

Distr.
GENERAL

CES/AC.71/2003/3
22 January 2003

ENGLISH and FRENCH only

**UNITED NATIONS STATISTICAL COMMISSION and
ECONOMIC COMMISSION FOR EUROPE
CONFERENCE OF EUROPEAN STATISTICIANS**

**EUROPEAN COMMISSION
STATISTICAL OFFICE OF THE
EUROPEAN COMMUNITIES (EUROSTAT)**

**ORGANISATION FOR ECONOMIC
COOPERATION AND DEVELOPMENT (OECD)
STATISTICS DIRECTORATE**

Joint ECE/Eurostat/OECD meeting on the management of statistical information systems
(Geneva, 17-19 February 2003)

Topic I: Measures for the improvement of quality at the IT management level

SOFTWARE QUALITY FRAMEWORK

Invited paper

Submitted by Statistics Canada¹

***Abstract:** As Canada's National Statistical Office, Statistics Canada has the responsibility for providing statistics of the highest quality to its clients. In order to ensure confidence in the quality of the information it produces, Statistics Canada has many policies and guidelines to help employees achieve this goal. The use of high quality Information Technology (IT) solutions is an important component of this effort. Data quality has had much work done to describe its characteristics, but not much has been done in describing software quality. This paper provides an overview of software quality in Statistics Canada by presenting a general definition of software quality. It demonstrates the importance of quality characteristics and their dependencies. In addition, the paper describes the various perspectives of the stakeholders in software quality. It also shows how cost, schedule and product features impact the compromises made to software quality. The paper describes other influences on software quality (people, process, product and technology). We also consider techniques for improving the quality of the software developed and maintained at Statistics Canada.*

KEY WORDS: Software, Quality Improvement.

I. INTRODUCTION

1. Statistics Canada is in the information business. It does not manufacture products like cars or furniture, it produces statistical information. To be effective in the collection, analysis, processing and dissemination of this information, Statistics Canada utilizes a variety of information systems. An information system consists of:

- data (the descriptive, spatial, numeric or logical values in the system);
- metadata (data that describes other data in the system);
- documentation (plans, procedures, manuals to help make use of the information);

¹ Prepared by Matt Edgar, Mel J. Turner (Mel.Turner@statcan.ca) et al.

- people (the persons that perform tasks associated with the system);
- hardware (the technology for processing and storing the information);
- software (the information that instructs the hardware how to perform particular tasks).

2. The quality of all aspects of an information system is crucial for our survival:

Confidence in the quality of the information it produces is a survival issue for a statistical agency. If its information becomes suspect, the credibility of the agency is called into question and its reputation as an independent, objective source of trustworthy information is undermined. Therefore attention to quality is a central preoccupation for the management of a National Statistical Office.
(Source: Brackstone 1999)

3. As Canada's National Statistical Office, Statistics Canada has the responsibility for providing statistics of the highest quality to its clients. In order to ensure confidence in the quality of the information it produces, Statistics Canada has many policies and guidelines to help employees achieve this goal.

4. Data quality has had much work done to describe its characteristics. There are quality guidelines for the data we gather and the information we process (Statistics Canada, 1997a). *"The Quality Assurance Framework documents the processes in place in the agency to manage quality, within the organizational structure, and in face of risks and opportunities recognized by the agency"* (Statistics Canada, 2002). Unfortunately, not much has been done in describing quality for the other aspects of information systems. This paper focuses on software and the importance software quality has at Statistics Canada.

5. To provide high quality information, Statistics Canada makes use of substantial investments in information technology (hardware and software). We spend significant portions of our budget each year on the development and maintenance of software. Therefore, a clear understanding of what defines software quality is critical to effectively delivering quality software solutions. However, software quality is not an absolute. It means different things in different situations. Regardless, we should strive to achieve high software quality.

6. Section II of this paper defines software quality in general terms to set the foundation for further discussion. A summary of software quality characteristics and their dependencies, given in Section III, helps support this definition. There are many perspectives to software quality. Section IV describes several possible views and illustrates how these viewpoints relate to one another.

7. Software quality is subject to a variety of constraints:

- The cost of the software is important to an efficient and effective government department, like Statistics Canada.
- The nature of conducting surveys implies very strict schedules and timeframes for successful implementation.
- The complexity or degree of difficulty in developing software is inherent in the complexity of our problem domains and the ever-increasing demands for high-quality products that support rapidly evolving technologies for information gathering, transformation and dissemination.

