PolyPublie
Polytechnique Montréal

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Titre: Comprehensive software metrics framework
Title:

Auteurs: Germinal Boloix, & Pierre N. Robillard
Authors:

Date: 1994

Type: Rapport / Report

Référence: Boloix, G., & Robillard, P. N. (1994). Comprehensive software metrics framework. (Rapport technique n° EPM-RT-94-07). https://publications.polymtl.ca/9587/
Citation:

Document en libre accès dans PolyPublie
Open Access document in PolyPublie

URL de PolyPublie: https://publications.polymtl.ca/9587/
PolyPublie URL:

Version: Version officielle de l'éditeur / Published version

Conditions d'utilisation: Tous droits réservés / All rights reserved
Terms of Use:

Document publié chez l'éditeur officiel
Document issued by the official publisher

Institution: École Polytechnique de Montréal

Numéro de rapport: EPM-RT-94-07
Report number:

URL officiel:
Official URL:

Mention légale:
Legal notice:

Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal

https://publications.polymtl.ca

0 7 AVR. 1994

EPM/RT-94/07

Comprehensive Software Metrics Framework

Germinal Boloix

Pierre N. Robillard

Département de génie électrique
et génie informatique

École Polytechnique de Montréal
Mars 1994

Tous droits réservés. On ne peut reproduire ni diffuser aucune partie du présent ouvrage, sous quelque forme que ce soit, sans avoir obtenu au préalable l'autorisation de l'auteur,
OU des auteurs

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

Compter 0.10 $ par page et ajouter 3,00 $ pour la couverture, les frais de poste et la manutention. Régler en dollars canadiens par chèque ou mandat-poste au nom de l'École Polytechnique de Montréal.

Nous n'honorerons que les commandes accompagnées d'un paiement, sauf s'il y a eu entente préalable dans le cas d'établissements d'enseignement, de sociétés ou d'organismes canadiens.

Comprehensive Software Metrics Framework

Germinal Boloix, Pierre N. Robillard
8 March 1994
Ecole 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

We propose a comprehensive software metrics framework to guide organizations in classifying software metrics data. The framework facilitates identifying homogeneous metrics data for comparison purposes. It can be used in the establishment of metrics programs in organizations. The framework integrates system's knowledge into three perspectives: the software production project, the characteristics of the system, and its organizational environment. The framework takes in consideration ancient studies on productivity models and software quality factors, introducing an important dimension that characterizes the contribution of the system to the organization.

The framework perspectives represent the before (i.e., development or enhancement), during (i.e., the product itself), and after (i.e., post-evaluation stage) views of systems. The project dimension considers the characteristics of the agents, the process and the tools during software production. The system dimension depicts the characteristics of the product, its technology, and its performance. Finally, the organizational dimension that identifies the domain of the problem, introduces factors related to compliance of the system meeting its requirements, the usability of the system from the user's perspective, and the contribution of the system to the organization.

We expect the framework to help practitionners in sharing and extrapolating experiences consistently inside their organization and among different organizations. Researchers can benefit from the framework when using real data from industrial sites in their studies. Improvements in software engineering practice are expected by sharing a common framework based on a comprehensive analysis of the software, its construction, and its contribution.

Keywords: software metrics, metrics framework, system benefits, homogeneous metrics data, productivity factors, quality factors

1.0 Introduction

Software metrics represent a useful resource to assess the characteristics of software and the software production process. Software metrics provide supporting aids for more accurate estimation of project milestones, monitoring of project progress, and evaluation of project results. Estimation using analogy from previous experiences is improved with trustful historical metrics. Project control activities may detect abnormal trends monitoring software metrics. Project evaluations may recommend changes to the process model, to the software development methodology, to the type of personnel involved and their training, to improve productivity.

There are several factors that affect software projects data. Productivity of software development or maintenance is a familiar concern that affects the cost of software; personnel considerations are paramount in the cost of a system. Another concern is the complexity of software; characteristics of the software determine how easy it is to produce, impacting also software cost. Another aspect that has not received adecuate attention is how human and organizational productivity are directly affected by the use of computer systems [GBL91]; it is important to evaluate the contribution of a system to the organization. However, there are so many factors affecting the software and its production process that a systematic approach to select those factors is required.

Organizations have gathered software metrics data for years following undefined frameworks; that data has not been carefully characterized for future use. Historical project data is useful to compare projects of similar size, application domain, and complexity. Reasons behind lack of confidence on historical software metrics data collected in industries lie on inappropriate categorization of project attributes due to unclear measurement objectives. Statistics from completed projects can be helpful to determine correlations among several variables such as size, complexity, and quality, versus effort. When data like effort is gathered, there are many parameters that influence its relationship to software productivity (e.g., personnel experience, software development methods and tools, type of software application, and so on). If such data is not well characterized, through a software metrics framework that categorizes the data, it is not possible to compare projects or systems consistently, and it is less reliable to make good use of the data for estimation purposes.

Several researchers have proposed the use of metric data collected from industry to perform multivariables studies. Most of this type of research has been done by correlating product metrics with effort; these research results can be misleading. Without characterizing the environment surrounding the data, parameters like effort of development or maintenance are not trustful for correlation purposes. One way of overcoming this problem is by documenting the conditions under which the system was produced, the characteristics of the software itself, and the results of using the software, such that similar projects be identified for statistical analysis studies. Homogeneity surrounding metrics data is required to make meaningful comparisons.

The benefits of operating a system in an organization is an area of concern for any business. Productivity is the fundamental economic measure of a technology's contribu-

tion. Delivered computing power in the U.S. economy has increased by more than two orders of magnitude since 1970 yet productivity, especially in the service sector, seems to have stagnated [B93]. There is a lack of metrics for usability and productivity of people and organizations who use computers. There are almost no quantitative behavioral measures of general trends, over the years, on how human and organizational productivity are directly affected by the use of computer systems [GBL91].

The importance of different types of knowledge to be gathered during software projects is recognized in conceptual modeling. Mylopoulus [LZ91] suggested the important types of knowledge to gather for a system in what was called 'worlds'. The usage world keeps 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.

In an effort to establish an homogeneous estimation base, Boehm [B81] suggested three basic modes of development for software projects: organic, semidetached and embedded; several features that permit classifying these modes, and cost drivers affectting project estimation were also proposed. Other authors have suggested additional productivity factors [WF77], [BZ78], [M81], [RB89], [CDS86], [KMO91]. In general, these productivity approaches emphasize product, personnel, project, and computer attributes. Software quality has also been subject to analysis, pioneer work by Boehm et al., [BBL76] and McCall [M79] suggest the importance of software quality characteristics. SEI's Capability Maturity Model (CMM) has been suggested to improve current software production practices, it assesses the software process maturity level of organizations [H88]. Other important factors that affect projects such as application domain, software technology and people experience are not directly considered in CMM [PCC93].

Internal software characteristics also have an impact on productivity. Business applications (e.g., transaction-oriented) differ from scientific applications (e.g., mathematical, simulation). Real time process control systems also differ from database applications. Even within a particular type of application there are differences on emphasis. Some applications are more user interaction oriented rather than computation oriented, while others are more database oriented or decision support oriented.

Our objective is to develop a comprehensive software metrics framework that facilitates the selection of a basic set of attributes to characterize software projects. There are many factors affecting the productivity of projects, the quality of software, and the benefits obtained from systems operation. A top-down approach, as the one suggested in this paper, has the advantage of providing a metrics selection mechanism that avoids overflow on the number of metrics to be gathered. The framework provides a hierarchical classification of metrics data through three basic perspectives: the production process, the internal characteristics of software, and the contribution of the software to the organization. The framework can be used to classify software projects data for analysis purposes. It can be used to identify a set of attributes required for estimation purposes. It can also be used as the baseline to establish what type of metrics data to gather in an organization.

The article is organized as follows. Section 2 presents current approaches to characterize software projects, see also the appendix. Section 3 describes the comprehensive software metrics framework and its levels of categorization. Section 4 presents current approaches versus our framework. Section 5 presents an example project classification. Finally, section 6 gives some conclusions and further research.

2.0 Current approaches

There are several models suggested in the literature that allow categorization of projects, quality, and productivity factors. Productivity models include Walston et al., productivity factors [WF77]; Basili et al., factors affecting software development [BZ78]; Boehm's Software Development Modes and Cost Drivers [B81]; Mohanty's Software Cost Estimation factors [M81]; SEI's Capability Maturity Model [H88]; Conte et al., factors affecting productivity [CDS86]; Ramsey et al., homogeneous projects [RB89]; Kemayel et al., factors for programmer productivity [KMO91]. Software quality models include those suggested by Boehm et al., [BBL76] and McCall [M79]. See the appendix for additional details on each approach.

Walston and Felix productivity factors

The objective of this research was to search for a method of estimating programming productivity [WF77]. Twenty-nine factors that correlate with programming productivity were identified, related to complexity, user participation, personnel experience and qualifications, staff size/duration, programming techniques, constraints on the programs, type of application, data base classes of items, and pages of documentation.

Basili and Zelkowitz factors on software development

Data from several projects was collected at the Software Engineering Laboratory (NASA Goddard Space Flight Center and the University of Mariland) to evaluate software engineering methodologies [BZ78]. For each project, a set of factors that affect software development were gathered: people factors, problem factors, process factors, product factors, resource factors, and tools.

Boehm's Software Development Modes and Cost Drivers

Boehm [B81] distinguishes three modes of software development: organic, semidetached 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; premium on early completion; product size range. Boehm also proposed several cost drivers for estimation purposes, orga-

nized by product attributes, computer attributes, personnel attributes, and project attributes.

