Hill, Inc. 1980.

10 Clement, A. 'Computing at Work: Empowering Action by Low-level Users', CACM Vol. 37, No. 1, January 1994, pp. 53-63.

Appendix

Current approaches

Approaches which suggest important factors to consider in software measurement include productivity and quality models. Productivity models surveyed are Walston et al., productivity factors[a1]; Basili et al., factors affecting software development[a2] ; Boehm's Software Development Modes and Cost Drivers[a3]; Mohanty's Software Cost Estimation Factors[a4]; Conte et al., factors affecting productivity[a5]; Ramsey et al., homogeneous projects[a6]; and Kemayel et al., factors for programmer productivity[a7]. Software quality models surveyed are those suggested by Boehm et al.,[a8] and McCall[a9]. Also, SEI's Capability Maturity Model[a10] for process assessment and the ISO 9126 standard for software quality. Total quality approaches include the Malcolm Baldrige[a11] and European Quality Awards.[a12]

Productivity models center their analysis in the project and system dimensions, whereas software quality models emphasize the system and some usability factors from the environment dimension. Quality Awards are more organization-oriented than software-oriented, providing a wider view of quality concerns.

WF-77. Walston and Felix productivity factors

The objective of this research was to search for a method of estimating programming productivity. Twenty-nine factors which correlate with programming productivity were identified. These were related to complexity, user participation, personnel experience and qualifications, staff size/duration, programming techniques, constraints on the programs, type of application, database classes of items, and pages of documentation.

BZ-78. Basili and Zelkowitz factors on software development

Data from several projects were collected at the Software Engineering Laboratory (from NASA's Goddard Space Flight Center and the University of Maryland) to evaluate software engineering methodologies. For each project, a set of factors affecting software development was gathered: people factors, problem factors, process factors, product factors, resource factors and tools.

B-81. Boehm's Software Development Modes and Cost Drivers

Boehm distinguishes three modes of software development: organic, semi-detached and embedded. The important features that allow the identification of Software Development Modes are: organizational understanding of product objectives; experience in working with related software systems; conformance with pre-established requirements; conformance with external interface specifications; concurrent development of associated new hardware and operational procedures; innovative data processing architectures, algorithms; a premium on early completion; and product size range. Boehm also proposed several cost drivers for estimation purposes, organized by product attributes, computer attributes, personnel attributes and project attributes.

M-81. Mohanty's Software Cost Estimation factors

Mohanty identified the significant factors considered by various model builders in the literature. These factors were organized by system size (i.e., number of instructions), database (i.e., number of words in the database), system complexity (e.g., system uniqueness, complexity of interfaces and program structure), type of program (i.e., type of application), documentation (i.e., number of pages of documentation), environment (e.g., development environment, languages, computer speed and memory capacity, productivity), and other factors (e.g., safety considerations, system growth requirements).

CDS-86. Conte, Dunsmore and Shen: factors on productivity

There are many factors that appear to affect the software development process and the product. Some of the factors that affecting productivity are related to people factors (e.g., individual capability, size of the team, language experience), process factors (e.g., establishment of milestones, development schedule, use of specification techniques or methods), product factors (e.g., size of product, type of program structures, amount of reused code) and computer factors (e.g., response time, turnaround time, storage constraints).

RB-89. Ramsey and Basili: homogeneous environment

The software which provided the data for Ramsey and Basili's study was developed at the NASA Goddard Space Flight Center. The authors claim that the software development environment was homogeneous, i.e., many similar projects are developed for the same application area. Additional considerations for homogeneous environments are: a standard process model, a software development methodology that is similar across projects, and a great deal of reuse of code from previous projects.

KMO-91. Kemayel, Mili, and Ouederni: programmer productivity factors

In a survey study, the authors suggested 33 controllable factors of programmer productivity. These factors are related to personnel, the software process and the user community. Personnel includes motivation (e.g., nature of work, level of responsibility, salary) and experience (e.g., experience on application domain, programming language and user community). Process factors include project management (e.g., adherence to a software life cycle, use of cost estimation procedures) and programming environment (e.g., use and power of programming tools, modern programming practices and power of the equipment).

BBL-76. Boehm, Brown, and Lipow: software quality factors

A set of important software characteristics related to quality have been proposed. Metrics to assess the degree to which the software has the defined characteristics were developed and correlated with the characteristics. Refinements were made on the set of characteristics to produce a set that supports software quality evaluation. Relationships were established between characteristics and refined characteristics. Finally, the metrics themselves were refined. The list of characteristics includes: portability, reliability, efficiency, human engineering, testability, understandability, modifiability and maintainability.

Mc-79. MacCall: software quality criteria

Eleven quality factors were proposed and grouped according to three orientations or viewpoints (i.e., product operation, product revision and product transition). The factors are conditions or characteristics that actively contribute to the quality of the software. They represent a management-oriented view of software quality. To introduce a dimension of quantification, this management orientation must be translated into a software-related viewpoint. This is accomplished by defining a set of criteria for each factor. The criteria are independent attributes of the software, or of the software production process, by which the quality can be judged, defined and measured. Finally, quality metrics can be established to provide a quantitative measure of the attributes represented by the criteria. The list of quality factors includes: correctness, reliability, efficiency, integrity, usability, maintainability, testability, flexibility, portability, reusability and interoperability.