8. The description of these constraints in Section V includes a discussion for the necessity of compromises and priorities when deciding on quality objectives.

9. Once a software project is underway, there are many influences on the resulting quality. Section VI details the influence of people, process, product and technology on the quality of the software delivered to the clients. Section VII provides a brief list of some of the techniques that are available to improve the software Statistics Canada develops and maintains.

II. DEFINITION OF SOFTWARE QUALITY

10. It has been said, "*quality is in the eyes of the beholder*" (Davis 1995). Experience shows that everyone has his or her own perspective as to what quality means. This makes it a difficult concept to define and measure. When asked to define quality of software, people typically describe a set of criteria to judge the quality (e.g. accuracy, flexibility, robustness and reliability). However, these are only *characteristics* of high quality software. Also known as attributes or dimensions, characteristics of software quality do not capture the essence of what defines software quality.

11. The definition offered by the IEEE² for quality is "the degree to which a system, component or process meets customer or user needs or expectations" (IEEE Std 610.12-1990). In the case of software, this implies that the measurement of quality is how well it meets the needs of the clients. However, determining the needs of the clients and measuring the needs of the clients are not easy tasks. Wiegers (1999) points out that users typically will have some implied expectations for a piece of software that weren't given in the stated requirements. Therefore, only satisfying the stated requirements isn't necessarily sufficient when attempting to meet the quality expectations of the users. Moreover, what of the developers? Certainly, they too have say in what constitutes a piece of quality software. After all, it is their job to construct the best possible solution to users' problems.

12. With these ideas in mind, this paper provides the following definition of software quality:

Software Quality is the degree to which the software satisfies the stated and implied requirements of all stakeholders.

13. This definition contains the essence of the IEEE definition of quality and parallels the definition of data quality adopted by Statistics Canada. It also remains true to the idea that satisfying the stated and implied requirements are both necessary to meet the needs of the clients. The users, developers and all other stakeholders have their own perspective and therefore have a say in describing the characteristics of software quality.

14. The characterization of software quality expectations (both stated and implied) can be quite easy. There seems to be a never-ending list of characteristics for software quality. The challenge is to choose those characteristics that are most important to the software stakeholders.

III. SOFTWARE QUALITY CHARACTERISTICS

15. Statistics Canada's mandate is to "collect, compile, analyze, abstract, and publish information on the economic, social and general conditions of the country and its citizens" (Statistics Canada 2000). To successfully fulfill this mandate, the methodologists and economists at Statistics Canada have the responsibility to ensure that the information they produce is of the highest quality. The Quality Assurance Framework (Statistics Canada 2002) outlines these objectives very clearly and also introduces the concept of defining data quality by defining six dimensions of data quality. Table 3-1 contains a summary of these six dimensions

Table 3-1 The Six Dimensions of Data Quality

Relevance	The relevance of statistical information reflects the degree to which it meets the real needs of clients.
Accuracy	The accuracy of statistical information is the degree to which the information correctly describes the phenomena it was designed to measure.

² IEEE - Institute of Electrical and Electronics Engineers

Timeliness	The timeliness of statistical information refers to the delay between the reference point (or the end of the reference period) to which the information pertains, and the date on which the information becomes available.
Accessibility	The accessibility of statistical information refers to the ease with which it can be obtained from the National Statistical Office.
Interpretability	The interpretability of statistical information reflects the availability of the supplementary information and metadata necessary to interpret and utilize it appropriately.
Coherence	The coherence of statistical information reflects the degree to which it can be successfully brought together with other statistical information within a broad analytic framework and over time.

(Source: Brackstone 1999)

16. These dimensions provide an excellent starting point for the list of characteristics that can describe software quality. These dimensions tend to focus on the external characteristics rather than internal characteristics. External characteristics deal with how the users of the software view its quality. Internal characteristics represent the attributes of software quality that reflect how well a software product is constructed. In *Peopleware* (DeMarco 1999), the point is made that internal quality standards are quite high when set by the developers. This suggests that their perception of software quality (internal) is different than that of the other stakeholders (external).

17. The *Information Technology Framework* (ITF) (Statistics Canada 1996) was one of the more recent documents to address software issues within our organization. It outlines a list of strategic goals the Informatics Branch viewed as important in providing a sound informatics infrastructure for Statistics Canada. The ITF outlines goals of availability, interoperability, usability, reliability, security and flexibility for software solutions. At a high level, these goals reflect the needs of a national statistical office with respect to effectively leveraging its investments in informatics. It is a good mix of both user and developer characteristics, but still focuses at an external level.