Mohanty's Software Cost Estimation factors

Mohanty {M81] identifed the significant factors that have been considered by various model builders in the literature. Factors were organized by system size, data base, system complexity, type of program, documentation, environment, and other factors.

SEI's Capability Maturity Model

The CMM [H88] is designed to provide guidance to control the software production process and to evolve towards software excellence. The model indentifies the current process maturity level of an organization and the most critical issues to software quality and process improvement.

Conte, Dunsmore and Shen factors on productivity

There are many factors that appear to affect the software development process and the product [CDS86]. Some of the factors that affect productivity are related to people factors, process factors, product factors, and computer factors.

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 [RB89]. The authors claim 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, software development methodology is similar across projects, and a great deal of reuse of code from prior projects.

Kemayel, Mili, and Ouederni programmer productivity factors

In a survey study the authors suggested 33 controllable factors of programmer productivity [KMO91]. These factors are related to personnel, the software process and the user community.

Boehm, Brown, and Lipow software quality factors

A set of important software characteristics related to quality were proposed [BBL76]. Metrics to assess the degree to which the software has the defined characteristic were developed and correlated with the characteristics. Refinements were performed to the set of characteristics into a set that supports software quality evaluation. Relationships were established among characteristics and refined characteristics. Finally, the metrics were also refined.

MacCall software quality criteria

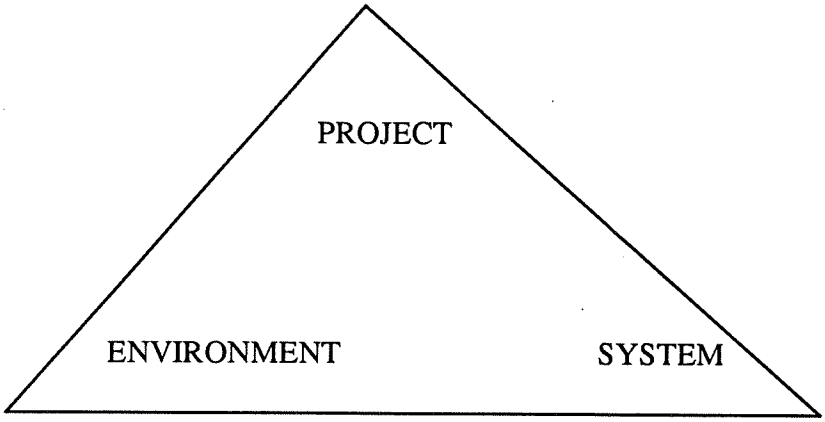
Eleven quality factors were proposed, grouped according to three orientations or viewpoints (i.e., product operation, product revision and product transition) [M79]. The factors are conditions or characteristics which actively contribute to the quality of the software. Factors represent a management-oriented 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 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.

After analysing these productivity models suggested in the literature, it is possible to identify the main areas that impact project productivity. Project personnel has definitely a major impact, even though it is difficult to quantify because of subjective criteria; personnel includes users, managers, analyst and programmers involved in the software production process. The process model and the development methodology define the software production environment; large projects usually follow a strict process that generates prespecified deliverables, requiring greater overhead than smaller projects; availability of reusable parts improves the productivity of the process. The structure of the software also has an impact on productivity; software architecture determines the general decomposition criteria. Structuring processing towards input, output or interactive applications is a source of product differences. Finally, the type of technology available for software production (e.g., CASE tools, project control tools) and system implementation (i.e., target technology), determines the level of automation of the system and its production process.

3.0 Metrics Framework

A comprehensive software metrics framework's purpose is to organize the important dimensions impacting software project data. Software is more than its internal characteristics and the production process that creates it, software provides a service to customers. Using the framework, metrics data can be classified consistently, using the three perspectives (i.e., project, system, environment), for comparison purposes. Homogeneous project data, derived from the framework, can be used for analysis or estimation purposes. Each development or evolution project (e.g., new, correction, modification, enhancement) should register metrics information following the framework. Specific characteristics of each project are recorded, to account for different circumstances during the life cycle of a system; a system that followed a rigourous development methodology, may face more relaxed maintenance activities, or vice versa. System internal characteristics help to understand its complexity. Finally, organizational environment considerations provide the system's contribution from the user perspective. Figure 1 shows the three perspectives of the framework.

FIGURE 1. Metrics Framework Dimensions



3.1 Elements of the framework

The comprehensive software metrics framework considers not only the process to produce the software and the characteristics of the software itself, but also introduces the importance of the organizational environment where the software is used. It includes important factors to characterize projects and integrates several variables affecting software metrics data into the following dimensions: project, system, and environment. Each dimension introduces a series of factors that have to be assessed according to different levels of categorization.

PROJECT

Project organizations use to carry out activities in coordinated ways. Project teams differ in how they operate as well as in their underlying observable operation. Projects can be coordinated by a traditional hierarchy of authority, by reliance on individual initiative, by collaborative discussion and negotiation, or by virtue of alignment with a common vision or direction [C93].

The process that led to the development of the system, together with the design decisions and their justification, constitutes the backbone to improved quality and productivity. The methodology adopted to build the system determines the characteristics of the tangibles to produce. The team of system analysts and programmers participating during development or maintenance is a major factor for successful systems. Tools to aid during the production of software acknowledge the available automatic facilities. All this knowledge is relevant during the initial development of the system, and also later on during its evolution.

The type of project identifies the main orientation of the activities (i.e., development or maintenance). A system may be subject to several changes after development (e.g., correction, modification, enhancement) during its lifetime; each of those tasks is performed by different projects, each with different characteristics.

Process

- Process model: the type of process model (e.g., waterfall, prototyping, incremental) and its level of description formality are factors that determine project management success. Planning and control activities are better performed when precise process models are established. Considerations about how conformance of requirements would be handled, should be defined when establishing the process model, as rework activities impact productivity. Flexibility of the process model also determines its applicability under different circumstances; constrained processes require a more severe style of control than those more informally oriented.
- Methods: well documented guidelines determine the type of aids available to the developer and the expected deliverables. Step by step guidelines documenting precisely the products to generate have to be assessed; the degree of tailoring flexibility to adapt guidelines to specific cases has to be evaluated. Artifacts' notation expressing product models indicate their level of formality; working products and deliverables have to be assessed for their formality.
- Innovative applications have to be differentiated from rutine applications, the former requires much more effort through trial and error, and more rework activities because of their level of uncertainty. System uniqueness and its degree of difficulty affect productivity of the process.
- Reuse of existing artifacts such as specifications or code is another factor to consider when characterizing the software development environment. Development of systems using libraries of components is significantly different than building systems anew.

To characterize the process model, an outside process assessment facility may be required. SEI's assessment procedure indicates the level of maturity of the organization, and provides more background to the possible quality of the software being produced. Boehm's software development modes is another way to characterize the software process environment.

Agents

Team size is a factor that affect productivity because of group dynamics. The level of formality of the project organization has to be determined. Well defined project organizations with rigid command lines and pre-established authority decision points, have to be differentiated from informal organizations; autocratic organizations also differ from democratic styles of project management. Experience of personnel involved during software development may impact attributes of the software (e.g., quality, complexity); it is well recognized that the quality of resources has a major impact on project quality and productivity. Motivation is also an important characteristic to retain, even if difficult to measure objectively.

The experience of personnel involved during software production relate to:

- Application domain: knowledge of the problem domain by previous experience or by participating in other projects related to the same domain. Managers, analysts and users' experience in the application domain have a definite impact on productivity.
- Technology and tools: target technology experience (e.g., real time, distributed applications); programming language experience; operating system experience. Experience using software tools facilitates production work. Designers and programmers are primarily affected by technology considerations. Managers and users should possess a basic knowledge of software systems.
- Methods: familiarity with the software production method; degree of expertise with the techniques; availability of standards and guidelines. Managers require a general understanding of the methods, whereas producers require a solid understanding of the techniques.
- Project control: communication and project management skills may impact productivity. Previous experience working on teams and familiarity with the team may improve productivity. Availability of software metrics to compare with normal trends help controlling the project. Management requires to master project control skills.

Tool

The importance of using tools in software projects is well recognized by the software community, though definite conclusions on their impact are lacking. It is hoped that tools would contribute to increased productivity. It is important to document the type of tools used in software projects as more tool assessment is expected in the near future.

- Automation: tools to produce the software impact the quality of deliverables and the productivity of developers and managers. It is convenient to assess the level of automation in the project. CASE tools and project control tools impact project efficiency.

SYSTEM

Characteristics to be included in this dimension are related to the internal characteristics of the system and the implementation technology. The static point of view, as well as the dynamic aspects of performance of the system are considered. It also is possible to characterize the system according to its architecture; Perry et al., [PW92] and Shaw [S89] provide possible scenaria to classify software architectures; associating the system with a particular architecture provides a precise categorization of system's metrics data.

Product

During the software production cycle, different products are generated and referred. Characteristics of products, including work products, deliverables and the final software, are assessed on their formality. Products are the result of process activities, thus a close relationship between processes and products exists. Keep in mind that the software being produced, together with its documentation constitutes the output of the process.

- Type of system: batch, real time, or distributed.