SEI-88. SEI's Capability Maturity Model

The CMM is designed to provide guidance to control the software production process and to evolve towards software excellence. The model identifies the current process maturity level of an organization and the issues most critical to software quality and process improvement. Key Process Areas are defined by Maturity Level. Level 1 (Initial) has no key process areas. Level 2 (Repeatable) has the following key process areas: requirements management, software project planning, software project tracking and oversight, software subcontract management, software quality assurance and software configuration managment. Level 3 (Defined) includes: organization process focus, organization process definition, training program, integrated software management, software product engineering, intergroup coordination and peer reviews. Level 4 (Managed) includes: quantitative process management and software quality management. Finally, level 5 (Optimizing) includes: defect prevention, technology change management and process change management.

ISO-9126. Software product evaluation

The International Standard ISO 9126 'Information technology - Software product evaluation - Quality characteristics and guidelines for their use' presents a list of software quality characteristics for evaluating the quality of a software product. It includes functionality, reliability, usability, efficiency, maintainability and portability.

Quality Awards

Awards promote awareness of quality as an increasingly important element in competitiveness, in understanding the requirements for performance excellence, and in sharing information on successful performance strategies and the benefits derived from implementation of these strategies. Even though the awards are not directly oriented towards software systems, they stress the importance of customer satisfaction, which is an indicator of benefits to the organization.

Malcolm Baldrige National Quality Awards

The awards are made annually to recognize companies for business excellence and quality achievement. Awards may be given in each of three eligibility categories: manufacturing companies, service companies and small businesses.

Leadership: The senior executives' success in creating and sustaining a quality culture.

Information and Analysis: The effectiveness of information collection and analysis in maintaining a customer focus, driving quality excellence and achieve excellence.

Strategic Quality Planning: The effectiveness of integrating quality requirements into business plans.

Human Resource Development and Management: The success of efforts to realize the full potential of the work force to meet a company's quality and performance objectives.

Management of Process Quality: The effectiveness of systems and processes for ensuring the quality of products and services.

Quality and Operational Results: Improvement in quality and operational performance, and supplier quality, as demonstrated through quantitative measures.

Customer Focus and Satisfaction: The effectiveness of systems to determine customer requirements and satisfaction, and the demonstrated success in meeting customers' expectations.

The European Quality Award

Quality encompasses the activities that organizations perform to meet the needs and expectations of its custormers, its people, its financial stakeholders and society at large. Quality has already become the competitive edge.

ENABLERS

1. Leadership: The behavior of all managers in driving the organization towards Total Quality

2. Policy and Strategy: The organization's mission, values and strategic direction and the manner in which it achieves them

3. People Management: The management of the organization's people

4. Resources: The management, utilization and preservation of resources

5. Processes: The management of all value-adding activities within the organization

RESULTS

6. Customer Satisfaction: What the perception of your external customers is of the organization and of its products and services

7. People Satisfaction: What your people's feelings are about their organization

8. Impact on Society: What the perception of your company is in the community at large. This includes views of the company's approach to quality of life, the environment and to the preservation of global resources

9. Business Results: What the organization is achieving in relation to its planned business performance

References

a1 [WF-77] Walston, C.E.; Felix, C.P. 'A method of programming measurement and estimation', IBM Systems Journal, No. 1, 1977, pp. 54-73.

a2 [BZ-78] Basili, V.R.; Zelkowitz, M.V. 'Analyzing Medium-scale Software Development', 3rd. International Conference on Software Engineering, May 10-12, 1978, Atlanta, Georgia, USA.

a3 [B81] Boehm, B. *Software Engineering Economics*, Prentice-Hall, Inc., 1981.

a4 [M81] Mohanty S.N. 'Software Cost Estimation: Present and Future', Software Practice and Experience, Vol. 11, 1981, pp. 103-121.

a5 [CDS86] Conte, S.D.; Dunsmore, H.E.; Shen, V.Y. *Software Engineering Metrics and Models*, The Benjamin / Cummins Publishing Company, Inc. 1986.

a6 [RB-89] Ramsey C.L.; Basili V.R. 'An Evaluation of Expert Systems for Software Engineering Management', IEEE Transactions on Software Engineering, Vol. SE-15, No. 6, June 1989, pp. 747-759.

a7 [KMO-91] Kemayel, L.; Mili, A.; Ouederni, I. 'Controllable Factors for Programmer Productivity: A Statistical Study', Journal of Systems and Software, 1991; 16, pp.151-163

a8 [BBL-76] Boehm, B.W.; Brown, J.R.; Lipow, M. 'Quantitative Evaluation of Software Quality', International Conference on Software Engineering, 1976, pp.592-605.

a9 [Mc79] McCall, J.A. 'An Introduction to Software Quality Metrics', in *Software Quality Management*, Cooper, J.D. and Fisher, M.J., editors. Petrocelly Books, Inc. 1979, pp.127-142.

a10 [SEI-88] Humphrey, W.S. 'Characterizing the Software Process: A Maturity Framework', IEEE Software, March 1988, pp. 73-79.

a11 European Quality Award, Self-assessment based on the European Model for Total Quality Management 1994, The European Foundation for Quality Management, 1994.

a12 Malcolm Baldrige National Quality Award, 1994 Award Criteria, United States Department of Commerce, Technology Administration, National Institute of Standards and Technology, 1994.