18. With no other Statistics Canada guides, it is challenging to determine the characteristics that are important to our developers. Therefore, we must look to the software industry for guidance. James McCall wrote an early discussion in the software industry on the factors of software quality while working for the General Electric Company (McCall 1977, Walters & McCall 1979). Often called the "GE Model" or the "McCall Model", it focuses on software developers for use during the development process. *Characteristics of Software Quality* (Boehm 1978) is another early attempt in the software industry at defining software quality. Frequently referenced, this book describes many quality attributes for software.

19. The ISO 9126 standard (ISO/IEC 9126:1991) describes a list of six dimensions to software quality. It is a product evaluation standard concerned with eliminating confusion between the users and developers. More recently, the *Guide to the Software Engineering Body of Knowledge-SWEBOK* (IEEE 2000) has adopted the ISO 9126 standard for describing the knowledge area of software quality. Other sources (McConnell 1993, Wiegers 1999) give their own set of characteristics from the developer's point of view.

20. Table 3-2 attempts to summarize these attributes. Some are run-time (user) attributes, and others are build-time (developers) attributes. Some of these will be of greater importance to some stakeholders, depending on the situation. Section IV will explore the different perspectives that people have when deciding on importance. Section V will then describe the effect that constraints have on the compromises and the priorities made to these software quality characteristics.

Table 3-2 Summary of Software Quality Characteristics

Attribute	Build / Run	Definition
Adaptability	Build	The ease of making changes to software required by changes in the operating environment.
Applicability	Build	The degree to which software can be used, without modification, in applications or environments other than those for which it was specifically designed.
Availability	Both	The ease with which the software can be made available, be obtained or be installed.
Clarity	Build	The ease with which design and construction of a system is understood by a developer or maintainer. Clarity has more to do with the low-level comprehension than understandability, which is at a higher level. Includes the presence of supplementary information and metadata necessary to interpret and utilize the software.
Compliance	Build	The degree to which the software complies with standard interfaces (de jure or defacto standards) such that it can interoperate with other software applications. Also referred to as interoperability.
Correctness	Build	The extent to which software is free from faults in its specification, design and implementation.
Maintainability	Build	The effort needed to make modifications, including corrections, improvements or adaptation of software to changes in requirements and functional specifications.
Performance	Run	The level of performance (execution time) of the software and the amount of resources used (memory, storage), under stated conditions. Also referred to as efficiency.
Portability	Build	The ability to transfer the software from one organization or hardware or software environment to another.
Precision	Run	The degree to which the software correctly functions as it was designed to. Precision differs from correctness; it is a determination of how well software does the job it was created for rather than whether it was built correctly.
Punctuality	Build	Was the software acquired or implemented on time? Although not necessarily an internal attribute of the software, it may be critical as a measure of a development service.
Reliability	Run	The extent to which software can be expected to perform its intended function with required precision.
Reusability	Run	The ease with which you can use parts of a system in other systems or a different context.
Robustness	Run	The capability of software to maintain its level of performance in the presence of invalid inputs or a stressful system environment.
Security	Run	The extent to which access to the software or data by unauthorized persons can be controlled. Also referred to as integrity.
Suitability	Run	The degree to which the software meets the needs of the clients; or its fitness for use.
Testability	Build	The ease with which the software can be tested to find defects and ensure that it meets specifications.
Understandability	Run	The ease with which you can comprehend a system of software.
Usability	Run	The ease of use of the software.

(Sources: Boehm 1978, ISO 1991, McConnell 1993, STC 1996, Walters & McCall 1979, Wiegers 1999)

A. Quality Framework

21. The following framework relates the attributes drawn from the software literature to the quality dimensions used to describe our statistical output, but retains the distinction between the perspectives of developers (build-time attributes) and the users (run-time attributes). The six quality dimensions are redefined as follows to refer to software quality:

- *Relevance:* As software is the automation of some theory, algorithm or process, relevance is a measure of how applicable the software is to a business solution, or how closely its function fits a particular purpose. A custom-designed application is presumably highly relevant while a generalised system may be less relevant in a particular case.
- *Accuracy:* The software implementation may be judged in terms of its *correctness* or how closely it adheres to the theory or specification. From a user perspective it is the *precision* or accuracy of the outcome that is important (which may depend in part on the quality of the input, not just the correctness of the software).
- *Timeliness:* Software is timely if it can be delivered when it is needed and then performs at a rate appropriate to its application. These are both measures of *availability* but from different perspectives.
- *Accessibility:* Accessibility refers to how easily the intended audience can locate and employ the software technology. This might include attributes such as commercial availability, the presence of business practices such as corporate licensing, or ensuring that the training, tools and practices associated with this technology are widely available. From a user perspective the focus is on the security and reliability aspects of access, rather than *usability* which is covered next.
- *Interpretability:* Can the software be clearly understood and applied by the intended audience? The developer and user perspectives differ significantly in this dimension. For the developer, the clarity of the documentation and software code is paramount in supporting maintenance and enhancement activity. For the user, ease of learning and usability are the key attributes.
- *Coherence:* This dimension refers to the level of standardisation adopted by the software and its application. If software is compliant with industry standards it is likely to be more portable across technical environments, interoperable with other software components, and more reusable in different business contexts. Reuse has a positive influence on quality because, as software is used more widely, it becomes more robust and error-free. From a user perspective, widely used software also means that user skills are transferable and need for specialised training is reduced.

22. The correspondence of descriptors is summarised in the following table. This correspondence is not exact and the table also contains some positive and negative indicators of cross-influence or interdependence. The implication of these for quality trade-offs is covered in the next section.

Table 3-3: Correspondence of Software Quality Characteristics

Software Quality Attributes	Framework Characteristic						Comments
	Relevance	Accuracy	Timeliness	Accessibility	Interpretability	Coherence	
Build-time Attribute							Developer perspective
Applicability	l			s	s	t	Solves a specific problem.
Correctness		l				s	Meets specifications.
Punctuality, Availability			l	s			Completed on time, on budget.
Adaptability, Testability			t	l		s	Professionally engineered.
Maintainability, Clarity	s			s	l	s	Enduring value.
Portability, Compliance	t		t			l	Adopts industry standards.
Run-time Attribute							User perspective
Suitability	l			s	s	t	Fitness for purpose.
Precision		l				s	Data can be trusted.
Performance, Availability			l	s			Assists getting job done.
Reliability, Security			t	l		s	Software can be trusted.
Understandability, Usability	s			s	l	s	Easy to learn and use.
Reusability, Robustness	t		t			l	Skills are applicable elsewhere.
l	<i>Primary attribute/characteristic.</i>						
s	<i>The attribute supports or enhances the framework characteristic.</i>						
t	<i>The attribute may conflict with or detract from the framework characteristic.</i>						

23. Although we can define many attributes of software quality, they will not always be complementary to each other. McConnell (1993) correctly states that the attempt to maximize certain characteristics will cause conflict with the attempt to maximize others. For example, high-performance software typically achieves this quality by using techniques that reduce its portability. Generic tools improve software coherence but may not be as relevant to a particular application. These types of conflict are commonplace with most software projects.

24. However, other attributes complement one another quite well. A solution that is relevant to the user typically reflects high degrees of accessibility. How understandable a piece of software is will have a positive impact on how coherent it is for the user. From the developer perspective, testability improves coherence but achievement of it may hurt timeliness.

IV. PERSPECTIVES

25. It is possible to describe software quality by a variety of characteristics, as outlined in the previous section. Therefore, it is perhaps inevitable that quality can be “classified according to a number of views or perspectives” (Gillies 1992, pg. 9). Each stakeholder often develops these perspectives of a software product. One could classify the stakeholders as either 'users' or 'developers'. However, this scheme is perhaps too restrictive for the needs of Statistics Canada. In practice the environment in which we work is more diverse and needs a better method to describe it.

26. The following are classes of stakeholders typically involved in a software project at Statistics Canada. The list has been adapted from similar lists in *Software Quality: Theory and Management* (Gillies 1992, Section 1.3) and *ITF: Project Management* (Statistics Canada 1993, Section 3).

Project Sponsor. The Project Sponsor (or Program Manager) is the individual, committee or division that is funding the software project. The project sponsor is usually the director of the division for the resulting software solution. They will often be remote from the day-to-day implementation issues and will want a *successful* outcome to justify their expenditure. They likely will judge a successful outcome by on-time delivery within budget as much as on broader quality issues.

Project Manager. The project manager has responsibility for the project deliverables/objectives. They are keen to produce a software product that is reliable and maintainable and will keep the customer satisfied. However, they have deadline constraints and budget constraints that will exert a force for compromise.

IT Project Leader. On most software projects, an IT project leader is assigned. The IT project leader is responsible for overseeing day-to-day operations of the developers. They ensure the developers know the functional and technical requirements for the software project. They also assist the Project Manager by liaising with the customers to better understand their needs and priorities.