- Level of formality of products generated .
- Degree of reuse within the delivered software or other type of documentation.
- The size and complexity of the system are factors that affect the productivity of agents involved during development. Size may be measured using function-oriented software metrics (e.g., function points) or implementation-oriented software metrics (e.g., lines of code, software science).
- Functional distribution provides an internal view of the system in terms that may be close to the user (e.g., inputs, outputs, files, interfaces, inquires), or to the software implementation (e.g., system architecture, module hierarchy).
- Documentation about the system at several layers of detail, from specifications and conceptual design to implementation. System manuals and documentation produced and maintained have to be identified.

Performance

- Non-functional requirements: aspects such as reliability, robustness, efficiency, and accuracy.
- Constraints imposed on the system have to be identified; execution time, turnaround time, and main storage requirements.
- Error tolerance conditions that determine fault tolerant computing; continuity of operation under non-nominal conditions.
- Accountability of software performance; possibility to measure performance; instrumentation of software to measure performance.

Technology

- Target software technology allows classification of systems for purposes of comparisons. Language and operating system characteristics, as well as hardware characteristics are identified. Single user or multi user microcomputers, minicomputers, and mainframes. Third generation languages (e.g., Fortran, Cobol), object-oriented languages (e.g., C++, Eiffel, Smalltalk), and advanced languages (e.g., 4GL, ADA, Prolog, Lisp) and simple database managers (e.g., Dbase IV) to advanced Data Base Management Systems (e.g., DB2). Operating systems like MS/DOS and MS/Windows to Unix or MVS.

ENVIRONMENT

The benefits provided by the system impact positively customers. Customers use a system when it helps them to solve problems, identify opportunities, look good to others, or simply feel good. A quality system to assure customer satisfaction should be part of any software production process [Z93]. The characteristics of the interfaces with the environment provide the information about its functionality and its importance to the organization. Characteristics to be included in this dimension are related to the application domain, compliance with requirements, usability considerations, and contribution to the organiza-

tion. Application domain characteristics identify the information maintained in the system; business applications should be differentiated from scientific applications.

Compliance involves the overall evaluation of the system from the users perspective. An important activity is to verify that users requirements are meet by the system. Usability involves some degree of subjectivity because it is the user's evaluation of the system. Some researchers in this area have suggested its importance and possible objective quantification of its indicators [N92], [GBL91]. Contribution of a system to the organization is a very important indicator to evaluate a system, there are however serious difficulties as to how to quantify this factor because, besides economic benefits, there are intangible benefits difficult to quantify [B93], [GW91], [C91].

Compliance

- Evaluation of system from the users perspective. User satisfaction and attitudes towards the system.
- Verify that user requirements are meet by the system. This may involve security and safety requirements, and disaster protection.
- Verify the impact on users and their jobs.
- Verify characteristics of the information maintained by the system. The adequacy, appropriateness, timeliness and currency of information.

Usability

Characteristics of the usability of system have been traditionally subjective. The user's appraisal of a system is based on a series of evaluations, including precision and easy of use.

- Effort required to learn, operate, prepare input, and interpret output of a system.
- Level of redundancy requested by the system.
- Level of uniformity of interfaces and their level of complexity.

Contribution

The role of information technology in organizational activities and their impact on the cost structures of firms and markets has to be evaluated. However, systems have to be assessed with regard to specific managerial contexts; business functions, market conditions, industry characteristics, and organizational cultures have an impact on the assessment.

- Characteristics of the organizational environment within which the system is intended to function. The size and structure of the organization involved with the system (e.g., number of departments and personnel interacting with the system), and activities required from agents.
- Type of activities being performed with the information supplied by the system, as well as activities required to gather the data to be supplied to the system. Decision making activities that help run the organization and their value to the organization or society.

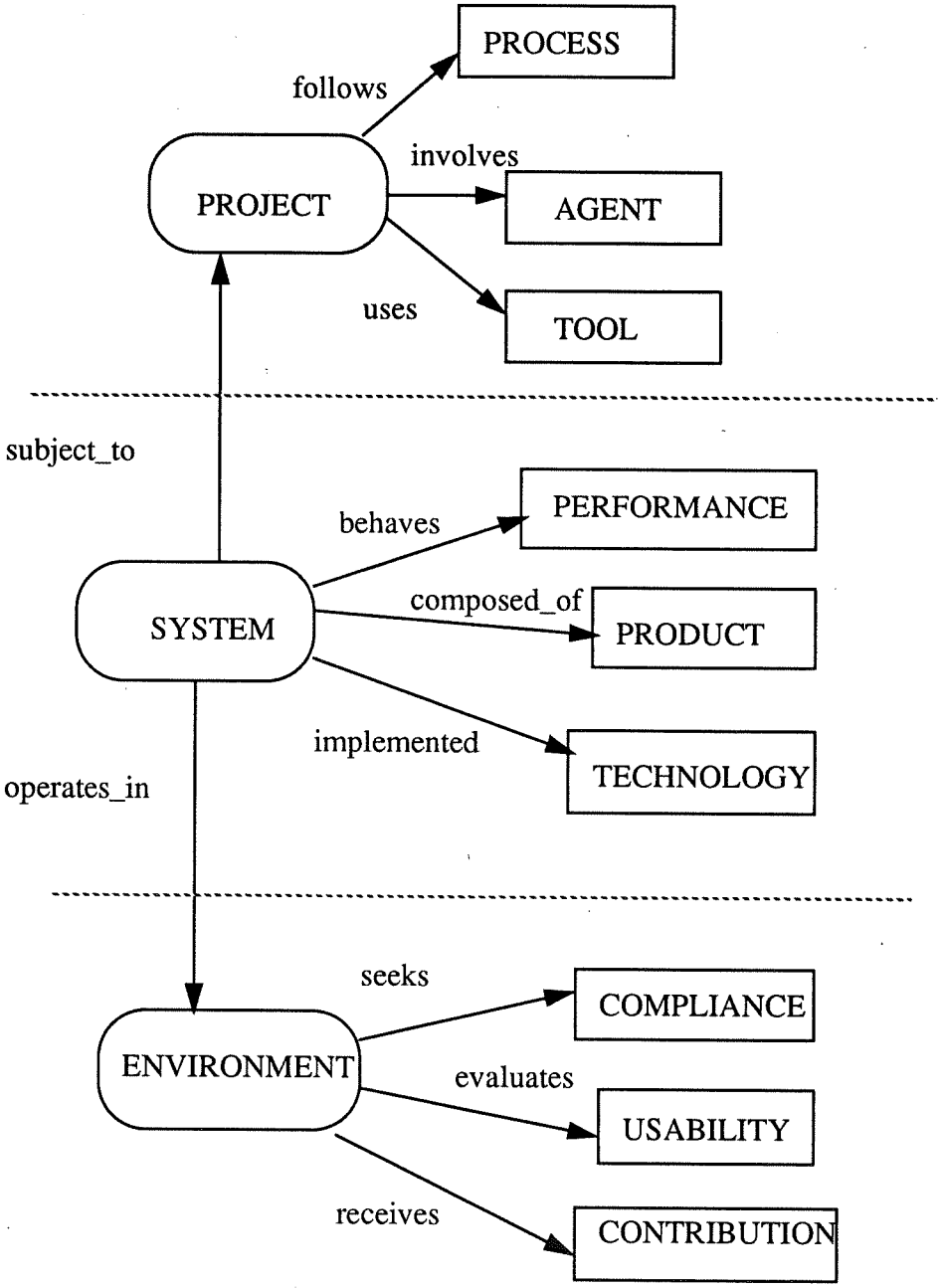
- Increases in productivity or market share by introducing a new system or enhancing an existing one.

Figure 2 presents an Entity-Relationship Diagram that decomposes the framework further into factors. A system may be *subjet_to* several projects during its lifetime. A system *operates_in* some organizational environment. A project *follows* a process, it *involves* some agents, and *uses* some tools. The system is *composed_of* products, it *behaves* with some performance level, and is *implemented* in a particular technology. The environment *seeks* compliance with system requirements, it *evaluates* the usability of the system from the user perspective, and it *receives* a contribution or benefit operating the system.

The three perspectives of the framework are orthogonal as defined. These perspectives represent the before (i.e., development or enhancement), during (i.e., the product itself), and after (i.e., post-evaluation stage) view of systems. However, depending on the interpretation, it is possible to think that there are factors affecting more than one dimension. To avoid misinterpretations, it is important to precise the point of view for each factor. For example, technology considerations may be associated to more than one perspective: project control tools and CASE tools are used during the production process; the system is implemented using a particular hardware and software; users interface with the system through workstations. Further on, the technology used during the production process may be intimately related to the target technology that implements the system. Nevertheless, our concern for technology is clearly defined under the system dimension; it is the system's technology the one being characterized, and not the tools' technology or the user's interface technology.

When implementing a metrics program for an organization, the data to gather requires additional detail. Each factor within the dimensions may require the identification of subsequent attributes to be measured using several possible metrics. Product metrics and process metrics are typical examples of factors that may introduce several attributes to account for quantitative parameters (e.g., size, complexity, productivity). Metrics within each dimension may require nominal or ordinal measurement scales because of a lack of accepted objective measurements in the software community.

FIGURE 2. Factors affecting each dimension of the framework



3.2 Quantitative framework

We have identified in the last section the major dimensions and their associated factors. For purposes of classifying projects according to the framework, it is necessary to define categories for each factor. For some factors there have already been suggestions on categorization (e.g., SEI's Capability Maturity Model and Boehm's Software Development Modes may be used to categorize the process environment), other factors require the definition of categories. In many cases, nominal or ordinal scales are necessary because there are no available metrics yet suggested in the literature. We suggest an initial set of categories for purposes of understanding how to use the framework to categorize systems.

