Implementing Quantitative SQA: A Practical Model

Victor R. Basili *and* **H. Dieter Rombach**, *University of Maryland*

Considering the product is not enough in SQA — you must also consider the development process. This model addresses both needs.

Because software affects more and more aspects of our life, the cost-effective development and maintenance of high-quality software is increasingly important. Software quality assurance (SQA) has become an indispensable dimension of software development, designed to guarantee that quality and productivity requirements are fulfilled.

We put this special issue together to help people understand the increasing importance of SQA as an essential part of software projects, outline some new ideas and approaches to SQA, and report on some practical experiences.

SQA's importance. Frequently, cost-effectiveness and high quality are considered to be conflicting goals in software development. In the past — and very often today — this conflict was resolved by schedules that favored cost-effectiveness over quality.

Such inability to cope with all requirements is characteristic of a new, immature field. But the software development and maintenance field is now about 30 years old. As part of a constant maturing process, we have developed better approaches and techniques to develop and maintain software, including better approaches and techniques for SQA. Three changes reflect the increasing maturity of our field:

- The business of software development and maintenance has become increasingly competitive. This requires software projects both to be cost-effective and to produce high-quality products to compete in the marketplace.
- Today's software applications are more complex. Software failures can result in financial damage and even threaten the health or lives of human beings. Financial, transportation, air-traffic control, and medical applications demand high-quality software. Such applications have two additional problems: (1) It is not enough for the developer to be convinced that certain quality requirements are met; the customer and users also must be convinced that *all* requirements are met. (2) The issue of legal liability means that, in the case of a fatal failure, the software developer must be able to prove that the development and maintenance process was performed according to state-of-the-art standards.
- The attitude of customers and users toward quality has changed. As the field of software development and maintenance has matured, so have customers and users, who today expect a higher level of product quality. Customers are less willing to

accept software products that violate explicit or implicit quality requirements.

These changes mean that software development must deal simultaneously with demands for better quality and higher productivity. SQA is supposed to guarantee that project-specific quality and productivity requirements are fulfilled.

SQA's scope. SQA, then, must be concerned with productivity (for example, cost and schedule), process quality (which in this context includes all development and maintenance activities), and product quality. SQA must address two classes of requirements: external and internal.

External quality and productivity requirements should be stated explicitly in the project's requirements document. They reflect the customer's criteria for deciding the success or failure of a software project. Examples of external requirements are process productivity (such as schedule, personnel, computer resources), process quality (such as methods, tools, guidelines), and final product quality (such as reliability, response time, documentation). Meeting external requirements is the main goal of all project activities.

Internal quality and productivity requirements are usually added by the company. These requirements address the long-term improvement (beyond the current project) of the developer's competitiveness in the marketplace. For example, a company may require that certain guidelines (or norms) be followed to develop reusable software. Meeting those internal requirements may not be important to the customer of the current project, but might increase the productivity of the development environment in future projects.

SQA departments must establish productivity and quality requirements of both classes and guarantee that those requirements are met.

SQA's four W's. SQA can be characterized by four "W" questions:

1. *What* must we assure? We must identify the quality characteristics of interest, state the relationships and weights among those characteristics, and define the degree to which they must be assured. This requires close interaction with the customer and should be part of the project's requirements document. The fulfillment of qualitative characteristics, such as reliability and performance, cannot be guaranteed unless they are quantifiable. For example, "reliability" might be quantified as "mean time between failures in hours." These quantitative entities are called measures.

Today, customers and users expect a higher level of product quality.

2. *When* must we assure? We must define at which project milestones the quality characteristics of interest should be controlled, to assure that the established requirements can be met, to discover that they cannot be met, or to take corrective action as early as possible.

3. *Which* methods and tools must we use? We must define which methods and tools are most adequate to gather the information that supports sound SQA activities. Today, the most common methods and tools are testing and reading techniques.

4. *Who* must do the assurance? We must decide what kind of professionals (their qualifications and relation with other project personnel) are most suited to do an effective quality assurance job and how they fit into the overall project organization.

Measurement. The effectiveness of SQA models and supporting methods and tools depends on whether they can be tailored to the specific needs and characteristics of a project and on whether both the development and maintenance processes and the resulting products are tractable. In this context, measurement is a very powerful mechanism for defining and analyzing software process and product quality. However, measurement must be goal-oriented: It must be driven by the overall objectives of SQA.[1,2]

One of our objectives in this special issue is to address realistic ways to use measurement in SQA. Elsewhere, we have provided an operational framework for using measurement in SQA.[3] Our model stresses the need for objective and subjective, as well as direct and indirect, measures.

Objective measures are numerical expressions (numbers, sums, ratios, and distributions) or graphical representations of numerical expressions that can be computed from software documents such as source code, designs, and test data. "Objective" means two people should compute the identical value independently. Subjective measures are relative, based on an individual's estimation or a compromise within a group. Typical subjective measures are "degree to which a method was used" or "experience of per-

sonnel with respect to the application."

Direct measures allow a project-specific quantification (and definition) of a quality factor of interest. An indirect measure helps predict the expected value of a direct measure. For instance, a meaningful direct measure for "operational reliability" might be "number of failures per week of operation" or "number of failures per call of some system component." The indirect measure "number of failures during the preceding acceptance test" might be useful for predicting operational reliability as defined by the direct measures already during development. Other meaningful indirect measures may include product measures such as complexity.