Developer. The developer is the person who builds the software.

End User. The end users sometimes have very little input into the development process. However, they are the ones that have to use the delivered software product so should be accorded the opportunity to influence the external characteristics and features of the product. Ultimately, their satisfaction will determine the success of the investment in a software project.

27. Each of these stakeholders has their own perspective of software quality and these views may be in conflict with one another.

I once attended a post mortem meeting for a recently completed survey-processing project. The first question was, "Who considers this project a success?" One of the line managers spoke up quickly "Yes, it was a great success! We delivered the software on time." From his perspective, that was the highest priority. However, one of the developers invited to listen in, could not agree. She responded, "How can you say that? The software is a mess and there is no documentation at all. The design is not adaptable to next year's changes. There are over 30,000 lines of code and most of it is terribly hard to understand." Her perspective was completely different. She didn't care that the software delivery was on time. Her priority was to ensure the system would be maintainable and reusable for next year's surveys. As the people around the table all had their say, the discussion focused on other aspects of the project. These too were viewed as successes by some and as failures by others. A consensus was possible, but only after much deliberation.

28. This scenario illustrates just how differently each stakeholder views software quality. It is important to appreciate that each viewpoint is valid and simply from a different perspective than the others. To help understand what characteristics are important to each class, Table 5-1 shows a profile of important qualities by stakeholder. It is not the only profile possible, but it is a typical one. It is worthwhile to understand these expectations near the start of a project when they can be managed.

Table 5-1 Typical Software Quality Priorities

	Project Sponsor	Project Manager	IT Project Leader	Developer	End-User
Relevance	Suitability		Applicability		Suitability
Accuracy		Precision	Correctness	Correctness	Precision
Timeliness	Punctuality	Punctuality	Availability	Performance	Availability
Accessibility		Adaptability	Testability	Security	Reliability
Interpretability	Clarity		Maintainability		Usability
Coherence	Robustness	Reusability	Portability	Compliance	Reusability

V. CONSTRAINTS

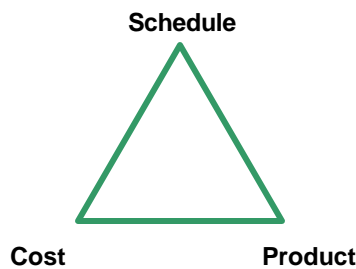
29. As mentioned, project stakeholders must decide where tradeoffs will occur and what priorities to assign to characteristics ranked in order of importance.

30. The obvious constraints of schedule and cost are ever apparent for IT projects at Statistics Canada. McConnell (1996) presents a compelling argument that a trade-off triangle of cost, schedule and *quality* is not as appropriate as a triangle of cost, schedule and *product*. Product in this case represents quality and all other software attributes including features, complexity, defect rate, etc.

31. To maintain the triangle one has to balance schedule, cost and product. Many software projects are in support of a statistical program (i.e. a survey of some type). It is the very nature of surveys that they be conducted at very specific times. Whether a monthly, quarterly or annual survey, or part of the Census, this rule holds fast. For the software that supports these surveys, it is virtually impossible to compromise the schedule. Other software projects allow for schedules that are more flexible. They can survive significant changes to their schedules without adversely affecting the success of the project.

32. The *reusability* of software components is particularly important in balancing the constraint triangle. By reusing an existing component it is possible to contain costs, respect tight deadlines and provide significant function. Even in those cases where the requirements have to be modified to match the reusable component it is often a worthwhile tradeoff.

Figure 5-1. Software constraint triangle



(Source: McConnell 1996)

33. The source of funding of software projects at Statistics Canada is sometimes a factor in determining the potential cost compromises. With an unlimited budget, the quality of software would be much higher. However, as a responsible government department, we have an obligation to taxpayers to make effective use of available funds.

34. Some software projects have fixed budgets that span part or all of a fiscal year (April - March), whereas other projects can span several years. Some projects obtain budgets from specific sources and must use the funding to complete particular objectives. Others obtain budgets from larger pools of funding and have much more control on the amount and timing of resources spent on a software project.

35. In most cases, given fixed funding and time, it is still possible to produce high quality software. Software quality requires planning as part of the development process in the same way that data quality requires planning as part of the methodological design. The same constraints exist in both cases: cost and schedule. High quality is possible, if one is willing to try new practices based on sound and proven principles. However, they require commitment from all stakeholders, not just a few people advocating their use.