PROJECT

Projects follow specific organizational paradigms that define how working groups set priorities and deal with human issues. These include aspects of how to manage extreme possitions such as continuity and change, tradition and innovation, individual and group, and unity and diversity [C93].

Agents

-1: small software teams (e.g., less than 6 people) develop software in a highly familiar, in-house environment. Most of the people have experience in working with similar applications. Experience with system technology is recognized and system support is easily available. Motivation tends to be high because of individual recognition.

0: medium size software teams (e.g., between 7 and 15) develop software in a constrained environment with semi-formal project organization. Team members have an intermediate level of experience with related applications. System experience tends to be high. Motivation is intermediate because of limited authority.

+1: large size software teams (e.g., over 15) develop software in a constrained environment with formal project organization. Experience is centered around a few experts; team members have a wide mixture of experience with related applications, and system experience is also varied. Motivation tends to be low because of the volatility of personnel.

Process

-1: the project is characterized as one without a stable environment for producing software. Methodologies are adapted for each project and there is no organizational learning from experiences. Performance can only be predicted by individual, rather than project, capability.

0: the project follows a standard process for producing software. The software engineering and software management processes group facilitates software process definition and improvement efforts. An organization-wide training program is implemented to ensure that the staff and managers have the knowledge and skills required to carry out their tasks. Projects use the organization-wide standard software process to create their own defined

software process that encompasses the unique characteristics of the project. Each project uses a peer review process to enhance product quality.

+1: the project sets quantitative quality goals for software products. Productivity and quality are measured for important software process activities. A process database is used to collect and analyze the data from a carefully defined process. Software processes have been instrumented with well-defined and consistent measures that establish the quantitative foundation for evaluating project processes and products.

Tools

The support and partial automation of software production activities is performed using computer-aided software engineering tools. There is growing interest in tools that support the software production process. A reference framework classifies these tools into *tools* that support only specific tasks in the software process, *workbenches* that support only one or a few activities, and *environments* that support (a large part of) the software process [F94].

-1: rudimentary programming tools are available for software production. Basic project-management tools are available (i.e., PERT tools). Text editors, spreadsheets and databases are available for editing purposes

0: programming, verification and validation, configuration management, and metrics and measurement tools are available for software production. Advanced project-management tools are available (i.e., estimation, planning, agendas, bulleting boards, conference desks). Text and graphical editors, spreadsheets and databases are available for editing purposes.

+1: workbench (CASE) tools and software development environments are available for software production. User-interface-development workbenches are available. Text and graphical editors, spreadsheets and databases are available for editing purposes.

SYSTEM

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

Technology

-1: single user and multi user microcomputers and minicomputers, using several types of languages, from third generation languages (e.g., Fortran, Cobol) and languages like C or Pascal. Simple database management systems (e.g., Dbase IV). Operating systems like MS/DOS, MS/Windows, Unix.

0: multi user mainframe technology using third generation languages (e.g., Fortran, Cobol) and languages like C or Pascal, to object oriented languages like C++. Data Base Management Systems like Oracle or DB2. Operating Systems like Unix, MVS, OS/2.

+1: open systems computer technology allowing distributed computing, using several languages, from third generation and object-oriented, to advanced languages (i.e., 4GL, ADA, Prolog, Lisp) and DBMS or OODB. Time-sharing networks are available with advanced operating systems.

Product

Measurement of product attributes is the area that has received more attention from the software metrics community. Traditional metrics like lines of code, cyclomatic number and Halstead Software Science have been followed by functional metrics like function points. There is a large amount of literature documenting product metrics. To categorize the product at this high level, we suggest size and complexity metrics. However, it is important to evaluate the possibility of using the architectural style as an alternative way of classifying the product.

-1: small size of no more than 50 thousand lines of code (or equivalent function points). Low complexity, including straight forward nesting.

0: medium size of up to 300 thousand lines of code. Medium complexity, including simple nesting and some intermodule control.

+1: large size of millions lines of code. High complexity, including highly nested logic and/or considerable intermodule control.

Performance

Performance analysis is the measuring and modeling of software's time, space, efficiency, and accuracy. It eventually drives to the tuning of the software to improve these characteristics. Reliability is the probability of failure-free operation of a system for a specified period of time. Predicting, measuring and managing the reliability of software systems is a major objective of software engineering.

-1: low requirements on reliability, accuracy, and efficiency. The effect of a software failure is an easy recoverable loss to users.

0: medium requirements on reliability, accuracy, and efficiency. The effect of a software failure is a situation from which users can recover without extreme penalty.

+1: high requirements on reliability, accuracy and efficiency. The effect of a software failure can be a major financial loss or a massive human inconvenience.

ENVIRONMENT

There are many different organizational environments associated with aplication domains. Organizations may be divided by function, by type of product, or by type of project. Applications can be classified as oriented to transaction processing, maintaining the corporate database, operation systems for process control and scientific computing,

and decision support systems. Additional example environments may help any future taxonomy endeavors:

- transportation systems, vehicular traffic control, and air traffic control;
- manufacturing facilities that need to quickly reconfigure plant operations to meet changing requirements and permit on-line maintenance and operation.
- sensor systems for monitoring weather patterns, seismic data, the status of power-distribution grids, and pollutant distribution
- satellites, fiber-optic networks, and high-speed switches that transmit large volumes of live audio, video, and text data.
- patient monitoring, heart-lung machines, CAT (Computer Axial Tomography) scanners, MRI (Magnetic Resonance Imaging), and other advanced medical equipment.
- surveillance, command and control, and other defense systems.

Compliance

-1: basic requirements are not satisfactory fulfilled by the system. There is no impact on the users and their jobs. The user's appraisal of the system is poor.

0: basic requirements are fulfilled by the system. There is some impact on the users and their jobs. The user's evaluation of the system indicates that some areas could be improved.

+1: all requirements are fulfilled by the system. The service provided by the system is appropriate, the information is adequate, current, and on time. There is a major positive impact on the users and their jobs. The user's compliance has been evaluated satisfactorily by the users.

Usability

-1: the system is not providing interfaces in user terms. Redundant information is required form the users. The system requires a long time to learn.

0: the system provides interfaces in user terms. The system requires a fair amount of time to learn. The user is concerned with the system because error messages are not clear and there are no shortcuts for experienced users.

+1: the system is user-oriented, it is efficient to use, prevents errors from the user and signal clearly the seriousness of the errors. Infrequent users have facilities to return to use the system without having to learn it over, and frequent users find shortcuts that improve their efficiency of use. The system requires a short time to learn

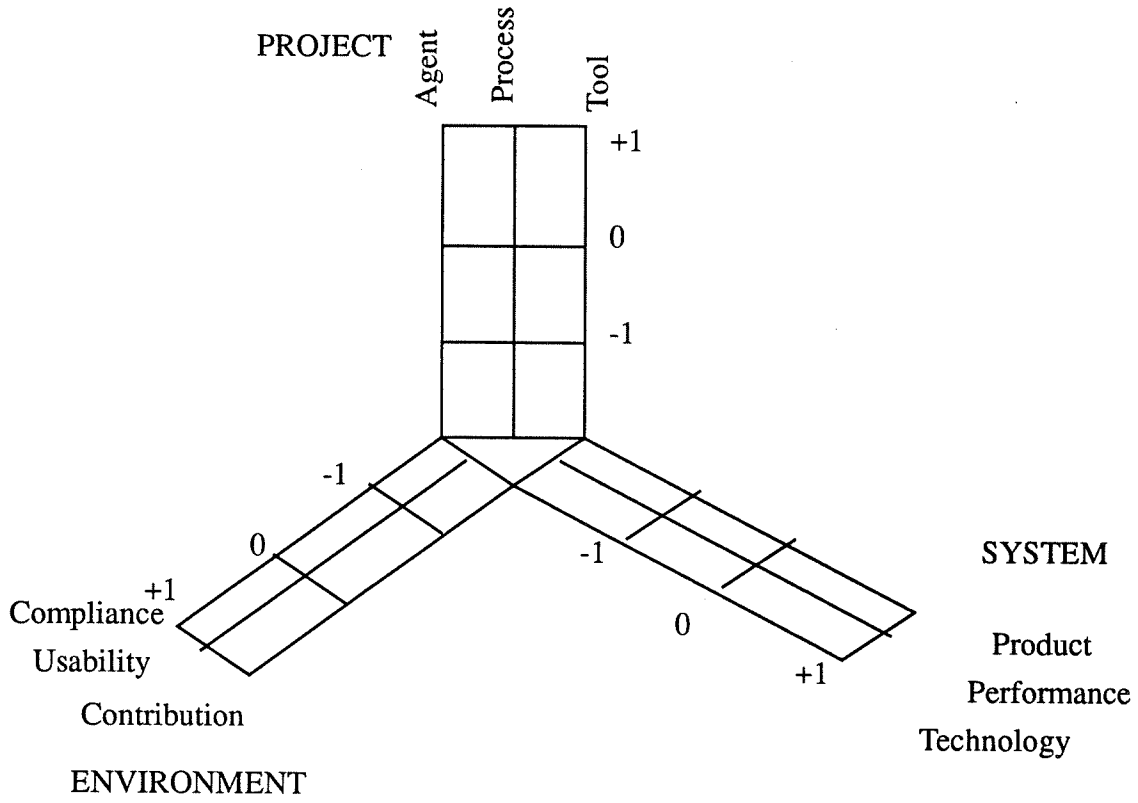
Contribution

-1: there is no major contribution to the organization, the system has automated the same tasks performed manually.