Indirect measures can contribute valuable information to SQA activities. Knowing the relationship between direct measures and indirect measures for a quality characteristic lets us predict whether requirements with respect to that characteristic can be fulfilled and, in turn, correct development when necessary.

Quantitative SQA. Our model for quantitative SQA consists of three phases, all of which suggest some kind of measurement:

1. Define quality requirements in quantitative terms. We must select the quality characteristics of interest, define priorities among and relations between those quality characteristics, define each characteristic by one or more direct measures, and define the quality requirements quantitatively by assigning an expected value to each measure.

2. Plan quality control. We must plan adequate actions to assure the fulfillment of the defined quality requirements, control the proper execution of these actions, and evaluate their results. Planning includes defining what criteria might control the quality characteristics of interest, how these criteria are quantifiable in terms of indirect measures, how these indirect measures can be used for prediction and control, when and how the data needed for computing all measures should be collected, and what methods and tools should be used. Selecting appropriate indirect measures requires that we have sound knowledge of the project's particular development or maintenance process.

3. Perform quality control. This activity consists of two parts: (1) measurement, in which the methods and techniques specified during the planning phase are applied to gather the actual values for all defined direct and indirect measures and distributions, and (2) evaluation, in which the direct measurements are compared to the quality requirements and indirect measurements are interpreted to explain or predict the values of direct measures. Evaluation also involves deciding if the requirements were met for each quality characteristic and for the entire set of project requirements.

We believe productivity increases automatically if a high-quality development process is used.

An important attribute of our quantitative SQA model is its consideration of the quality of the process, not just of the product. One reason quality and productivity are perceived as conflicting is that process quality is often neglected. We believe productivity increases automatically if a high-quality development process is employed. For SQA, this means that it is not enough to check whether the developed products are of the desired quality — to improve quality and productivity it is just as important to evaluate the impact of the methods and tools used to meet (or fall short of meeting) those quality requirements.[1]

Our model also accounts for the equal importance of analytic and constructive SQA activities. The term "assurance" (as opposed to analysis) indicates that the objective is both to determine if quality requirements are met (the analytic aspect) and, when they are not met to suggest corrective actions (the constructive aspect). Quality can only be achieved by undertaking appropriate constructive actions. Constructive actions might be suggested to meet the desired quality requirements for the current project or so the company can improve the quality of future projects.

Our model also covers all phases of development and maintenance. This is especially important for suggesting effective corrective quality assurance actions. The earlier in the software process that quality problems are detected or anticipated, the more effective the countermeasures can be.

Finally, our model stresses the importance of separating responsibilities for development and SQA. Defining quality requirements, planning quality control, and performing the evaluation part of quality control should be conducted by development-independent personnel. According to our model it is not important *who* performs the measurement part of quality control as long as it is planned for and evaluated by development-independent personnel.

Implementing this independence of SQA can be done in many ways, ranging from contracting two independent companies (one in charge of development, the other of SQA) to using different development groups that perform SQA for each other.

About this issue. This issue is not intended to give you an overview of SQA. Many other publications, including works by Boehm[4] and Buckley and Poston,[5] are available for that purpose. The articles we selected for this issue present new approaches and ideas for successful SQA and report on the applications of some SQA techniques in industry. ⊹

Acknowledgments

We thank the authors and reviewers who helped to make this issue a reality. Special thanks to Bruce Shriver, who initiated this special issue while he was editor-in-chief of *IEEE Software*, and to Ted Lewis, the current editor-in-chief, for his constant guidance, encouragement, and help in the review process.

References

1. V.R. Basili and H.D. Rombach, "Tailoring the Software Process to Project Goals and Environments," *Proc. Ninth Int'l Conf. Software Eng.*, CS Press, Los Alamitos, Calif., 1987, pp. 345-357.
2. V.R. Basili and D.M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Trans. Software Eng.*, Nov. 1984, pp. 728-738.

3. H.D. Rombach and V.R. Basili, "A Quantitative Approach to Quality Assurance," *Informatik Spektrum*, June 1987, pp. 145-158.
4. B.W. Boehm, J.R. Brown, and M. Lipow, "Quantitative Evaluation of Software Quality," *Proc. Second Int'l Conf. Software Eng.*, CS Press, Los Alamitos, Calif., 1976, pp. 592-605.
5. F.J. Buckley and R. Poston, "Software Quality Assurance," *IEEE Trans. Software Eng.*, Jan. 1984, pp. 36-41.

Victor R. Basili is a professor and chairman of the Computer Science Department at the University of Maryland, College Park. He has been involved in the design and development of several software projects, including the SIMPL family of programming languages. He is currenly measuring and evaluating software development in industrial settings and has consulted with many agencies and organizations.

Basili has written more than 80 papers on the methodology, the quantitative analysis, and the evaluation of the software development process and product. He serves on the editorial boards of the *Journal of Systems and Software* and the *IEEE Transactions on Software Engineering*, and is a senior member of the Computer Society of the IEEE.

H. Dieter Rombach is an assistant professor of computer science at the University of Maryland, College Park. His research interests include software methodologies, measurement of the software process and its products, and distributed systems.

He received a BS in mathematics and an MS in mathematics and computer science from the University of Karlsruhe, West Germany, and a PhD in computer science from the University of Kaiserslautern, West Germany. He is a member of the Computer Society of the IEEE, ACM, the German Computer Society (GI), the Software Engineering Laboratory, and the University of Maryland Institute for Advanced Computer Studies.