A. Tradeoffs and Priorities

36. By knowing the software quality constraints, it is necessary to identify those quality characteristics that are of the highest importance to the project stakeholders. Since they all have different (and sometimes conflicting) perspectives, this identification process results in a set of quality characteristics that best suits the overall needs of the IT project. By respecting all viewpoints, no one feels left out and everyone stays motivated to see that the project reaches its quality goals.

37. By assigning priorities to certain characteristics, the stakeholders will have a clearer picture of what to expect out of the delivered software product. A deadline known in advance (e.g. the mail-out date for a survey) makes it possible to determine where tradeoffs in the software product could occur. For example, it may be possible to scale back some features and give a lower priority to some of the efficiency and robustness of the solution. If an IT solution needs to be very maintainable (e.g. in support of known future enhancements), then it should place higher priority on readability and testability. If the end-users are not familiar with the software (e.g. a new time accounting system) then usability and security become important. If the people responsible for the software development are leaving, then the software's interpretability (e.g. notes, design specifications, user manuals, and file layouts) is important to ensure its sustainability.

38. Software project stakeholders must address these concerns with constraints and with the necessary tradeoffs and priorities. They will be in much better positions to know what quality characteristics will be present in software products.

VI. INFLUENCES ON SOFTWARE QUALITY

39. Once a project has begun, many influences affect the quality of the resulting software product. No matter the type of project, it has influence from four important factors: people, process, product and technology:

People perform quickly or they perform slowly. The process leverages people's time, or it throws up one stumbling block after another. The product is defined in such a way that it almost builds itself, or it is defined in a way that stymies the best efforts of the people who are building it. Technology assists the development effort, or it thwarts developer's best attempts.
(Source: McConnell 1996)

40. It is possible, but not desirable, to focus on one influence and ignore the others. However, it is better to treat all four influences as dimensions and manage them effectively to ensure the quality in the software.

A. People

41. The people that staff a software development project have a large influence on how successful it is. There is so much involved in this topic this paper cannot do it justice (see the Further Reading Section). However, in short, the activities of staff selection, team organization and motivation can have a huge impact on the outcome of a software project. In addition, it is necessary for staff to be aware and knowledgeable of software quality assurance (SQA) issues, practices and methods.

B. Process

42. The processes used to develop and maintain software products have been studied at great length. The Software Engineering Institute (SEI), the Project Management Institute (PMI), the Institute of Electrical and Electronics Engineers (IEEE), as well as other organizations have examined these processes and have documented evidence that supports their use. The key process areas (KPA) of the Capability Maturity ModelSM (CMM) (SEI 1993) detail a set of processes that help improve the quality of software products. Some of the more effective processes in the CMM are:

- Software Quality Assurance
- Requirements Management
- Peer Reviews
- Software Project Planning
- Project Tracking and Oversight
- Software Configuration Management

43. In some instances, Statistics Canada performs many of these processes well. However, there is a need for more standardization, training and sharing of experiences. The goal of the STC Software Best Practices Group is to address these concerns.

C. Product

44. The size of the software product has the most evident influence on its quality. If the software project is large, quality is harder to achieve. On smaller projects, quality is more straightforward to achieve. With fixed cost and schedule, it is usually the size and scope of a software project that require adjustment in order to complete the project successfully. Good estimates on the product size help ensure that a realistic set of features is determined for the software product.

D. Technology

45. It is always necessary to choose the right tool for an IT project. The developers and IT Project Leaders are typically responsible for making this choice because they have the most experience with information technology. Choosing a more effective tool or technique can greatly increase a developer's productivity and software quality as well. It is useful to consider technologies such as Visual Basic, Java, Object Oriented methodologies and software reuse. Using standard approaches across the entire agency (e.g. more use of SAS and less software diversity) helps developers use their skills on many different types of software projects. More productive development technologies give developers more time to ensure the achievement of the software quality goals.

VII. TECHNIQUES FOR IMPROVING SOFTWARE QUALITY

Software quality assurance is a planned and systematic program of activities designed to ensure that a system has the desired characteristics. Although it might seem that the best way to develop a high quality product would be to focus on the product itself, in software quality assurance the best place to focus is on the process.
(McConnell 1993)

46. It is important to know where to focus our efforts when attempting to achieve high quality software. As McConnell states, we should focus on the process, not necessarily the product. This section describes some techniques that assist in improving software quality. The descriptions only provide an overview of the techniques. Further detail is available in the books referred to in the Further Reading Section.