0: the process of decision-relevant information is done in a cost-effective way, improving the quality and speed of management decision making processes. Improvements on monitoring and performance measurements have reduced cost by inducing decentralization of decisions.

+1: external coordination cost have been reduced, making the organization more cost-effective. At the same time internal coordination cost have been reduced. The system provides tangible benefits to the organization.

FIGURE 3. Framework categories



From the categories just suggested, it is possible to evaluate if all combinations are possible. Uncommon situations would be to have a small team developing a large system, or the inverse, a large team developing a small system. Another uncommon case would be to have rudimentary tools to develop systems in a sophisticated process environment, or have rudimentary tools to build large systems. It would also be uncommon to implement a system with high performance requirements in a low profile technology.

4.0 Current approaches versus the framework

Let us find out the relationship between our framework and current approaches presented in Section 2. Table 1 was built to relate the dimensions of the framework with each of the current approaches. The numbers in the table are related to each factor suggested in those approaches (see the appendix). Current productivity approaches take in consideration primarily project considerations (i.e., process and people), also some system characteristics (i.e., product, performance) and finally some aspects of usability. Current software quality factors approaches do not account for the project dimension (i.e., the process), rather concentrating in the system dimension (i.e., the product), and some usability factors. Current approaches do not suggest any factor related to the contribution of the system to the organization.

Let us present a summary of results of the analysis of Table 1:

PROJECT

Agent: there is a complete agreement between current approaches and our framework. The experience of participants is the primary element considered within this factor. Additionnally, team characteristics and their organization are also included under agent.

Process: development methodologies, process model, current programming practices, use of standards, and so forth correspond directly with the framework. Current approaches stress issues about who participated during the process (e.g., users, designers, programmers) or who originated system changes; these were classified as process concerns and not as agent factors, even though they affect both. Aspects related to schedules, budget and project constraints were also classified under this item. Productivity concerns that relate several factors (e.g., people, activities) were also classified under process. Hardware under concurrent development was classified as a process concern rather than a product concern.

Tool: usage of tools during software production corresponded directly with our framework. Concerns about turnaround time during development, as well as the power of the tools, were accounted for in this factor. Availability of several locations for software production were also accounted in the tool factor.

SYSTEM

Technology: aspects about the target language and the software and hardware to run the system corresponded directly to this factor; one approach identified some specific devices like displays and random access devices. Portability aspects and device independence were attached to performance rather than technology.

Product: aspects about size and complexity of the product correspond directly with the framework. Type of software, such as batch, real time and so on, were related to the product instead of the domain. The final product (i.e., the system), documentation produced during the process, and operation and users manuals were classified under product. Aspects such as constraints on program design, that affect the product, were rather classified under process. The extent that a program can be reused in another application was

Table 1: Framework versus current approaches

		WF 77	BZ 78	B 81	M 81	SEI	CDS 86	RB 89	KMC 91	BBL 76	Mc 79
PROJECT	Agent	4, 5, 7, 8, 9	1, 2	8, 9, 10, 11, 12	18, 35, 36, 38		1		1--20, 29--32		
	Process	2, 3, 6,10, 11 15, 16, 17, 18	5, 7 8, 10 16, 17	13, 15	12, 13, 14,19, 37, 40	CMM	3,4,5 6, 8, 17,22	2, 3, 4	21, 22, 23, 24, 26, 27		
	Tool	12, 13	15 18, 19, 20, 21	7, 14	26, 39		7, 9, 10, 23		25, 28		
SYSTEM	Technology		9	6	15 27, 28, 29, 30, 31, 34		2				11
	Product	19, 20, 21, 22, 25, 26, 27, 28, 29	3, 4, 12,14	2, 3	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 16, 17, 20, 21, 22, 23, 24, 25		11, 13, 15, 16, 18, 20	1		1, 3, 5, 6, 7, 10, 12, 13, 19, 20, 21, 22	1, 6, 7, 8, 10, 11, 12, 14, 17, 18, 19, 20, 22, 25, 26, 30, 31, 32, 33
	Performance	23, 24	6, 11, 13	1, 4 5	32, 33		12,14 21,24			4,9,11 11,14 15,16	2, 3, 9, 15, 16, 21, 23, 24
ENVIRON-MENT	Compliance	14					19			2	4,13
	Usability	1			11					8, 17, 18, 23	5, 27, 28, 29
	Contribution										

classified under product; degree of reuse was not mentioned by any of the approaches. Reusability is a notion that affects also the process dimension.

Performance: constraints imposed on the software related to timing or storage, reliability, and efficiency, are classified under performance. Robustness, error tolerance, accuracy, and instrumentation were also classified under performance. Performance considerations have a close relationship with the evaluation of the system in the compliance factor. Portability was associated to performance instead of technology. There is also the issue of system security and safety that might be associated to usability or performance, but we have associated these to compliance.

ENVIRONMENT

The type of problem or application, such as real time, mathematical, and business applications, suggested by current approaches, were associated to the product. However, it is convenient to recognize its relationship to the environment.

Compliance: software quality models suggest completeness criteria that we associated to compliance. Security and privacy, and safety concerns were marginally mentioned by current approaches, and also classified under compliance. There is no explicit mention of customer system evaluation in current approaches.

Usability: interface complexity or quality, and system operability were classified under usability. Accessibility, communicativeness and human-engineered concerns were also associated to usability. Concerns on degree of familiarity with the system for purposes of using shortcuts were also identified under usability.

Contribution: none of the approaches identified factors on the contribution of the system to the overall productivity of the organization. Nevertheless, performance factors may impact the contribution of the system to the organization.

5.0 Using the framework

Let us apply the framework to a hypothetical system adapted from the literature according to our interpretation [Y89]. The example describes the information processing activities for a publishing company that prints software engineering books. The company has no experience building information systems, thus requiring to engage some personnel and a consulting firm for development purposes.

The company is part of a larger regional organization and interacts with other departmets like Accounting and Technical Services. Internally, the editorial is composed of four departments. Administration Services handles the day to day interactions with the customers; orders, invoice, payments, returns and credits, arranging shipments of books, and interacted with the Accounting Department. Sales & Marketing produces the catalogs for the various books, running ads in computer magazines, sending brochures, calling to large corporations for sales purposes. Acquisitions finds new authors and new books, discussing with authors up to the point of getting a manuscript. Editing takes a manuscript

and turns it into a published book, copyediting of the book, interacting with printers to obtain proposals; artwork, book cover and inside contents were also part of their responsibilities.

Project

Agent (-1) The team of developers is small, with some experience in this type of systems, and having access to the experience of the consulting firm. 1 senior analyst, 1 user representative, 1 senior programmer, and 1 junior programmer. The project organization is informal.

Process (-1) The system is an ad-hoc type of software development. The system is built anew, without reuse of existing parts. The methodology is Structured Analysis and Design and the process model is waterfall.

Tool (-1) There are some project control tools available, text editors and the C compiler.

System

Technology (-1) The language is C in the Unix operating system, runing in a PDP-11/45.

Product (-1) The size of the system is small, between 10 to 20 thousand lines of code. There are 50 books in stock; 4 to 6 new titles per year; two dozen authors; interaction with 200 potential authors; 50 orders per day; average order $100; 50,000 books shipped per year. The documentation being generated is composed of Data Flow Diagrams, Data Dictionary, Process Specifications, Entity Relationship Diagrams, Code, and Operations Manual. The system is file-oriented with textual screen-interactions.

Performance (-1) There are no major concerns on reliability of the system, and there is no major impact in case of software failures.

Environment

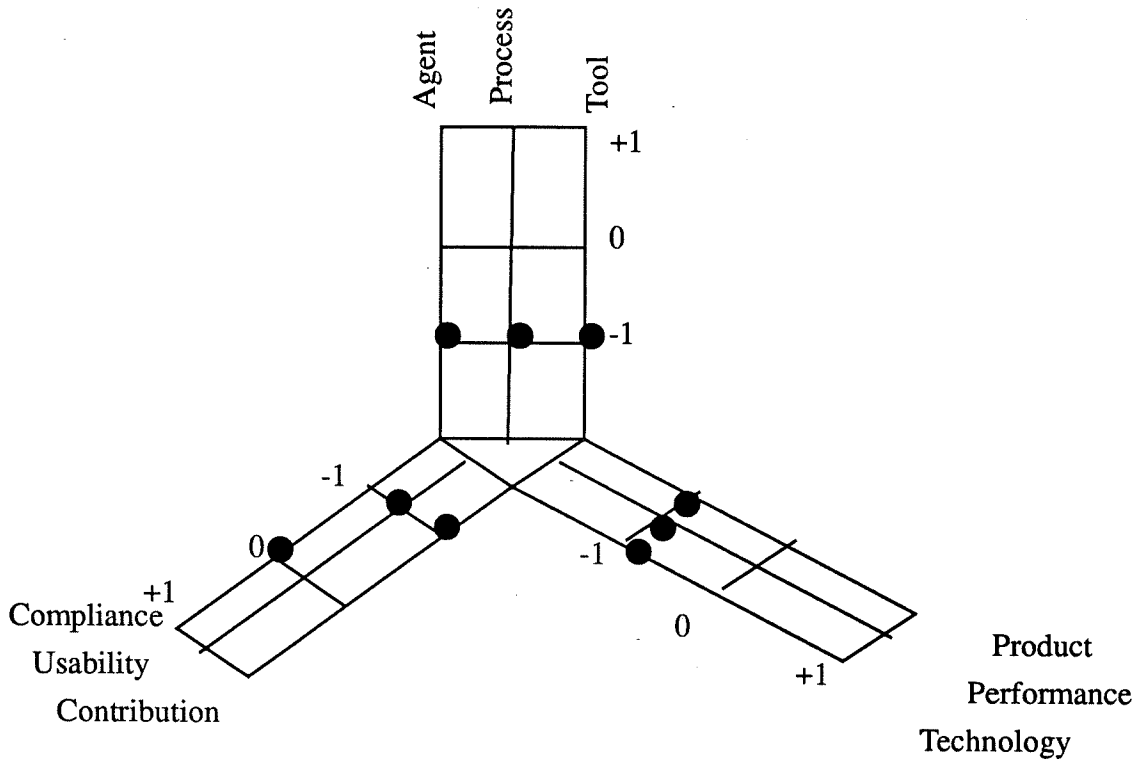
It is a business application in the book publishing industry. The following departments are using the system: Administrative Services, Sales & Marketing, Acquisitions, and Editorial. Twenty persons interact with the system, five persons per department. The organization is very small with a yearly income of less than $10 million. The type of data kept by the system is books, customers, orders, payments, authors, returns, inventory, and so on.

Compliance (0) Compliance with requirements is fair. The users are not very familiar with computerized applications.

Usability (-1) The are no major contributions on usability. The external entities interacting with the system: are Customers, Management, Authors, Warehouses, Sales, Editors, Credit Agencies, Printers, Accounting, Marketing

Contribution (-1) The system is replacing the current manual system. There are no major economic benefits expected from the system.

FIGURE 4. Example classification



According to this example, each of the perspectives, except compliance, remain at the lowest category level. The development project is rudimentary, the system is symple, and the environment's evaluation is fair. In terms of Boehm's software development modes, this project would classify as organic.

6.0 Conclusions and further research

We have proposed a comprehensive software metrics framework that integrates in a top down fashion important characteristics of systems, their development environment, and the organizational appraisal of the system. The framework includes aspects related to project characteristics during the evolution of software, the software internal structure, and the organizational environment where the software is used. Several factors that impact project productivity, product quality, and users satisfaction can be organized around the three dimensions of the framework (i.e., project, system, environment). The framework perspectives represent the before (i.e., development or enhancement), during (i.e., the product itself), and after (i.e., post-evaluation stage) view of systems.

Project formality involves several parameters such as project organization, process model, software representation techniques, and degree of automation during the process. The experience of developers, users, and managers involved during software production is another factor taken into consideration. Tools used during software production have to be

identified to characterize the level of automation during software production. Innovation, independently of being domain-oriented or technical-oriented has an impact on the productivity of the project.

The characteristics of the system depicts its internal structure, its functional distribution, and product size and complexity. The target technology that implements the software identifies precisely the available building blocks for the system, including the hardware and the software environment within which the system is operating. Performance considerations contribute to characterize dynamic features important for evaluation purposes.

The organizational environment surrounding the operation of the system provides additional information to classify the system. Compliance with requirements, the usability of the system, and the contribution of the system to the organization are the factors to account for in this dimension. Characteristics of the organization and the scope of the system within the organization are indicators of the importance of the system. Embedded systems functioning within precise interfaces can be differentiated from information systems that interact with users from different locations. The characteristics of the interfaces with agents, departments, and other systems, indicates the degree of formality and reliability of the system.

The comprehensive software metrics framework acknowledges current contributions from productivity and quality models, and introduces an additional perspective related to the organizational appraisal of the operating software. It has been identified in this research that current approaches on productivity and quality have neglected the importance of the contributions of systems to the organization. Economic benefits are a primary concern for any organization, as well as intangible contributions of the system, like improving service to customers. Customer satisfaction issues and the social impact of computing (see for example CACM special issue, January 1994) have received research contributions lately, and we expect that the software enginering community, in particular the software metrics community, will get eventualy involved in proposing metrics in these areas. Metrics that consider the overall benefits of a system to the organization are currently needed. Cost/benefit analysis of operational systems, as well as software quality evaluations involving user satisfaction should be included in organizational metrics programs.

The framework, as depicted, presented the first two levels of decompositional factors. Metrics can be suggested for each factor and more detail can be added to the framework, by introducing additional levels of decomposition. Each of these factors can be decomposed into several subfactors or attributes, and corresponding metrics be suggested. It would be, for the organization, a matter of evaluating the cost involved in gathering that additional detail. Using the framework it is possible to understand the meaning of each factor or subfactor, and to understand precisely the relationships of composite metrics, those computed from measures corresponding to more than one factor.

We believe that by classifying projects according to these dimensions, it is possible to utilize historical records consistently. There are many variables that affect software production, the framework provides a vehicule to classify project similarity. Boehm sug-

gested software development modes [B81] to characterize software projects. SEI's Capability Maturity Model also proposed a classification in terms of process model considerations. Each organization has to determine the set of metrics to be gathered for each project using the framework, following guidelines like the GQM and improvement paradigms [BR88], according to their own needs.

Defining a comprehensive software metrics framework to be shared among organizations is a step towards improvement oriented software engineering. According to implemented metrics programs, the initial stage of implementation in an organization 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 organization metrics program. Further research is required 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 extrapolate metrics gathered following the framework.

There are several applications to the framework. The framework allows classifying software projects data uniformly for comparison purposes. The framework can be used as a baseline to establish metric programs in organizations. Assessment of selected metrics, for analysis or estimation purposes, can be validated against the framework. Current research involves validating and proposing a set of metrics for an organization. Validation of the framework requires classifying several real systems to refine its categories.

More research is required to propose metrics for each of the factors and attributes identified in the framework. Metrics at high levels are mostly subjective and are represented using basic measurement scales like nominal or ordinal. The objective is to propose metrics for low level attributes at the lowest measurement scale (i.e., absolute), and obtain composite metrics at higher levels of the framework hierarchy with higher measurement scales (i.e., interval or ratio).

Further research to classify automatically systems from historical metrics data is deemed important. 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 automatically a system to the categories of the framework, avoiding subjective classifications.

References

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

[B93] Brynjolfsson, E. 'The Productivity Paradox of Information Technology', Communications of ACM, Vol. 36, No. 12, December 1993, pp. 66-77.

[BBL76] Boehm, B.W.; Brown, J.R.; Lipow, M. 'Quantitative Evaluation of Software Quality', International Conference on Software Engineering, 1976, pp.592-605.

[BR88] 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.

[BZ78] Basili, V.R.; Zelkowitz, M.V. 'Analyzing Medium-scale Software Development', 3rd. International Conference on Software Engineering, May 10-12, 1978, Atlanta, Georgia, USA.

[C91] Clemons, E.K. 'Evaluation of Strategic Investments in Information Technology', Communications of the ACM, Vol. 34, No. 1, January 1991, pp.22-36.

[C93] Constantine, L.L. 'Work Organization: Paradigms for Project Management and Organization', Communications of the ACM, Vol. 36, No. 10, October 1993, pp. 34-43.

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

[F94] Fuggetta, A. 'A Classification of CASE Technology', IEEE Computer, Vol. 26, No. 12, December 1993, pp. 25-38.

[GBL91] Gould, J.D.; Boies, S.J.; Lewis, C. 'Making Usable, Useful, Productivity. Enhancing Computer Applicatons', Communications of the ACM, Vol. 34, No. 1, January 1991, pp.74-85.

[GC87] Grady, R.B.; Caswell, D.L. 'Software Metrics: establishing a company-wide program', Prentice-Hall, 1987.

[GW91] Gurbaxani, V.; Whang, S. 'The impact of Information Systems on Organizations and Markets', Communications of the ACM, Vol. 34, No. 1, January 1991, pp.59-73.

[H88] Humphrey, W.S. 'Characterizing the Software Process: A Maturity Framework', IEEE Software, March 1988, pp. 73-79.

[KMO91] Kemayel, L.; Mili, A.; Ouederni, I. 'Controllable Factors for Programmer Productivity: A Statistical Study', Journal of Systems and Software, 1991; 16, pp.151-163

[LZ91] Loucopoulos, P.; Zicari, R. 'Conceptual Modeling, Databases and CASE: An Integrated View of Information Systems Development', McGraw Hill, 1990.

[M79] 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.

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

[N92] Nielsen, J. 'The Usability Engineering Life Cycle', IEEE Computer, March 1992, pp. 12-22.

[P93] Pfleeger, S.L. 'Lessons Learned in Building a Corporate Metrics Program', IEEE Software, May 1993, pp. 67-74.

[PCC93] Paulk, M.C.; Curtis, B.; Chrissis, M.B.; Weber, C.W. 'Capability Maturity Model, Version 1.1', IEEE Software, July 1993, pp. 18-27.

[PW92] Perry, D.E.; Wolf, A.L. 'Foundations for the Study of Software Architecture', ACM SIGSOFT, Software Engineering Notes, Vol. 17, No. 4, Oct. 1992, pp.40-52.

[RB89] 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.

[S89] Shaw, M. 'Larger Scale Systems Require Higher-Level Abstractions', ACM SIGSOFT, Software Engineering Notes, Vol. 14, No. 3, May 1989, pp.143-146.

[S93] Subramanian, G.H. 'An Empirical Examination of Software Development Modes', Journal of Systems and Software, 1993; 23:3-7.

[WF77] Walston, C.E.; Felix, C.P. 'A method of programming measurement and estimation', IBM Systems Journal, No. 1, 1977, pp. 54-73.