47. **Software quality objectives.** Setting explicit quality objectives implies that everyone involved with the software project knows what the goals are! If people know what the expected goals are (and they are reasonable), chances are their achievement will be possible. Explicitly identifying all stakeholders and

knowing their quality priorities is an important step in any software project. It is a powerful technique that is very simple to use at the beginning of a software project and revised when necessary.

During a preliminary meeting for one of my recent projects, the project manager informed the stakeholders that the project had to reuse the existing software as much as possible. The project sponsor didn't want to 'reinvent the wheel' and wanted to respect the current conceptual data model wherever possible. By placing a high priority on reusability, it allowed everyone in the design process to consider all options with this priority in mind. Many designs were contemplated, however the one chosen for the project was the one that reused a good portion of existing components and concepts. Knowing the quality goals allowed people to respect them and incorporate them into their work.

48. **Explicit quality assurance activity.** Assuring software quality has to be made a priority. When management invests in an explicit quality assurance process, it makes it clear to all project stakeholders that quality is a priority. The wealth of knowledge for quality assurance is considerable. Many techniques assure quality in the development process. They simply need to be learned, practiced and refined.

49. **Testing strategy.** All too often, testing falls on the shoulders of the developers and users to perform at their discretion. Developers rarely make effective testers of their own work because they simply can't be objective enough to find and expose errors. Users make effective testers but unfortunately, they too can miss defects. Often they focus on cases that are only important to them, but these test cases are not exhaustive. Consequently, many errors still creep into the software undetected. In the worst cases, developers do little testing and have the attitude that it is the user's job to do the testing! However, some projects have a testing strategy that they follow throughout the project. It allows them to itemize all the cases that need to be tested, identify who is responsible for the tests and to then document the results. The delivery of these software projects usually consists of much higher overall quality.

50. **Software engineering guidelines.** The body of knowledge associated with software engineering is staggering. However, there are some fundamental concepts and practices whose knowledge is necessary (regardless of the project or technology). These guidelines apply to the entire development process and include naming conventions, requirements analysis, system design, software construction and system testing. Providing your development team with these guidelines greatly improves their productivity and the resulting software quality.

51. **Technical reviews.** Errors found early are easier (and cheaper) to deal with. Two of the most effective methods for improving software quality are informal and formal technical reviews. Informal reviews performed by the developers can identify obvious errors and omissions before a formal review is necessary. Formal reviews also perform this function, but they have the added benefit of identifying critical errors long before they cause a problem for the other stakeholders. Besides error reduction, technical reviews also provide a mechanism to improve the source code itself. Reviews can improve the quality of the source code in terms of various characteristics of quality such as modularity, cohesion and coupling, style, readability, generality of interfaces, and so on.

52. **Risk management.** Managing the risks that can occur in a software project has a positive impact on the software quality. If risks are identified and handled in a reliable and predictable manner, the resulting software will be reliable and predictable as well.

53. **Change control procedures.** Uncontrolled changes can degrade software quality quite easily. Many different procedures used by Statistics Canada developers and IT project leaders reduce the impact that changes have on the software. The majority of the projects we initiate are small enough to manage changes informally with satisfactory results. However, some larger projects could easily collapse if it were not for the formal change control procedures in place. Depending on the size and complexity of your project, a known and documented change control procedure can be a lifesaver.

54. **Measurement of results.** The quality of the software depends on the estimates used to determine the size of the software and effort required when developing it. These estimates are more accurate when based

on prior experience. At Statistics Canada, many people have been working on a project for a number of survey cycles. They usually provide quite good estimates. However, we are now working in an environment that encourages professionals to participate in developmental and rotational assignments so we need better ways to record and communicate experience. Measuring and recording the results of software development projects helps guard us against the problems related to high turnover of staff.

55. **Prototyping** Practiced frequently at Statistics Canada, prototyping a software product early in its development gives the stakeholders a first look at the attempted solution. It is extremely useful in gathering feedback from the users and better focusing the software requirements. Its only drawback is that it is too easy to think of a prototype as the final product. These prototypes should not require much effort to construct and should be thrown away once they have served their purpose. The knowledge gained in the prototyping process is then used to create the actual delivered software product.

56. By reviewing this list of techniques, one should have made the following observations:

- there are many techniques at one's disposal that can improve software quality,
- these techniques are all sensible approaches to software development and maintenance,
- it is possible to try some of these immediately without the aid of fancy (or expensive) tools, and
- improved software quality is achieved more through people and process, than through product and technology.

VIII. CONCLUSION

57. The objective of this paper has been to provide an overview of software quality at Statistics Canada. A number of characteristics support the definition of software quality and help better quantify the quality objectives of the Agency's software development and maintenance projects.