[Y89] Yourdon, E. 'Modern Structured Analysis', Prentice-Hall, 1989.

[Z93] Zultner, R.E. 'TQM for Technical Teams', Communications of the ACM, Vol. 36, No. 10, October 1993, pp. 78-91.

Appendix

PRODUCTIVITY FACTORS

Walston and Felix productivity factors

The objective of this research was to search for a method of estimating programming productivity. Twenty-nine factors that correlate with programming productivity were identified:

- 1. Customer interface complexity
- 2. User participation in the definition of requirements
- 3. Customer originated program design changes
- 4. Customer experience with the application area of the project
- 5. Overall personnel experience and qualifications
- 6. Percentage of programmers doing development who participated in design of functional specifications
- 7. Previous experience with operational computer
- 8. Previous experience with programming languages
- 9. Previous experience with application of similar or greater size and complexity
- 10. Ratio of average staff size to duration (people / month)
- 11. Hardware under concurrent development
- 12. Development computer access, open under special request
- 13. Development computer access, closed
- 14. Classified security environment for computer and 25% of programs and data
- 15. Structured programming
- 16. Design and code inspections
- 17. Top down development
- 18. Chief programmer team usage
- 19. Overall complexity of code developed
- 20. Complexity of application processing
- 21. Complexity of program flow
- 22. Overall constraints on program design
- 23. Program design constraint on main storage
- 24. Program design constraints on timing
- 25. Code for real time or interactive operation, or executing under severe timing constraint

- 26. Percentage of code for delivery
- 27. Code classified as non-mathematical application and I/O formatting programs
- 28. Number of classes of items in the data base per 1000 lines of code
- 29. Number of pages of delivered documentation per 1000 lines of delivered code

Basili and Zelkowitz factors on software development

Data from several projects was collected at the Software Engineering Laboratory (NASA Goddard Space Flight Center and the Universoty of Mariland) to evaluate software engineering methodologies. For each project, a set of factors that affect software development were gathered.

People factors

- 1. size and expertise of development team,
- 2. team organization

Problem factors

- 3. type of problem to solve,
- 4. magnitude of problem,
- 5. format of specifications,
- 6. constraints placed upon solution

Process factors

- 7. specification,
- 8. design languages
- 9. programming languages,
- 10. techniques such as code reading, walkthroughs, top-down design and structured programming

Product factors

- 11. reliability,
- 12. size of system,
- 13. efficiency,
- 14. structure of control

Resource factors

- 15. target and development computer system,
- 16. development time,
- 17. budget

Tools

- 18. libraries,
- 19. compilers,
- 20. testing tools,
- 21. maintenance tools

Boehm's Software Development Modes

Boehm distinguises three modes of software development: organic, semidetached and embedded.

Organic Mode

Small software teams develop systems in a highly familiar, in-house environment. Personnel in the project have extensive experience working with related systems within the organization, and have a thorough understanding of how the system under development will contribute to the organization's objectives. There is no communication overhead. The project is relatively relaxed about the way the software meets its requirements and interface specifications, allowing for negotiation when discrepancies arrive. The development environment is stable. Minimal need for innovative architectures or algorithms. There is low premium on early completion of the project. The projects are relatively small (50 KDSI). Larger organic products may be developed by using existing software.

Semidetached Mode

Team members have an intermediate level of experience with related systems. Teams are composed of a wide mixture of experienced and inexperienced people. Team experience is about some aspects of the system under development, but not others. A typical project might be a transaction processing system with some very rigorous interfaces. The size of the software is greater than the organic-mode (300 KDSI).

Embedded Mode

The project needs to operate within tight constraints. The systems must operate within a strongly coupled complex of hardware, software, regulations, and operational procedures. Negotiation is not very easy, requiring to expend more effort in accomodating changes and fixes. This type of projects charts its way through unknown territory, requiring smaller teams of analysts during the early stages to avoid communication overhead. During detailed design and implementation, large teams are required to build the system faster.

Boehm distinguished the following features for Software Development Modes

- 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
- Premium on early completion
- Product size range

Boehm proposed the following cost drivers for estimation purposes

Product Attributes

- 1. Required software reliability: probability the software performs its intended functions satisfactorily over its next run or its next quantum of execution time
- 2. Data base size: the total amount of data, its structure, and complexity of data handling activities
- 3. Software product complexity: level of complexity by type of module; control, computation, device-dependent, data management

Computer attributes

- 4. Execution time constraint imposed upon a software subsystem
- 5. Main storage constraint imposed on a software system
- 6. Virtual machine volatility (hardware and software that the subsystem calls upon to accomplish its tasks)
- 7. Computer turnaround time: response time experienced by the project team developing the subsystem

Personnel Attributes

- 8. Analyst capability: rating of five levels of analyst capability
- 9. Application experience: time of experience of the team with this type of application
- 10. Programmer capability: factors related to programmer ability, efficiency and thoroughness, and ability to cooperate and communicate
- 11. Virtual machine experience: time of project team experience with the virtual machine
- 12. Language experience: time of project team experience with the programming language

Project Attributes

- 13. Use of modern programming practices: specific practices included are top down requirements analysis and design, structured design, top down incremental development, design and code walkthroughs and inspections, structured code, program librarian
- 14. Use of software tools to develop the software
- 15. Development schedule constraint: from strechout to accelerated schedule constraints

Factors not included in Boehm's COCOMO

- Type of application
- Language level
- Other size measures: complexity, entities, and specifications
- Requirements volatility
- Personnel continuity
- Management quality
- Customer interface quality
- Amount of documentation
- Hardware configuration
- Security and privacy restrictions

Mohanty's factors in software cost estimation

Significant factors that have been considered by various model builders to estimate software cost are organized by system size, data base, system complexity, type of program, documentation, environment, and other factors.

System size

- 1. Number of estimated instructions
- 2. Number of delivered machine language instructions
- 3. Number of delivered source language instructions
- 4. Percentage of new instructions
- 5. Percentage of clerical instructions
- 6. Number of decision instructions
- 7. Number of non-decision instructions
- 8. Percentage of information storage and retrieval instructions

Data base

- 9. Number of words in data base

System complexity

- 10. System complexity
- 11. Complexity of interfaces
- 12. System uniqueness (familiarity with problem, hardware, software)
- 13. Job type and difficulty (number of system interactions)
- 14. Degree of difficulty (old-easy, medium, hard; new-easy, medium, hard)
- 15. Hardware-software interfaces
- 16. Program structure consideration
- 17. Number of files, reports, and application programs
- 18. Total life cycle manpower, total development manpower, total test and validation manpower
- 19. Total life cycle time and total development time

Type of program

- 20. Type of application (business/non-business0
- 21. Program categories (control, i/o, pre/post processor, algorithm, data management, time-critical)
- 22. Real-time/non-real-time

Documentation

- 23. Documentation in pages
- 24. Number of document types for customer
- 25. Number of document types for internal use

Environment

- 26. System development environment
- 27. New or old computer
- 28. Special displays (used/not used)
- 29. Number of display consoles
- 30. Random access device (used/not used)
- 31. Language used (HOL/Assembly)
- 32. Core occupancy constraints
- 33. Computer system speed and memory capacity

- 34. Time sharing or batch processing
- 35. Programmer familiarity with language/compiler, etc.
- 36. Programmer experience in programming (years)
- 37. Programmer participation in design (percentage of total effort)
- 38. Personnel continuity
- 39. Number of locations for program development
- 40. Productivity (lines of code/unit time)

Other factors

- Number of miles travelled
- Frequency of operation after delivery of software
- Degree and time phasing of simulation
- Intent of prototype code
- Fault tolerant computing
- Safety
- Single CPU/multi-CPU application environment
- Growth requirements-maintainability

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 indentifies the current process maturity level of an organization and the most critical issues to software quality and process improvement.

The CMM contains five levels of software process maturity: Initial, Repeatable, Defined, Managed, and Optimizing.

Initial: the organization is characterized as one without a stable environment for developing and maintaining software. Few stable software processes are in place, and performance can only be predicted by individual, rather than organizational, capability.

Repeatable: the organization has installed basic software management controls; that is, stable processes are in place for planning and tracking the software project. Project software managers track software costs, schedules, and functionality; problems in meeting commintments are identified. Software configuration management procedures are used to baseline and control software requirements. Project standards exist, and the software quality assurance group ensures that they are followed. In essence, there is a stable, managed, working environment.

Defined: the organization has a standard process for developing and maintaining software across the organization. The software engineering and software management processes

group facilitates software process definition and improvement efforts. An organization-wide training program is implemented to ensure that the staff and managers have the knowledge and skills required to carry out their tasks. Projects use the organization-wide standard software process to create their own defined software process that encompasses the unique characteristics of the project. Each project uses a peer review process to enhance product quality.

Managed: the organization sets quantitative quality goals for software products. Productivity and quality are measured for important software process activities across all projects in the organization. A process database is used to collect and analyse the data from a carefully defined process. Software processes have been instrumented with well-defined and consistent measures that establish the quantitative foundation for evaluating project processes and products.

Optimizing: the organization focuses on continuous process improvement. The organization has the means to identify weak process elements and strengthen them, with the goal of preventing the occurrence of defects. Statistical evidence is available on process effectiveness and is used in performing cost-benefit analyses on new technologies. Innovations that exploit the best software engineering practices are identified.

Conte, Dunsmore and Shen factors on productivity

There are a large amount of factors that appear to affect the software development process and the product. Some of the factors suggested by the authors are:

People factors

- 1. individual capability, years of experience, language experience, experience with similar problems, previous experience with the system being used, the size of the team, organization of the team, experience of the team working together, morale level of individuals, the quality of management

Process factors

- 2. the programming language,
- 3. use of a program design or specification language,
- 4. top down design, HIPO diagrams, use of structured programming, use of chief programmer teams, code walkthroughs, code inspections,
- 5. milestones,
- 6. use of a program librarian,
- 7. testing tools, automatic flowcharters, optimizing compilers, utility tools,
- 8. the development schedule,
- 9. data base system availability,
- 10. multisite development

Product factors

- 11. size of the product, size of the data base,
- 12. real-time requirements, reliability, portability,
- 13. control structure, data structure, number of modules, module coupling,
- 14. memory requirements,
- 15. complexity,
- 16. the amount of reused code,
- 17. state of problem definition,
- 18. the amount of documentation,
- 19. security restrictions,
- 20. type of software

Computer factors

- 21. response time, turnaround times,
- 22. hardware under concurrent development,
- 23. the development machine system volatility,
- 24. storage constraints, timing constraints

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 the software development environment is homogeneous, i.e., many similar projects are developed for the same aplication area. The considerations for homogeneous environments are:

- 1. Same application area
- 2. A standard process model
- 3. Software development methodology is similar across projects
- 4. Great deal of reuse of code from prior projects

Kemayel, Mili, and Ouederni programmer productivity factors

Controllable factors suggested by this authors are related to personnel, the software process and the user community.

Kemayel, Mili and Ouederni proposed the following 33 controllable factors:

Personnel factors

Motivation

- 1. Recognition: reaction of the institution to programmer's performance
- 2. Achievement: satisfaction the programmer gains from doing a challenging task properly
- 3. The work: nature of the tasks to be executed
- 4. Responsibility: degree of responsibility to personnel
- 5. Advancement: possibility of career improvement
- 6. Salary: remuneration to personnel
- 7. Possibility for growth: professional growth expected in the company
- 8. Interpersonal relations, subordinates:
- 9. Status: importance of the worker in the company
- 10. Interpersonal relations, superiors
- 11. Interpersonal relations, peers
- 12. Technical supervision: willingness of supervisors to help solve problems
- 13. Company policy and administration: command structure of the company
- 14. Working conditions: office, space, and so on
- 15. Factors in personal life
- 16. Job security

Personnel experience

- 17. Application domain experience
- 18. Virtual machine experience: working logical environment
- 19. Programming language experience
- 20. Experience with the user community: familiarity with the user community

Process factors

Project management

- 21. Using a goal structure
- 22. Adherence to a software life cycle: precise definition of stages
- 23. Adherence to an activity distribution: precise definition of activities
- 24. Usage of cost estimation procedures: software cost-estimation model being used

Programming environment

- 25. Programming tools: use and power of tools
- 26. Modern programming practices: use of advanced techniques
- 27. Programming standards
- 28. Power of equipment used: memory space and time limitations

User factors

- 29. Previous education in computing
- 30. Experience in computing: previous use of computers
- 31. Experience with the type of application: involvement in similar applications
- 32. Experience with the group of programmers / analysts

QUALITY FACTORS

Boehm et al., software quality framework

A set of characteristics important for software were proposed. Metrics to assess the degree to which the software has the defined characteristic were developed and correlated with the characteristics. Refinements were perfromed to the set of characteristics into a set that supports software quality evaluation. Relationships were established among characteristics and refined characteristics. Finally, the metrics were also refined. The characteristics developed were:

Characteristics

1. Understandability: Code possesses the characteristic understandability to the extend that its purpose is clear to the inspector.

2. Completeness: Code possesses the characteristic completeness to the extent that all its parts are present and each part is fully developed.

3. Conciseness: Code possesses the characteristic conciseness to the extent that excessive information is not present.

4. Portability: Code possesses the characteristic portability to the extent that it can be operated easily and well on configurations other than its current one.

5. Consistency: Code possesses the characteristic internal consistency to the extent that it contains uniform notation, terminology and symbology within itself, and external consistency to the extend that the content is traceable to the requirements.

6. Maintainability: Code possesses the characteristic maintainability to the extent that it facilitates updating to satisfy new requirements or to correct deficiencies.

7. Testability: Code possesses the characteristic testability to the extent that it facilitates the establishment of verification criteria and supports evaluation of its performance.

8. Usability: Code possesses the characteristic usability to the extent that it is reliable, efficient and human-engineered.

9. Reliability: Code possesses the characteristic reliability to the extent that it can be expected to perform its intended functions satisfactorily.

10. Structuredness: Code possesses the characteristic structuredness to the extent that it possesses a definite pattern of organization of its interdependent parts.

11. Efficiency: Code possesses the characteristic efficiency to the extent that it fulfills its purpose without waste of resources.

Refined Characteristics

12. Device-Independence: Code possesses the characteristic device-independence to the extent that it can be executed on computer hardware configurations other than its current one.

13. Self-Containedness: Code possesses the characteristic self-containedness to the extent that it performs all its explicit and implicit functions within itself.

14. Accuracy: Code possesses the characteristic accuracy to the extent that its output are sufficiently precise to satisfy their intended use.

15. Robustness/Integrity: Code possesses the characteristic robustedness to the extent that it can continue to perform despite some violation of the assumptions in its specification.

16. Accountability: Code possesses the characteristic accountability to the extent that its usage can be measured.

17. Accessibility: Code possesses the characteristic accessibility to the extent that it facilitates selective use of its parts.

18. Communicativeness: Code possesses the characteristic communicativeness to the extent that it facilitates the specification of inputs and provides outputs whose form and content are easy to assimilate and use.

19. Self-Descriptiveness: Code possesses the characteristic self descriptiveness to the extent that it contains enough information for a reader to determine or verify its objectives, assumptions, constraints, inputs, outputs, components, and revision status.

20. Legibility: Code possesses the characteristic legibility to the extent that its function is easily discerned by reading the code.

21. Augmentability: Code possesses the characteristic augmentability to the extent that it can easily accomodate expansion in component computational functions or data storage requirements.

22. Modifiability: Code possesses the characteristic modifiability to the extent that it facilitates the incorporation of changes, once the nature of the desired change has been determined.

23. Human Engineering: Code possesses the characteristic human engineering to the extent that it fulfills its purpose without wasting the user's time and energy, or degrading their morale.

McCall Software Quality factors

Eleven quality factors were porposed, grouped according to three orientations or viewpoints (i.e., product operation, product revision and product transition). The factors are conditions or characteristics which actively contribute to the quality of the software. Factors 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 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.

Factors

1. Correctness: Extent to which a program satisfies its specifications and fulfills the user's mission objectives.

2. Reliability: Extent to which a program can be expected to perform its intended function with required precision.

3. Efficiency: The amount of computing resources and code required by a program to perform a function.

4. Integrity: Extent to which access to software or data by unauthorized persons can be controlled.

5. Usability: Effort required to learn, operate, prepare input, and interpret output of a program.

6. Maintainability: Effort required to locate and fix an error in an operational program.

7. Testability: Effort required to test a program to insure it performs its intended function.

8. Flexibility: Effort required to modify an operational program.

9. Portability: Effort required to transfer a program from one hardware configuration and/or software system environment to another.

10. Reusability: Extent to which a program can be used in other applications-related to the packaging and scope of the functions that the programs perform.

11. Interoperability: Effort required to couple one system with another..

Criteria

12. Traceability: Those attributes of the software that provide a thread from the requirements to the implementation with respect to the specific development and operational environment

13. Completeness: Those attributes of the software that provide full implementation of the functions required

14. Consistency: Those attributes of the software that provide uniform design and implementation techniques and notations

15. Accuracy: Those attributes of the software that provide the required precision in calculations and outputs

16. Error Tolerance: Those attributes of the software that provide continuity of operation under non-nominal conditions

17. Simplicity: Those attributes of the software that provide implementation of functions in the most understandable manner. (Avoiding practices that increase complexity).

18. Modularity: Those attributes of the software that provide a structure of highly independent modules

19. Generality: Those attributes of the software that provide breadth to the functions performed

20. Expandability: Those attributes of the software that provide for expansion of data storage requirements or computational functions

21. Instrumentation: Those attributes of the software that provide for the measurements of usage or identification of errors

22. Self-Descriptiveness: Those attributes of the software that provide explanation of the implementation of a function

23. Execution Efficiency: Those attributes of the software that provide for minimum processing time

24. Storage Efficiency: Those attributes of the software that provide for minimum storage requirements during operation

25. Access Control: Those attributes of the software that provide for control of the access of software and data

26. Access Audit: Those attributes of the software that provide for an audit of the access of software and data

27. Operability: Those attributes of the software that determine operation and procedures concerned with the operation of software

28. Training: Those attributes of the software that provide transition from current operation or initial familiarization

29. Communicativeness: Those attributes of the software that provide useful inputs and outputs which can be assimilated

30. Software System Independence: Those attributes of the software that determine its dependency on the software environment (operating systems, utilities, input/output routines, etc.)

31. Communications Commonality: Those attributes of the software that provide the use of standard protocols and interface rutines

32. Data Commonality: Those attributes of the software that provide the use of standard data representations

33. Conciseness: Those attributes of the software that provide for implementation of a function with a minimum amount of code

ÉCOLE POLYTECHNIQUE DE MONTRÉAL
3 9334 00228226 5