58. Software quality has many different perspectives. From the project sponsor to the end-users, everyone has his or her viewpoint. Respect for these viewpoints is necessary when deciding what characteristics are important to the overall software quality.

59. Many conflicts can occur when attempting to develop and maintain software of high quality. Fixed costs and schedules make stakeholders assign priorities to some characteristics and accept the compromises that result.

60. The influences of people, process, product and technology can be used to your advantage to deliver a high quality software product. Learning to leverage all you can out of these influences is a great asset when achieving the quality goals of IT projects.

61. Improving software quality requires the use of sound and practical techniques. There are many such techniques at our disposal that can have an impact immediately and with minimal effort.

IX. FURTHER READING

A. References

Boehm, B. [et al]. 1978. *Characteristics of Software Quality*. New York: North Holland. Chapter 3.

Brackstone, G. 1999. "Managing Data Quality in a Statistical Agency." *Survey Methodology* (Statistics Canada Catalogue no. 12-001-XIB). Ottawa: Minister responsible for Statistics Canada, (December 1999): pg. 139-149.

DeMarco, T. and Lister, T. 1999. *Peopleware: Productive Projects and Teams, 2nd ed.* New York: Dorset House. pg. 19-22.

Gillies, A. 1992. *Software Quality: Theory and Management*. London: Chapman. pg. 4-29.

Institute of Electrical and Electronics Engineers (IEEE). 2000. *Guide to the Software Engineering Body of Knowledge (Stone Man Version 0.7)*. <http://www.swebok.org/>: Released April 2000. (Accessed October 2, 2000).

McCall, J. [et al] 1977. *Factors in Software Quality*. NTIS AD-A049-014, Rome Air Development Center.

McConnell, S. 1993. *Code Complete*. Redmond, WA: Microsoft Press. pg. 557-571.

McConnell, S. 1996. *Rapid Development*. Redmond, WA: Microsoft Press. pg. 11-17, 126-127.

Software Engineering Institute (SEI). 1993. *Key Practices of the Capability Maturity Model, Version 1.1* (CMU/SEI-93-TR-25). Carnegie Mellon University. Pittsburgh, PA.

Statistics Canada. 1993. *Information Technology Framework (ITF): Project Management*. Informatics Branch. Released October 1993.

Statistics Canada. 1996. *Information Technology Framework (ITF)*. Informatics Branch. Released March 1996.

Statistics Canada. 2001. *Information Technology Framework (ITF)*. Informatics Branch. Released June 2001.

Statistics Canada. 2002. Quality Assurance Framework. Methods and Standards Committee, Statistics Canada.

Statistics Canada. 1998. *Statistics Canada Quality Guidelines: Third Edition - October 1998* (Statistics Canada Catalogue no. 12-539-XIE). Ottawa: Minister responsible for Statistics Canada: pg. 74-80.

Statistics Canada. 2000. *The Statistics Act, section 3a*. Internal Communications Network (ICN). http://www.statcan.ca/10/10_006_e.htm: Last Modified February 10, 2000, STC Intranet. (Read October 13, 2000).

Walters, G. and **McCall, J.** 1979. "Software Quality Metrics for Life Cycle Cost-Reduction." *IEEE Transactions on Reliability*. Vol. R-28, No. 3 (August 1979). pg. 212-219.

Wiegars, K. 1999. *Software Requirements*. Redmond, WA: Microsoft Press. pg. 195-206.

B. Peopleware Issues:

Constantine, L. 1995. *Constantine on Peopleware*. Englewood Cliffs, NJ: Yourdon Press

DeMarco, T and **Lister, T.** 1999. *Peopleware: Productive Projects and Teams, 2nd ed.* New York: Dorset House.

C. Testing Strategies:

Myers, G. 1979. *The Art Of Software Testing*. New York, Toronto. Wiley.

D. Software Reviews:

Yourdon, E. 1985. *Structured Walkthroughs*. New York, NY: Yourdon Press

E. Software Development Techniques:

McConnell, S. 1993. *Code Complete*. Redmond, WA: Microsoft Press.

McConnell, S. 1996. *Rapid Development*. Redmond, WA: Microsoft Press.
All the books listed above are available from the Statistics Canada Library (<http://lib2.statcan.ca/>). The Library maintains an Information Technology Collection (ITC) that contains a wealth of books (some of which are available in French as well) on developing quality software. Their assistance in researching this paper was invaluable.